Impact Factor:					
	ISRA (India) = 6.317	SIS (USA) = 0.912	ICV (Poland) = 6.630
	ISI (Dubai, UAE) = 1.582	РИНЦ (Russia) = 3.939	PIF (India) = 1.940
	GIF (Australia) = 0.564	ESJI (KZ) = 8.771	IBI (India) = 4.260
	JIF = 1.500	SJIF (Morocco) = 7.184	OAJI (USA) = 0.350

SOI: 1.1/TAS DOI: 10.15863/TAS

International Scientific Journal

Theoretical & Applied Science

p-ISSN: 2308-4944 (print) **e-ISSN:** 2409-0085 (online)

Year: 2022 **Issue:** 05 **Volume:** 109

Published: 30.05.2022 **http://T-Science.org**

Issue Article

Aleksey Aleksandrovich Ulianov
Peter the Great St. Petersburg Polytechnic University
Master's Student
Institute of Computer Science and Technology

Oleg Yurievich Sabinin
Peter the Great St. Petersburg Polytechnic University
Candidate of Engineering Sciences, Docent
Institute of Computer Science and Technology

DEVELOPMENT OF A DATA REPLICATION TECHNOLOGY IN REAL TIME USING ORACLE GOLDENGATE SYSTEM

Abstract: *The purpose of the article is to describe the development of real-time data replication technology using the Oracle GoldenGate system.*

Key words: *Data warehouse, Data replication, Online replication, Oracle GoldenGate.*

Language: *Russian*

Citation: Ulianov, A. A., & Sabinin, O. Y. (2022). Development of a data replication technology in real time using oracle goldengate system. *ISJ Theoretical & Applied Science, 05 (109)*, 927-941.

Soi: http://s-o-i.org/1.1/TAS-05-109-92 ***Doi***: crossref https://dx.doi.org/10.15863/TAS.2022.05.109.92

Scopus ASCC: *1700.*

РАЗРАБОТКА ТЕХНОЛОГИИ РЕПЛИКАЦИИ ДАННЫХ В РЕЖИМЕ РЕАЛЬНОГО ВРЕМЕНИ ПРИ ПОМОЩИ СИСТЕМЫ ORACLE GOLDENGATE

Аннотация: *Целью статьи является описание разработанной технологии репликации данных в реальном времени с использованием системы Oracle GoldenGate.*

Ключевые слова: *Хранилище данных, Репликация данных, Онлайн репликация, Oracle GoldenGate.*

Введение

Предметная область

Хранилище данных – это разновидность системы управления данными, оптимизированная для хранения структурированных данных с целью выполнения высокоскоростных SQL-запросов, обеспечивающих поддержку своевременной бизнес-аналитики.

Перед началом проектирования любого хранилища данных необходимо точно определить бизнес-требования, которые с его помощью будут решены, и очертить его сферы применения. Далее нужно определиться с логической и физической моделями разрабатываемого хранилища. Логическая модель отвечает за сами объекты и взаимосвязь между ними, а физическая – за выбор оптимального способа хранения, передачи и извлечения объектов, а также за процесс резервного копирования и восстановления. Стоит отметить, что хоть чаще всего конечным пользователям и нужны данные в обобщенном виде для различного рода анализа, а не в виде отдельных транзакций, но тем не менее при разработке хранилища нужно сразу закладывать резервные ресурсы для добавления новых возможностей и учитывать потенциальную потребность в расширении по мере развития продукта.

Таким образом, при проектировании хранилища данных обязательно нужно учитывать следующие факторы:

- специфику содержания данных;
- взаимосвязи внутри групп данных и между ними;

- системные среды обеспечения хранилища данных;
- необходимые преобразования над данными из источников;
- частоту обновления данных.

Современное хранилище данных обычно включает в себя следующее компоненты [3]:

- реляционную базу данных, которая заметно упрощает управление данными и предоставляет различные возможности для их дальнейшего анализа;
- решение для самостоятельного извлечения, преобразования и загрузки данных, которое является промежуточным этапом перед анализом данных;
- средства для статистического и глубинного анализа данных;
- средства для отчетности по хранящимся данным;
- разные параметры аналитики, которые упрощают использование данных без необходимости их перемещения;
- автоматизированное управление для упрощения выделения ресурсов, масштабирования и администрирования.

В классической архитектуре хранилищ данных выделяют три общих модели: виртуальное хранилище, витрина данных и корпоративное хранилище данных.

Для переноса данных из источников в хранилище данных можно использовать один из следующих способов: ETL (Extract, Transform, Load) и ELT (Extract, Load, Transform).

При подходе ETL из источников сперва извлекаются данные и хранятся во временной промежуточной базе данных. Далее осуществляются операции преобразования, которые нацелены на структурирование данных в такую форму, которая оптимально подходит для архитектуры системы хранилища данных. После этого структурированные данные переносятся в хранилище и готовы к анализу.

При подходе ELT данные извлекаются из источников и, минуя какие-либо промежуточные действия, сразу же загружаются в хранилище. Преобразование данных происходит уже непосредственно в системе хранилища данных при использовании различных инструментов из аналитики и бизнес-аналитики.

Хранилище с промежуточной областью наиболее часто используются в ситуациях, когда источники данных содержат разнородную информацию с множеством различных форматов и типов данных. Одна из разновидностей такой временной структуры хранения – это использование промежуточной витрины данных.

Постановка проблемы

Чаще всего данные в хранилище поступают из OLTP-систем – это системы со специальным способом организации, при котором обработка транзакций происходит в реальном времени; чаще всего они используются для ввода, структурированного хранения и обработки информации в режиме онлайн. Оба описанных ранее подхода загрузки данных не оптимально подходят для онлайн репликации из таких систем вследствие того, что при сборе данных из источников тратится время и зачастую большие ресурсы на поиск в источниках новых или измененных строк. А именно быстрота реагирования на те или иные действия пользователя сейчас является ключевым преимуществом в условиях конкурентного рынка. В наилучшем случае, как только произошла некая транзакция, ее данные сразу должны поступать в хранилище для возможности аналитики изучить их и предложить клиенту персонализированное спецпредложение в реальном времени.

Предложения по решению

Для решения вопроса доставки данных из источников в хранилище в реальном времени используется специальное высокопроизводительное программное обеспечение с технологией Change Data Capture (CDC). CDC в режиме онлайн захватывает изменения из журналов базы данных, тем самым заметно снижая нагрузку на источник и уменьшая объем передаваемых данных. В данной работе будет рассмотрена система Oracle GoldenGate (OGG).

Таким образом, чтобы объединить данные из разных источников и иметь всю актуальную информацию по ним в режиме реального времени, в данной статье будет приведено решение, включающее в себя способ разработки хранилища данных таким образом, чтобы успешно интегрировать в него средство репликации данных Oracle GoldenGate.

