УДК 519.254, 004.048 **DOI: 10.14529/cmse250102**

ИНТЕГРАЦИЯ В РЕЛЯЦИОННУЮ СУБД СРЕДСТВ ВОССТАНОВЛЕНИЯ ПРОПУСКОВ ВРЕМЕННЫХ РЯДОВ В РЕЖИМЕ РЕАЛЬНОГО ВРЕМЕНИ

© 2025 А.А. Юртин

Южно-Уральский государственный университет
(454080 Челябинск, пр. им. В.И. Ленина, д. 76)
E-mail: iurtinaa@susu.ru

Поступила в редакцию: 15.10.2024

В статье рассмотрена проблема интеграции восстановления временных рядов в реляционную СУБД. Предложен метод ImputeDB, обеспечивающий внедрение нейросетевых моделей восстановления пропусков в реальном времени в СУБД PostgreSQL. Восстановление пропусков осуществляется с помощью триггеров (хранимых функций, автоматически выполняемые ядром СУБД при наступлении события вставки новых данных). При активации триггера пропущенные значения заменяются синтетическими, генерируемыми обученной нейросетевой моделью. Используя предложенный метод, прикладной программист базы данных может внедрить процесс восстановления пропущенных значений в стандартный цикл обработки временных рядов, не прибегая к сторонним сервисам. Предложенный метод включает набор следующих программных компонентов, реализованных как пользовательские функции (UDF, user-defined functions) на языках Python и PL/Python: Конструктор триггеров, Менеджер моделей, Хранилище моделей и Восстановитель. Конструктор триггер используется для создания триггеров, которые автоматически выполняют восстановление пропущенных значений в вставляемых данных. Менеджер моделей отвечает за обучение нейросетевых моделей. Хранилище моделей используется для сохранения моделей в файловом хранилище. Восстановитель, в свою очередь, синтезирует пропущенные значения с помощью обученных моделей. В исследовании были проведены эксперименты для оценки производительности метода ImputeDB. В ходе экспериментов измерялось время обработки вставки данных с автоматическим восстановлением пропусков в зависимости от размерности временного ряда. Эксперименты проводились в двух сценариях (одиночная и множественная вставка). В качестве моделей восстановления использовались нейросетевые методы с различными архитектурами, включая рекуррентные нейросети, автоэнкодеры и трансформеры. Результаты экспериментов продемонстрировали, что в условиях увеличения размерности временного ряда, роста накладных расходов на сетевые запросы и передачу данных, ImputeDB показывает наилучшую производительность. В частности, система обеспечила прирост эффективности на 22.5% по сравнению с аналогом, при этом сохраняя точность восстановления используемых методов.

Ключевые слова: временной ряд, СУБД, PostgreSQL, восстановление пропущенных значений, нейронные сети.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Юртин А.А. Интеграция в реляционную СУБД средств восстановления пропусков временных рядов в режиме реального времени // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2025. Т. 14, № 1. С. 30–46. DOI: 10.14529/cmse250102.

Введение

Анализ временных рядов играет ключевую роль в различных областях, включая финансы [1], мониторинг сложных технических систем [2] и анализ потребительского спроса [3]. Рост объемов данных временных рядов и сложность их анализа делают базы данных неотъемлемым инструментом для хранения и обработки таких данных. Современные базы данных позволяют эффективно управлять временными рядами, предоставляя возможности для их масштабирования, организации и анализа. В последние годы активно исследуется интеграция методов интеллектуального анализа временных рядов непосредственно в систе-

мы управления базами данных (СУБД) [4, 5]. Различные исследования предлагают способы решения задач анализа временных рядов: поиск аномалий [6], вычисление матричного профиля [7], прогноз и восстановление значений, и др. Например, базы данных ориентированные на хранение временных рядов, такие как TimeTravel [8], F2DB [9], tspDB [10], RecovDB [11] и др., используют различные аналитические методы для прогноза новых и восстановления пропущенных данных временных рядов.

Однако нейросетевые модели демонстрируют лучшую точность в задачах восстановления временных рядов по сравнению с традиционными методами. Одной из СУБД, внедряющей в работу баз данных глубокое обучение, является Lindorm TSDB [6]. СУБД позволяет производить обнаружение аномалий и прогнозирование временных рядов. Однако набор поддерживаемых моделей ограничен базовыми алгоритмами (ARIMA [12], DeepAR [13] и TFT [14]), которые уступают более передовым аналогам по точности.