Цель и задачи исследования

Целью данной работы является исследование и разработка технологии репликации данных в режиме реального времени при помощи системы Oracle GoldenGate.

Для достижения сформулированной цели необходимо решить следующие задачи:

- провести обзор технологии Oracle GoldenGate и принципа ее работы;
- разработать концепцию по репликации данных из разных источников в режиме онлайн при помощи системы Oracle GoldenGate;
- разработать прототип описанной репликации данных;

- разработать хранилище данных для выбранной к работе БД;
- реализовать разработанную концепцию репликации данных на примере выбранной БД;

Обзор системы Oracle GoldenGate

OGG – это высокопроизводительное программное обеспечение, которое обеспечивает двустороннюю репликацию на основе журналов СУБД [7], а также использующееся для сбора данных из источников, их трансформации и доставки уже в структурированном виде в хранилище в реальном времени. С его помощью можно строить сложную отчетность.

Стоит отметить, что данная система может быть использована не только с СУБД Oracle, она интегрируема и с другими. Главной задачей CDC OGG является обеспечение потока данных о событиях в реальном времени из OLTP-источников на аналитическую платформу, причем данные перемещаются с высокой скоростью, низкой нагрузкой, минимальной задержкой и при этом сохраняя транзакционную целостность. После развертывания данного программного обеспечения на источнике начинают собираться изменения из redo-log- или archive redo-log-файлов и пересылаться по средствам передаточных trail-файлов на OLAP-сервер. Данный подход является весьма эффективным, так как используются исключительно журналы повторного выполнения; не происходит никаких лишних запросов и действий с источником. Помимо этого, появляется возможность получить все изменения данных в режиме онлайн и при вязать их к конкретной дате и времени, когда они произошли в базе данных.

Можно выделить следующие основные возможности, которые предоставляет CDC OGG:

- обеспечение постоянного сбора измененных данных и их доставка из OLTP-источников в единое хранилище данных;
- возможность выполнять задачи отчетности на отдельных серверах в режиме реального времени;
- интеграция оперативных данных между OLTP-системами в реальном времени.

Применение технологии CDC OGG обеспечивает следующие ключевые возможности и преимущества [7]:

- данные поступают в реальном времени;
- поддержка различных систем-источников;
- высокая надежность;
- транзакционная целостность;
- высокая производительность с минимальной нагрузкой на источники;
- интеграция с другим ПО;
- гибкая поддержка различных конфигураций;
- обнаружение и разрешение конфликтов;
- маршрутизация, сжатие и шифрование данных;
- возможность отложенной доставки по указанию пользователя.

Концепция разработанной технологии

Для успешного выполнения поставленной задачи был разработан следующий план действий:

- проанализировать суть данных, содержащихся в OLTP-источниках, и выделить бизнес-требования, по которым будет проводиться их анализ;
- проанализировать формат данных, который содержится в OLTP-источниках, и определить какие требуются преобразования над ними перед переносом;
- спроектировать хранилище данных в соответствии с выбранными бизнес-требованиями и исходя из выявленных преобразований над данными;
- при помощи технологии OGG настроить маршрутизацию данных из источников в хранилище в режиме онлайн.

Если же говорить о наиболее тонком моменте, а именно о способе разработки хранилища данных таким образом, чтобы успешно интегрировать в него средство репликации данных Oracle GoldenGate, была принята следующая концепция: для каждого источника на стороне хранилища создается своя схема, которая всегда находится в актуальном состоянии за счет OGG, а данные со всех источников объединяются в одной общей схеме при помощи работы триггеров на таблички схем источников. Визуально данная концепция представлена на рисунке 9.

Анализ реплицируемых данных

Начальным этапом перед разработкой любого хранилища является составление основных бизнес-требований, на которых будет строиться некий анализ данных для обеспечения ключевого преимущества компании. В качестве результата – создается список требований и примерный список аналитических функций, которые будут применены к данным на стороне хранилища. Опираясь на данный список, можно приступать к анализу данных с точки зрения того, каким наиболее эффективным методом их нужно будет хранить в хранилище данных.

Чем эффективнее пройдет данный этап при разработке, тем меньше в дальнейшем придется вносить правок в уже работающую систему, поэтому очень важно ориентироваться не только на оптимальное хранения данных для повышения производительности сложных аналитических запросов, но и закладывать некий функционал для потенциальных дальнейших правок по мере

использования. На этом этапе также берется в расчет и расширенный функционал программного обеспечения OGG, который позволяет указывать различные фильтры на данные, которые нужно забирать из источников. Можно выделить следующие основные условия для сборки данных из OLTP-систем:

- указать лишь определенный набор таблиц, из которых будут забираться данные;
- указать лишь определенные поля таблиц и лишь их данные будут передаваться;
- указать конкретные операции, при которых будет происходить перенос данных;
- указать конкретное условие к транзакциям и будут переданы данные только тех, которые удовлетворят этому условию;
- указать конкретные условия для значений столбцов и те строки, которые будут им удовлетворять – будут переданы, иные – нет.

Настройка репликации данных

При помощи системы OGG можно выделить следующий план действий для настраивания беспрерывной репликации данных:

1. На обеих сторонах репликации устанавливается соответствующая версия ПО OGG.
2. Как только происходит какая-либо транзакция в OLTP-системе источнике, происходит запись в redo-log-файл. При помощи агента OGG читает эту последнюю транзакцию, преобразует ее соответствующим образом и добавляет ее в конец специально созданного trail-файла.
3. Процесс pump видит, что данные транзакции дописались в конец соответствующего trail-файла, читает этот конец и передает его на сторону хранилища.
4. Данные из trail-файла переносятся по-обычному TCP/IP-протоколу, причем для удобства они могут шифроваться или сжиматься.
5. На стороне хранилища эти данные также записываются в trail-файл, где они промежуточно сохраняются.
6. Далее процесс replicat переносит данные транзакции из trail-файла в хранилище.

Разработка прототипа

Определившись с концепцией, следующем пунктом идет разработка прототипа репликации данных при помощи технологии OGG. По причинам того, что данная работа содержит в себе исследовательскую составляющую, то было принято решение прототип реализовывать для репликации Oracle – Oracle, а не MySQL – Oracle, которая будет использована позже для выбранной в качестве источника БД.

Первым делом нужно сымитировать две стороны, участвующие в передаче данных. Для этого были использованы два персональных компьютера, на каждом из которых была установлена СУДБ Oracle 19c и соответствующая версия системы OGG для Windows. Во время развертывания было принято решение создать подключаемую базу данных (PDB), поскольку в реальной ситуации вполне возможно, что к одному серверу будут подключены сразу несколько PDB и реплицировать данные нужно будет с каждой из них.

Таким образом, на каждой стороне было создано по своей PDB со следующими именами: unik – это БД-источник и orclpdb – целевая БД. В качестве реплицируемых данных была создана на каждой стороне схема GGTEST, а в ней табличка Student2, которая состоит из двух полей: stud_id – идентификатор студента и stud_name – имя студента. Для подключения к OGG использовалась утилита командной строки ggsci.exe.

Включение журналирования

Исходя из принципа работы ПО OGG, описанного ранее, ему необходимо, чтобы все изменения, связанные с данными, были отражены в журналах redolog. Для включения и выключения этой функции в БД Oracle есть специальный параметр log_mode, который может принимать значение либо «ARCHIEVELOG», либо «NOARCHIVELOG». При этом для того, чтобы его изменить, БД должна находиться на этапе mount для изменения файлов контроля. Также стоит отметить, что данный параметр меняется на уровне корневой БД, а не на уровне PDB.

Однако журналы Oracle в первую очередь оптимизированы для быстрого восстановления данных, а не репликации. В стандартной конфигурации они не хранят информацию по всем атрибутам изменяющихся таблиц. Но есть специальный процесс SUPPLEMENTAL LOGGING, который отвечает за запись дополнительной информации в журнал во время операций изменения. У него, в свою очередь, есть разные типы, позволяющие выбрать либо логировать дополнительную информацию при изменении любого столбца таблички, либо определить набор столбцов, изменения которых нужно отслеживать, и разные уровни:

- Minimal - в этом режиме БД логирует дополнительный объем данных, который необходим для идентификации, группировки и соединения операций Redo, связанных с DML изменениями; не добавляет значительную нагрузку на БД;
- ALL - безусловно заставляет БД записывать в журналы состояние до изменения для всех

столбцов в изменяемой строке (за исключением LOBs, LONGS, и ADTs);

- PRIMARY KEY - безусловно (даже если первичный ключ не меняется) заставляет базу записывать в журналы состояние до изменения для первичного ключа в изменяемой строке;
- UNIQUE - при условии, что изменяется столбец, входящий в уникальный или bitmap индекс, логируется состояние до изменения для всех столбцов, принадлежащие этому индексу;
- FOREIGN KEY — при условии, что изменяет столбец, входящий в FK, логируется состояние до изменения для всех столбцов FK.

В рамках тестирования на небольших данных мной был выбран уровень ALL.

Теперь, когда мы подключили необходимый функционал для полноценного и необходимого логирования в журналы, нужно изменить еще один параметр типа Boolean в БД, который указывает СУБД использовать или нет сервисы, поставляемые OGG. Это параметр enable_goldengate_replication. По своей сути он точно также указывает включить дополнительное логировние в журналы, но без его установления в TRUE, на моменте репликации будет возникать ошибка. Для этого используем следующую команду (ее нужно также выполнять на уровне корневой БД источника): alter system set enable_goldengate_replication = true scope = both.

Создание пользователя для работы OGG

Теперь для корректной работы с системой OGG нужно создать специального пользователя и выделить ему все необходимые привилегии. Данный этап нужно выполнить на обеих базах данных.

Одна из особенностей глобальных пользователей в системе Oracle, которые не относятся к группе системных, это то, что они должны начинаться с префиксом «c##». Таким образом, на обеих БД были созданы пользователь c##ggowner и табличное пространство goldengate для него. Дальше начался процесс определения нужных привилегий для него.

Помимо стандартных привилегий на создание сессий, таблиц, выделение ресурсов и т.д., этому пользователю необходимо выделить следующие неочевидные специфические привилегии:

- в папке, куда был установлен goldengate, нужно запустить sqlplus под пользователем SYSDBA и скрипт role_setup.sql. Это встроенный стандартный скрипт, который создает специальную роль GGS_GGSUSER_ROLE с минимальным набором привилегий, необходимых пользователю для работы с Goldengate, и включает переданного пользователя в эту роль;
- выполнить команду вида: exec dbms_goldengate_auth.grant_admin_privilege('c##ggowner', 'apply', container=>'orclpdb'). Эта команда выделяет все необходимые привилегии для взаимодействия GoldenGate и PDB, которая указывается через параметр container (можно указать all для всех PDB, относящихся к корневой системе);
- выполнить в интересующей вас PDB команду вида: GRANT SELECT ANY DICTIONARY TO c##ggowner. Она также оказалась необходимой, иначе в дальнейшем будут ошибки при репликации.

Настраивание непрерывной репликации данных

Для начала нужно на обеих базах данных запустить эту утилиту и выполнить команду «Edit params ./GLOBALS». Она создает глобальный файл конфига для работы агента Oracle GoldenGate, в котором можно указать глобальные параметры. В моем случае я остановился на указании схемы, которую агент будет использоваться для поддержания корректной репликации. В этом качестве будет использована схема пользователя c##ggowner, который был создан на этапе ранее.

Далее при помощи команды «dblogin userid ***, password ***» мы подключимся к корневой базе данных на стороне-источнике. Теперь агент GoldenGate связан непосредственно с БД и можно использовать весь функционал.

Сам алгоритм репликации можно разбить на 3 процесса: 2 процесса извлечения EXTRACT на стороне источника и один процесс репликации REPLICAT на стороне целевой БД.

Для начала разберемся с процессами EXTRACT:

- первый процесс называется EXTRACT и отвечает за выборку данных из таблиц, которые выбраны для репликации. Для этого нужно создать файл со следующими параметрами:
  - dboptions host, connectionport, sourcedb, userid, password – параметры соединения с базой данных: имя хоста, порт (на котором работает БД), имя БД, логин и пароль пользователя (под которым будут читаться данные); в моем случае еще была указана конкретная PDB база данных, в которой содержится интересующая меня табличка, но стоит отметить, что можно использовать таблички из разных контейнеров, но с прямым их указанием перед названием схемы;

  - Discardfile – параметр, указывающий путь к отчету с ошибками при сбоях;
  - Table – указывает на таблицу, которая будет включена в репликацию;
- второй процесс называется PUMP и отвечает за настройку подключения к хосту целевой базы данных. Для это процесса также нужно создать файл с параметрами, который содержит:
  - Rmthost – ip-адрес хоста хранилища;
  - Mgrport – порт, на котором работает агент OGG на целевой стороне;
  - Rmttrail – параметр для указания расположения файлов трассировки с реплицируемыми данными на стороне удаленного хоста;
  - Table – указывает на таблицы, которые будут включены в репликацию на стороне БД источника;

Теперь разберем процесс REPLICAT, который устанавливается для целевой базы данных. Для него также создается файл с параметрами, в котором указывается:

- sourcedb, userid, password – логин и пароль для подключения к целевой базе данных, причем в случае использования PDB, нужно указать именно ее для подключения;
- соотношение табличек для репликации: через MAP указывается табличка на стороне источника с указанием PDB и схемы, через TARGET указывается табличка на стороне целевой БД с указанием PDB и схемы.

Для корректной работы процесса EXTRACT-EXTRACT необходимы 3 команды:

- edit param «name» – создание файла конфигурации;
- ADD EXTRACT «name», TRANLOG, BEGIN NOW – создание процесса и регистрация его на исполнение для менеджера процессов, где:
  - TRANLOG – указывает, что в качестве источника данных будут выступать транзакционные логи.
  - BEGIN NOW – начать захват данных сразу после запуска extract.
- ADD EXTTRAIL “path”, EXTRACT “name”, MEGABYTES “value” – создание канала для передачи данных, где:
  - EXTTRAIL – указывает, где будут храниться реплицируемые данные.
  - MEGABYTES – допустимый размер файлов.

Для корректной работы процесса EXTRACT-PUMP необходимы 3 команды:

- edit param «name» – создание файла конфигурации;
- ADD EXTRACT “name”, EXTTRAILSOURCE “path”, BEGIN NOW – создание процесса и регистрация его на исполнение для менеджера процессов, где:
  - EXTTRAILSOURCE – указывает, где будут храниться реплицируемые данные;
  - BEGIN NOW – начать захват данных сразу после запуска pump.
- ADD RMTTRAIL “path”, EXTRACT “name”, MEGABYTES “value” – создание канала для передачи данных, где:
  - RMTTRAIL – указывает, где будут храниться реплицируемые данные на стороне хранилища;
  - MEGABYTES – допустимый размер файлов.

Для корректной работы процесса REPLICAT необходимы 2 команды:

- edit param «name» – создание файла конфигурации репликации;
- add replicat repora, EXTTRAIL ./dirdat/r1, BEGIN NOW, NODBCHECKPOINT – создание процесса и регистрация его на исполнение для менеджера процессов, где:
  - EXTTRAIL – указывает, где будут храниться реплицируемые данные на стороне хранилища; для replicat это значение должно соответствовать значению RMTTRAIL для pump;
  - BEGIN NOW — начать захват данных сразу после запуска replicat;
  - NODBCHECKPOINT – контрольные точки не будут храниться в целевой базе данных.

По причине использования на источнике PDB и ссылок на нее в файлах конфигурации процессов EXTRACT, нужно произвести еще одну команду по их связыванию между собой: register extract “name” database container(“PDB”). Иначе агент OGG при подключении к корневой БД не будет видеть PDB, к которой идет обращение, и будет возникать ошибка во время репликации.

Теперь еще осталось напрямую указать агенту отслеживание транзакций для реплицируемых табличек и можно запускать созданные сервисы. Это делается на стороне источника при подключении к корневой БД с явным указанием PDB, таблички или схемы следующим образом: add trandata “pdb”.“schema”.“table”.

Теперь все готово для запуска трех созданных процессов. На стороне источника нужно использовать команду «start extract» для запуска созданных процессов, а на стороне источника «start replicat». Текущий статус процессов можно посмотреть при помощи команды «info all». На рисунке 1 показаны результаты этой команды на стороне источника, а на рисунке 2 – на стороне целевой БД.

```
GGSCI (LAPTOP-L3SBJED6 as c##ggowner@orcl/CDB$ROOT) 5> info all

Program     Status      Group       Lag at Chkpt  Time Since Chkpt

MANAGER     RUNNING
EXTRACT     RUNNING     EXT1        00:00:00      00:00:09
EXTRACT     RUNNING     PUMPORA     00:00:00      134:39:09
```

Рисунок 1 – Статус процессов на источнике

```
GGSCI (NB2434 as c##ggowner@orcl/ORCLPDB) 10> info all

Program     Status      Group       Lag at Chkpt  Time Since Chkpt

MANAGER     RUNNING
REPLICAT    RUNNING     REPORA      00:00:00      00:00:01
```

Рисунок 2 – Статус процесса в хранилище

Далее для тестирования реализованного прототипа был написан простой скрипт для вставки 8 новых записей в табличку student2 на стороне источника. По окончанию его выполнения при помощи команды «stats» можно посмотреть статистику по интересующим процессам. На рисунке 3 показаны результаты этой команды для процесса EXTRACT-EXTRACT, по которым видно, что все 8 вставившихся операций успешно были записаны в трассировочные файлы.

```
GGSCI (LAPTOP-L3SBJED6) 9> stats ext666

Sending STATS request to EXTRACT EXT666 ...

Start of Statistics at 2022-02-06 03:10:45.

Output to ./dirdat/2/e1:

Extracting from UNIK.GGTEST.STUDENT2 to UNIK.GGTEST.STUDENT2:

*** Total statistics since 2022-02-06 03:10:20 ***
        Total inserts                                       8.00
        Total updates                                       0.00
        Total deletes                                       0.00
        Total upserts                                       0.00
        Total discards                                      0.00
        Total operations                                    8.00
```

Рисунок 3 – Статистика процесса EXTRACT

Разработка хранилища данных

Перед разработкой хранилища данных нужно иметь принципиальное понимание количества источников и их специфики. В рамках данной работы будет имитация работы двух реальных объектов, которые контролируют доступ на определенную территорию и заносят информацию в БД о транспортных средствах и владельцах. Оба они имеют идентичную структуру БД, поэтому при разработке хранилища достаточно определить подход к взаимодействию с одним источником, а все остальные будут добавлены по аналогии. На стороне источников будет использована СУБД MySQL, а на стороне хранилища СУБД Oracle.

Здесь стоит остановиться и отдельно поговорить о том, как будут храниться данные с разных источников и обрисовать словесную модель хранилища, особенно учитывая специфику обновления данных в режиме онлайн.

Для каждого объекта в целевой базе данных будет создана своя схема, в которой логическая модель будет полностью совпадать с единообразной моделью источника. Это необходимо для корректного обеспечения обновления всех данных в режиме реального времени при помощи системы OGG. Далее необходимо объединить данные с разных источников и структурировать их, чтобы была возможность извлечь из них бизнес-смысл. Для этого будет создана схема icv_data, которая по своей сути будет исполнять и функции витрины данных и ведения определенной исторической политики. Однако не каждое обновление несет в себе бизнес-смысл и не каждые таблички на разных источниках необходимо объединять. Например, на каждом объекте присутствуют таблицы, которые хранят только тип автомобиля, тип корпуса, цвета транспортных средств и т.д. Их не нужно объединять, они хранят полностью идентичные данные, занимают мало места, достаточно создать одну такую таблицу в схеме icv_data. Эта схема будет также основана на

исходной модели источника, однако будут некоторые новые сущности и большинство табличек будут дополнительно хранить ряд служебных полей:

- новые таблицы:
  - Src_Sources – таблица для уникального определения каждого источника;
  - Audit_Log_Journal – таблица для хранения всех изменений, происходящих с данными.
- новые представления:
  - Customers_Pass_V – представление для вывода всех посещений обслуживаемых территорий по всем автовладельцам;
  - Source_Pass_V – представление для вывода частоты посещений всех обслуживаемых объектов.
- дополнительные столбцы:
  - Src_Source_Id – поле для определения источника, с которого пришла строка;
  - Active_Flg – поле, указывающее содержится ли данная строка на стороне источника;
  - Create_Dttm – поле, хранящее время первого прихода записи со стороны источника;
  - Last_Update_Dttm – поле, хранящее время последнего изменения записи на источнике.

Конечно же, при проектировании хранилищ данных ключевую значимость несут требования, выдвигаемые заказчиком. Поскольку в данном случае их нет, то мной они были сформулированы самостоятельно. Как уже было сказано ранее, основная ценность этих данных – информация о транспортных средствах и о частоте их посещения обсуживаемых объектов. На основании этого были выделены следующие требования:

- возможность анализа исторических данных, отслеживание изменений по конкретному проекту (таблица Audit_Log_Journal);
- возможность анализа частоты посещений автовладельцами всех объектов (представление Customers_Pass_V);
- возможность анализа частоты посещений по каждому объекту (представление Source_Pass_V).

Таким образом, таблицы в схеме icv_data можно разбить на 3 категории:

- видоизмененные исходные таблицы, которые будут хранить объединенную информацию со всех источников и очень часто обновляться;
- исходные таблицы, хранящие небольшую однотипную информацию, которую не требуется объединять в разрезе источников; обновляются такие таблицы крайне редко;
- отдельная таблица Audit_Log_Journal, в которой будут отображаться все изменения, которые происходят с данными.

Осталось разобраться с процессом заполнения таблиц в схеме icv_data. Как было сформулировано ранее, все они так или иначе напрямую зависят от изменений в исходных схемах на стороне хранилища данных. Их репликацию в режиме реального времени поддерживает OGG, который не имеет возможности дополнительно еще и обновлять таблицы других схем. Однако, эту проблему можно эффектно решить при помощи создания триггеров для каждой реплицируемой таблицы. Триггер – это особая хранимая процедура, которая автоматически вызывается при определенных в ее условии DML-командах. Она поддерживает широкий спектр функционала PLSQL и, что самое главное, позволяет обращаться как к версии строки до изменения, так и после. В разрезе этого функционала была написана функция для сравнения двух значений, в которую будет передаваться значение каждого атрибута до и после изменений и в случае их неравенства, в качестве возвращаемого значения будет передаваться название данного атрибута. Данная функция имеет три версии для строчных значений, числовых и дат. Этот инструментарий необходим для поддержания исторической политики и возможности увидеть, что конкретно изменялось по какому-либо изменению, пришедшему из источника. Все это будет отображено в таблице Audit_Log_Journal. Однако не все изменения несут в себе ключевую бизнес-ценность, поэтому сравнение можно настроить не по всем атрибутам, а выборочно. При удалении же будут логически закрываться соответственные строки через служебное поле Active_Flg в таблицах схемы icv_data. В случае же вставки новых строк сравнение вызывать не нужно, можно сразу добавлять их в соответствующие сущности схемы icv_data.

Исходя из всего вышесказанного, для его реализации потребуется также разработать две вспомогательные процедуры, которые являются автономными транзакциями, что позволяет им фиксировать все изменения, происходящие внутри них, не дожидаясь коммита внешней транзакции. Первая – это вставка всех изменений в журнал аудита Audit_Log_Journal, она является универсальной для всех таблиц. Вторая – типовая процедура, которая переносит все изменения из источника в соответствующую таблицу схемы icv_data. Ее необходимо определить на каждую таблицу.

Естественно, создавать такие триггеры и вспомогательные процедуры вручную – плохой подход с точки зрения экономии человеко-часов. Тем более, что СУБД Oracle поддерживает весь

функционал динамического SQL. С его помощью можно пройтись по всем схемам источников и на основании метаданных их табличек сгенерировать код для создания всех необходимых PLSQL объектов. Дальше уже можно для каждой таблички модифицировать стандартный триггер в соответствии с определенными на нее бизнес-требованиями.

Создание схем данных источников

Изначальный скрипт для развертывания базы данных источников был написан под СУБД MySQL, в связи с чем требовал существенных модификаций. Ключевые изменения:

- изменение типов данных, которые поддерживаются СУБД Oracle
- переработка синтаксиса sql-кода в соответствии со стандартами СУБД Oracle;
- удаление вторичных ключей – в первую очередь они обеспечивают целостность данных на источниках, поэтому в хранилище они уже поступают в таковом виде; более того, в связи с репликацией данных в реальном времени, есть небольшая вероятность, что порядок изменений не будет соответствовать логике вторичных ключей или целостность отправленных транзакций будет нарушена, а это чревата ошибкам обновления уже на стороне хранилища.

Таким образом, был создан универсальный скрипт для создания схемы для любого типового источника данных. Он был успешно использован для создания схемы src_01. В дальнейшем эта схема будет использована для репликации данных с первого (тестового) контура. Также, стоит отметить, что для его использования, необходимо создать соответствующего пользователя и наделить его необходимыми привилегиями.

Следующим шагом было развертывании схемы icv_data. Для этого был использован тот же скрипт, но с дополнительными модификациями, которые были описаны ранее.

Создание вспомогательных процедур и функций

Итак, еще раз обозначим виды процедур и функций, которые необходимы для выбранного способа репликации данных:

- процедура для введения исторической политики, которая вставляет изменения в таблицу audit_log_journal;
- процедуры для каждой таблицы в схеме источника, которые на основании изменений в них вставляют записи в соответствующие таблицы схемы icv_data;
- функции, которые сравнивают два значения любого атрибута любой таблицы (до и после изменений);

Далее более подробно рассмотрим их процесс создания. Отдельно отметим, что для их полноценной работы и правильного использования необходимо выделить как системные привилегии, так и пользовательские, которые, к примеру, позволяют одному пользователю выполнять PL/SQL-объекты другого.

Процедура первого вида называется AUDIT_JOURNAL_LOG_PRC и у нее можно выделить ее ключевые особенности:

- она является автономной, что позволяет ей закоммить изменения, которые были внутри нее, вне зависимости от того, как завершилась внешняя транзакция; это осуществлено через использование PRAGMA autonomous_transaction;
- она содержит информацию про таблицу, с которой произошло изменение, владельца этой таблицы, атрибуты, которые изменялись, время изменения и поле obj_id, которое содержит значение либо только первичного ключа этой таблицы, либо первичного ключа и кода источника, для возможности отсмотреть историю изменения по объекту;
- Содержится в схеме icv_data.

Рассмотрим конкретный пример процедуры второго вида с названием Customers_audit. Можно выделить ее следующие ключевые особенности:

- она также является автономной через использование PRAGMA autonomous_transaction;
- входные параметры соответствуют атрибутам таблички, к которой она относится; соответственно, количество входных параметров равно количеству атрибутов таблички + атрибут SRC_SOURCE_ID, который соответствует схеме источника + флаг i_DML_FLG, который указывает, какое именно изменение происходит: ‘I’ – Insert, ‘U’ – Update, ‘D’ – Delete;
- если i_DML_FLG = ‘I’, то вставляется новая строка с соответствующим идентификатором источника;
- если i_DML_FLG = ‘U’, то строка обновляется новыми значениями, ее поиск идет по первичному ключу и идентификатору источника;
- если i_DML_FLG = ‘D’, то строка логически закрывается путем обновления флага ACTIVE_FLG на значение 'N'.

Далее, удачно скомпилировав процедуру Customers_audit и убедившись в ее работоспособности, была написана процедура для создания скрипта для создания процедуры этого вида для каждой таблички схемы источника. Она

написана с использованием функционала динамического SQL. В ней через внешний цикл перебираются все таблички схемы источника через системное представление user_tables и через внутренний цикл по системному представлению user_tab_columns перебирает все их атрибуты. В результате заполняется переменная l_script типа CLOB, в ней содержится plsql-код для генерации процедур, который можно сразу же выполнить через команду EXECUTE IMMEDIATE, однако мне показался удобнее немного другой подход, при котором эта переменная вставляется во вспомогательную табличку script_table после каждого прогона внешнего цикла. Так, перед запуском можно еще раз все проверить и создавать их как последовательно, так и параллельно.

Функций, описанных ранее, всего 3 и они используются для сравнения следующих типов данных: number, date, varchar2. Они принимают на вход название атрибута, его значения до и после изменения и символ, которым можно разделять значения. На выходе в случае, если переданные значения отличаются, то передается название атрибута + символ разделения, если значения не отличаются, то отдается null. Также, стоит отметить, что поскольку входные атрибуты могут быть разного типа, а название должно быть одинаковым, то они были помещены в пакет CHECK_ATTR_API.

Создание триггеров для переноса данных

Итак, еще раз определимся зачем нам нужны триггеры на таблички и как они будут использованы. На каждый источник будет отведена своя схема, каждая ее табличка будет актуализироваться в соответствии с изменениями с источника в режиме реальном времени при помощи системы Oracle GoldenGate. Далее, чтобы иметь все актуальные данные со всех источников вместе, их нужно объединить. Для этого создана схема icv_data, в которую и нужно доставить данные. Именно для этого и используются триггеры, которые реагируют на каждое изменение.

Теперь более подробно рассмотрим реагирования на разные изменения. Для примера возьмем снова табличку Customers.

Если это вставка, то нужно через процедуру CUSTOMERS_audit вставить новые значения в соответствующую табличку схемы icv_data путем вызова с флагом i_DML_FLG = 'I'. Также, для ведения исторической политики вызывается процедура audit_journal_log_prc, куда вставляется запись о вставке.

Если это обновление, то первым делом для каждого атрибута вызывается функция check_upd_attr_value_fnc и в результате заполняется строковая переменная, которая содержит все изменившиеся атрибуты. В случае, когда эта переменная не пустая, вызывается процедура CUSTOMERS_audit с флагом i_DML_FLG = 'U', где фиксируется список изменившихся атрибутов. После этого вызывается процедура audit_journal_log_prc для фиксации обновления данной строки.

Если это удаление, то вызывается процедура CUSTOMERS_audit с флагом i_DML_FLG = 'D', которая логически закрывает соответствующую запись в схеме icv_data, и вызывается процедура audit_journal_log_prc, которая фиксирует удаление этой записи.

Все вышесказанное было реализовано и создан триггер SRC_01.CUSTOMERS$audit_trg. После этого была проведена проверка его работоспособности. Для этого в табличку SRC_01.CUSTOMERS была вставлена одна запись, после чего для нее же было произведено обновление атрибута lastname. Как итог, вставлена была одна запись в табличку источника src_01, 2 записи в табличке аудита audit_log_journal и 1 запись в схеме icv_data с Active_Flg = ‘Y’.

Теперь, убедившись, что разработанная концепция репликации данных внутри БД хранилища, реализована правильно, нужно было создать скрипт, который создаст триггер такого вида для каждой таблички схемы источника. Для этого была написана процедура, которая по аналогии с ранее описанным способ создания скрипта для генерации вспомогательных процедур для всех таблиц, также использует динамический sql, проходится по системным представлениям user_tables и user_tab_columns, собирает код создания каждого триггера внутри переменной l_script типа CLOB и заносит это значение в табличку script_table.

После того, как все вышеописанное было реализовано для схемы src_01, был выгружен скрипт создания этой схемы со всеми объектами и привилегиями. Этот скрипт можно теперь использовать для создания уже готовой схемы для новых источников, нужно лишь поменять имя src_01 на новое. В частности, именно так была создана src_02 для второго источника данных.

Реализация технологи репликации данных при помощи OGG

Имея представление о настройке репликации данных в режиме реального времени при помощи OGG, полученное во время разработки прототипа, и развернутое хранилище данных, можно было приступать к финальному пункту в плане реализации. Для этого, как уже было сказано, необходимо было иметь 2 сервера с развернутыми на них базами данных, которые будут имитировать работу реальных объектов. На первом источнике БД называется icv_21, на

втором – icv_20; в хранилище они будут связаны со схемами src_01 и src_02 соответственно.

Также, необходимо отметить, что OGG будет работать только с движком хранения InnoDB и кодировкой utff8mb4 в MySQL. Оба эти параметра можно указывать по умолчанию как при создании БД, так и для конкретных табличек. Кодировку можно менять уже после создания объекта при помощи следующей команды: ALTER DATABASE db_name CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci.

Далее будет поэтапно расписаны все действия, которые необходимо осуществить для установки соединения между БД источника и хранилищем.

Как уже было отмечено не раз ранее, для работы OGG необходимо, чтобы все изменения, связанные с данными, были отражены в журналах redolog. На этапе разработки прототипа я уже рассказывал, как это сделать для БД Oracle, теперь же в качестве источника выступает СУБД MySQL.

В данной СУБД этот функционал определяется через параметр log-bin и также определяется на уровне корневой БД. Данный параметр необходимо задать в файле конфигурации, который в зависимости от ОС называется либо my.ini, либо my.cnf. Он определяет место размещения бинарных журналов повтора. Однако для корректной работы GolgenGate этого мало, в этом же файле конфигурации необходимо определить еще и следующие параметры:

- log-bin-index – определяет место размещения индексов бинарных журналов; по умолчанию, они находятся в той же директории, что и бинарные журналы;
- binlog-format – формат журналирования; бывает трех видов:
  - ROW – в binlog записываются измененные бинарные данные; OGG будет работать только с этим форматом;
  - STATEMENT – репликация, которая основывается на анализе выполняемых запросов;
  - MIXED – промежуточный формат, при котором по умолчанию используется STATEMENT с переключениями в ROW.

По сравнению с СУБД Oracle в MySQL нет таких строгих ограничений с созданием глобальных пользователей. Более того, подключаться можно и под стандартным корневым пользователем root. Как итог, на первом источнике – ggsci, на втором – root, на хранилище – c##ggowner (который ранее был создан в рамках разработки прототипа).

Теперь создадим файл с параметрами для экземпляра OGG на каждом из трех хостов. Это делается при помощи команды «edit params mgr».

В моем случае я остановился на следующих параметрах:

- PORT – порт менеджера процессов Oracle GoldenGate, использующийся для взаимодействия с удаленными процессами;
- DYNAMICPORTLIST – перечень доступных портов, которые локальные процессы Oracle GoldenGate могут использовать для связывания с удаленными процессами Oracle GoldenGate на других машинах;
- AUTOSTART – параметр, позволяющий автоматически запускать процессы Oracle GoldenGate после запуска менеджера процессов;
- AUTORESTART – параметр для автоматического перезапуска процессов Oracle GoldenGate при сбоях по сети;
- RETRIES – максимальное количество перезапусков;
- WAITMINUTES – время ожидания в минутах от аварии до перезапуска;
- STARTUPVALIDATIONDELAY – время ожидания (в секундах) до проверки состояния процесса GoldenGate; в случае, если после этого времени процесс не ответил, то регистрируется ошибка;
- PURGEOLDEXTRACTS – позволяет очищать обработанные файлы для того, чтобы они не потребляли много дискового места.

Далее нам все также нужно на хранилище создать процесс REPLICAT, а на каждом источнике создать 2 процесса EXTRACT (EXTRACT и PUMP).

Сами эти файлы по своей структуре не сильно отличаются от тех, что я уже описывал при разработке прототипа, но все же определенные отличия имеются:

- процесс EXTRACT-EXTRACT:
  - cachemgr cachesize – размер кэша памяти, который может использовать процесс extract;
  - TRANLOGOPTIONS ALTLOGDEST – параметр для указания расположения файла индексов бинарных логов MySQL;
  - Table – все таблички схемы источника для репликации указываются через *.
- процесс EXTRACT- PUMP:
  - Table – все таблички схемы источника для репликации указываются через *.
- процесс REPLICAT:
  - Assumetargetdefs – параметр, указывающий на то, что структура таблиц источника и целевой базы схожи.
  - SETENV – задаем переменную окружения Windows; в частности,

указываем кодировку через NLS_LANG;

- параметр DBOPTIONS NOSUPPRESSTRIGGERS указывает на то, что триггеры, определенные на участвующие в репликации таблицы целевой БД, должны выполняться;
- соотношение табличек для репликации: через MAP указывается схема + * на стороне источника, через TARGET указывается схема + * на стороне хранилища.

С точки зрения синтаксиса по корректному добавлению этих процессов в менеджере OGG различий нет: для двух EXTRACT – 3 команды, для REPLICAT – 2 команды.

Теперь все готово для запуска всех созданных процессов. На стороне источнике используем команду «start extract»; на стороне хранилища – команду «start replicat». Текущее состояние процессов можно вывести при помощи команды info all. Данные команды представлены на рисунках 4, 5 и 6.

По статусам видно, что все процессы были созданы корректно.

Тестирование репликации данных

План тестирования был следующим: протестировать корректность репликации данных при вызове на стороне источников всех закладываемых при разработке DML команд – Delete, Update, Insert. Более того, для увеличения нагрузки на систему изменения в данных на обоих источниках запускались одновременно. Файл дампа был устроен стандартным образом и через него можно было проверить только команду Insert, пусть и достаточно внушительную. По этой причине для проверки изменения данных через команды Delete и Update было использовано ручное обновление.

Количественное сравнение было произведено сразу после загрузки обоих файлов дампа. Для этого было сперва зафиксировано количество записей в табличках на источниках, а потом уже результат репликации данных в соответственные схемы на стороне хранилища данных. Все эти опорные значения занесены с таблицы 1 и 2.

По результатам этой проверки можно заметить, что отличаются только те таблички, которые при разработке хранилища квалифицировались как те, которые хранят мало информации и обновляются крайне редко (такие как тип кузова машины, его цвет, тип доступа и тд). По этой же причине они не вошли в файлы дампа, которые был использованы. Более того, в схемах src хранить эту информацию даже и не обязательно, это было бы излишние хранение данных. Единственное исключение – это табличка customers из первого источника: на источнике – 89, в хранилище – 91. Это связано с тем, что в схеме src_01 уже хранились две тестовые записи, которые были получены ранее, при тестировании прототипа.

В рамках ручного тестирования было произведено обновление и удаление записей таблички analyzer_passages на стороне первого источника. В рамках тривиального скрипта в этой табличке сперва обновляется все 200 227 записей, а затем 14 810 из них удаляются.

На рисунке 7 приведены результаты команды stats процесса EXTRACT, на нем видны все описанные команды: сперва вставка всех записей при использовании дампа, затем обновления столбца Brand и удаление по условию на столбец PlateRegion

Теперь осталось проверить, что в разрезе схем src_001 и icv_data эти записи обновились соответствующим образом. Это можно сделать через запрос к журналу аудита. На рисунке 8 показана статистика по всем изменениям, произошедшим с табличкой analyzer_passages.

Выводы

В современном мире, где борьба за потребителя обострена до предела, скорость реагирования на какое-либо его действие играет важнейшую роль. Чем быстрее информация сможет дойти до аналитического ядра компании, тем быстрее получится сформировать для конкретного потребителя свое специализированное предложение.

В данной работе было исследовано решение этой проблемы при помощи использования технологии репликации данных в режиме реального времени Oracle GoldenGate. При чем была разобрана не только часть с передачей данных, но и часть с хранением и объединением информации в одном общем контуре – хранилище данных.

В результате была не только предложена общая стратегия решения заявленной проблемы, но и реализовано частное решение для конкретной БД.

Технология показала свою работоспособность, быстродействие и стойкость к потере данных. Во время тестирования нагрузка на нее была сопоставима с реальной. Более того, количество изменений и транзакций было даже больше среднестатистического в области применения рассмотренной БД.

Таблица 1. Количественное сравнение записей в источнике 1 и ее схеме в хранилище

Таблица	**Источник icv_21 (кол-во записей)**	**Хранилище src_01 (кол-во записей)**
Analyzer_Plate_Recognitions	206 138	206 138
Analyzer_Passages	200 227	200 227
Analyzer_Passage_Vehicle_connection	82 819	82 819
Analyzer_Emergency_Recognitions	44 342	44 342
Vehicle_Passages	33 584	33 584
Accesses	10 519	10 519
Accesses_Vehicle	10 519	10 519
Customers	89	91
Vehicle_Permissions	85	85
Vehivles	85	46

Таблица 2. Количественное сравнение записей в источнике 2 и ее схеме в хранилище

Таблица	**Источник icv_20 (кол-во записей)**	**Хранилище src_02 (кол-во записей)**
Analyzer_Plate_Recognitions	160 607	160 607
Analyzer_Passages	159 405	159 405
Analyzer_Emergency_Recognitions	73 029	73 029
Analyzer_Passage_Vehicle_connection	63 342	63 342
Vehicle_Passages	52 120	52 120
Analyzer_Cab_Recognitions	27 540	27 540
Accesses	25 695	25 695
Accesses_Vehicle	23 550	23 550
Vehicle_Permissions	153	153
Vehivles	151	151
Customers	107	107

Рисунок 4 – Процессы на 1-м источнике

```
GGSCI (DESKTOP-NFM39JL) 17> info all

Program     Status      Group       Lag at Chkpt  Time Since Chkpt

MANAGER     RUNNING
EXTRACT     RUNNING     EXT01       00:00:00      00:00:00
EXTRACT     RUNNING     PUMP01      00:37:17      00:00:04
```

Рисунок 5 – Процессы на 2-м источнике

```
GGSCI (LAPTOP-L3SBJED6) 6> info all

Program     Status      Group       Lag at Chkpt  Time Since Chkpt

MANAGER     RUNNING
REPLICAT    RUNNING     REP01       00:00:00      00:00:04
REPLICAT    RUNNING     REP02       00:00:00      00:00:01
```

Рисунок 6 – Процессы на стороне хранилища

```
From Table icv_21.analyzer_passages:
          #                 inserts:     200227
          #                 updates:     200227
          #                 deletes:      14810
          #                 upserts:          0
          #                discards:          0
```

Рисунок 7 – Статистика процесса Extract по всем изменениям

```
19 select t.audit_event_type_code, count(*)
20 from   icv_data.audit_log_journal t
21 where  t.table_name = 'ANALYZER_PASSAGES' and t.table_owner = 'SRC_01'
22 group by t.audit_event_type_code
23 order by 1 desc;
```

Query Result

SQL | All Rows Fetched: 3 in 0,245 seconds

	AUDIT_EVENT_TYPE_CODE	COUNT(*)
1	UPDATE	200227
2	INSERT	200227
3	DELETE	14810

Рисунок 8 – Проверка изменений таблицы Analyzer_Passages по журналу аудита

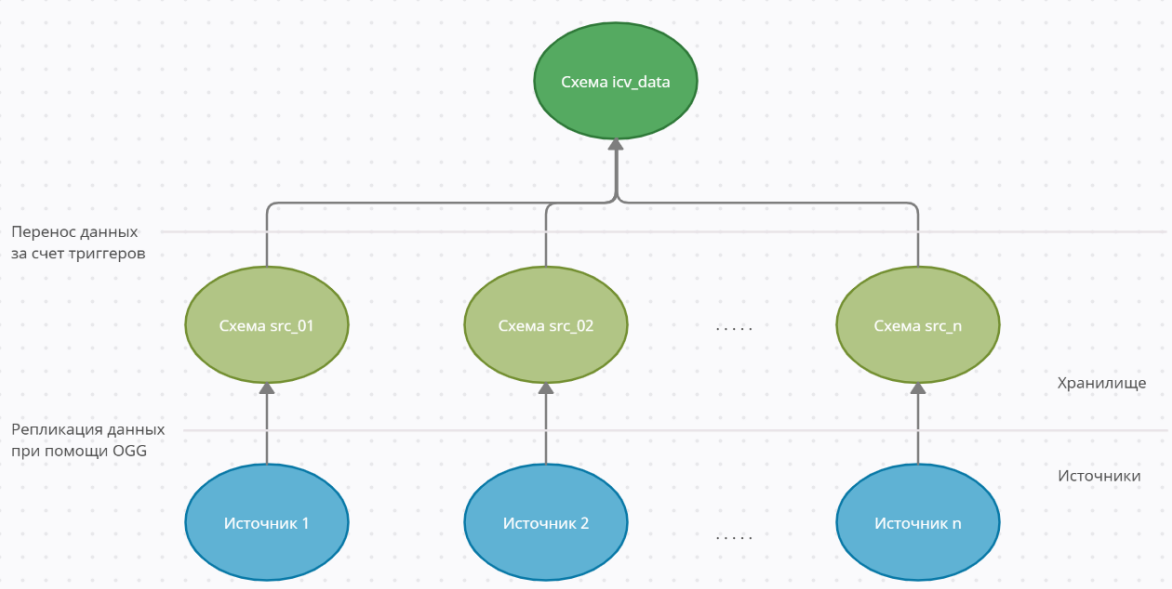

Рисунок 9 – концепция по переносу и объединению данных в режиме реального времени

References:

1. (n.d.). *Golden Gate: how to replicate terabytes per hour, or the experience of using CDC at Golden-Gate at VTB. Habr*. Retrieved 27.05.2022 from https://habr.com/ru/company/vtb/blog/479080/
2. (n.d.). *Experience with Oracle GoldenGate Solution. JetInfo.* Retrieved 27.05.2022 from https://www.jetinfo.ru/opyt-ispolzovaniya-resheniya-oracle-goldengate-dlya-business-intelligence/
3. (n.d.). *Data storage theory. CoderLessons*. Retrieved 27.05.2022 from https://coderlessons.com/tutorials/bolshie-dannye-i-analitika/teoriia-khraneniia-dannykh/teoriia-khraneniia-dannykh
4. (n.d.). *Data Warehouse Concepts. Aws. Aamzon*. Retrieved 27.05.2022 from https://aws.amazon.com/ru/data-warehouse/
5. (n.d.). *What is a data warehouse? ORACLE*. Retrieved 27.05.2022 from https://www.oracle.com/ru/database/what-is-a-data-warehouse/
6. (n.d.). *Data Warehouse Architecture: Traditional vs. Cloud. Panoply*. Retrieved 27.05.2022 from https://panoply.io/data-warehouse-guide/data-warehouse-architecture-traditional-vs-cloud/
7. (n.d.). *GoldenGate Most likely errors. Expertise Oracle Learning and experimenting on Oracle technologies.* Retrieved 27.05.2022 from https://oralink.wordpress.com/tag/ora-01031/
8. (n.d.). *ORACLE DATABASE 101*. Retrieved 27.05.2022 from https://oracledb101.wordpress.com
9. (n.d.). *Oracle GoldenGate. ФОРС*. Retrieved 27.05.2022 from https://www.fors.ru/business-solutions/analytical-systems-and-data-warehouse/oracle-goldengate/
10. (n.d.). *Oracle GoldenGate 19.1. Oracle*. Retrieved 27.05.2022 from https://docs.oracle.com/en/middleware/goldengate/core/19.1/index.html