Используя различные архитектуры (рекуррентные [15, 16], автоэнкодеры [17], трансформеры [18, 19]и др. [20, 21]), модели способны эффективно обрабатывать сложные зависимости в данных. Применение нейронных сетей позволяет учитывать корреляции между различными временными рядами, повышая точность восстановления. В отличие от простых статистических методов (например, интерполяции или использования среднего значения), нейросетевые решения адаптируются к уникальным особенностям временного ряда. Сложные модели глубокого обучения в основном не интегрируются в системы баз данных. Восстановление данных выполняется вне базы данных с использованием сторонних сервисов до их вставки в таблицу. Процесс может включать извлечение данных, передачу на обработку, восстановление пропущенных значений и возврат в базу данных. Одна из проблем такого подхода — дополнительные накладные расходы связанные с пересылкой данных. Данные проходят через несколько систем, что увеличивает общее время обработки и усложняет управление инфраструктурой. Постоянный обмен данными между серверами замедляет процесс обработки и создает дополнительные требования к пропускной способности сети, что может стать узким местом в высоконагруженных системах. Задержки, возникающие при передаче больших объемов информации, особенно критичны для задач, требующих оперативного восстановления временных рядов. Необходимость интеграции нескольких сервисов усложняет инфраструктуру, повышает риски сбоев и затрудняет контроль качества восстановления. Когда скорость и точность обработки данных играют ключевую роль, такие ограничения становятся значимым фактором, влияющим на эффективность.

В данном исследовании предложен метод ImputeDB, позволяющий внедрять нейросетевые модели восстановления временных рядов в режиме реального времени в качестве составных частей СУБД PostgreSQL. Для внедрения используются встроенные триггеры, которые активируются при вставке новых данных. Метод реализуется как набор пользовательских функций (UDF) на языках Python и PL/Python. Для реализации предложенного метода предполагается наличие следующих компонентов. Конструктор, который создает триггеры для таблиц, используя параметры модели, таблицы и размер окна. Менеджер, отвечающий за подготовку данных, обучение моделей, их сохранение и загрузку из хранилища. Хранилище, которое реализуется как файловое хранилище, осуществляет извлечение моделей. Восстановитель, использующий модели для генерации синтетических значений, заменяющих пропуски. Для подтверждения эффективности предложенного метода, были проведены вычислительные эксперименты. В экспериментах оценивалась производительность вставки данных при восстановлении пропусков с использованием разных нейросете-

вых методов. В ходе экспериментов предложенный метод сравнивался с внешним сервисом, использующим базу данных Redis. Изучалось влияние числа координат, объема вставляемых данных и числа пользователей на производительность.

Статья организована следующим образом. В разделе 1 представлен новый метод внедрения моделей машинного обучения в СУБД PostreSQL для восстановления временных рядов. Результаты вычислительных экспериментов, исследующих эффективность предложенного метода, приведены в разделе 2. Заключение содержит сводку полученных результатов и направления будущих исследований.

1. Внедрение моделей машинного обучения в СУБД PostreSQL для восстановления временных рядов

1.1. Формальные определения и обозначения

Одномерный временной ряд представляет собой хронологически упорядоченную последовательность вещественных значений:

$$T = \{t_i\}_{i=1}^{n}, \quad t_i \in \mathbb{R}. \tag{1}$$

Длина временного ряда, n, обозначается как $|T|$.

Подпоследовательность $T_{i,m}$ одномерного временного ряда T — это непрерывный промежуток из m элементов ряда, начиная с i-го элемента:

$$T_{i,m} = \{t_k\}_{k=i}^{i+m-1}, \quad 1 \leqslant i \leqslant n-m+1, \quad 3 \leqslant m \ll n. \tag{2}$$

Многомерный временной ряд — это набор семантически связанных одномерных временных рядов одинаковой длины, которые синхронизированы во времени. Обозначим размерность многомерного ряда (количество измерений в нем) как d ($d > 1$). Подобно одномерному случаю, многомерный временной ряд, его подпоследовательность и отдельные точки обозначим как $\boldsymbol{T}$, $\boldsymbol{T}_{i,m}$ и $\boldsymbol{t}_i$ соответственно, и определим их следующим образом:

$$\boldsymbol{T} = [\{T^{(k)}\}_{k=1}^{d}]^{\intercal}, \tag{3}$$

$$\boldsymbol{T}_{i,m} = [\{T_{i,m}^{(k)}\}_{k=1}^{d}]^{\intercal}, \tag{4}$$

$$\boldsymbol{t}_i = [\{t_i^{(k)}\}_{k=1}^{d}]^{\intercal}. \tag{5}$$

1.2. Архитектура предлагаемого решения.

Предполагается, что хранение временного ряда $\boldsymbol{T}$ в СУБД организовано следующим образом. Каждая i-я многомерные точки $\boldsymbol{t}_i$ представлена в виде отдельной записи в реляционной таблице, содержащей следующие столбцы: порядковый номер точки, отметка времени и d вещественных атрибутов. Структура таблицы обладает следующими свойствами. Каждый из d атрибутов соответствует одному из измерений многомерной точки. Все строки таблицы отсортированы по времени и не содержат пропусков. Вставка новых значений в таблицу осуществляется последовательно с равными временными интервалами. Каждая операция вставки добавляет в таблицу одну многомерную точку. Предполагается, что в СУБД обладает следующими свойствами. В СУБД реализован механизм триггеров. Данный механизм представляет собой встроенный инструмент, обеспечивающий автоматическое выполнение заданных процедур при возникновении событий, связанных с изме-

нением данных в таблице [22]. В СУБД поддерживается вызов PL/Python, позволяющий использовать язык Python для создания хранимых процедур и функций.

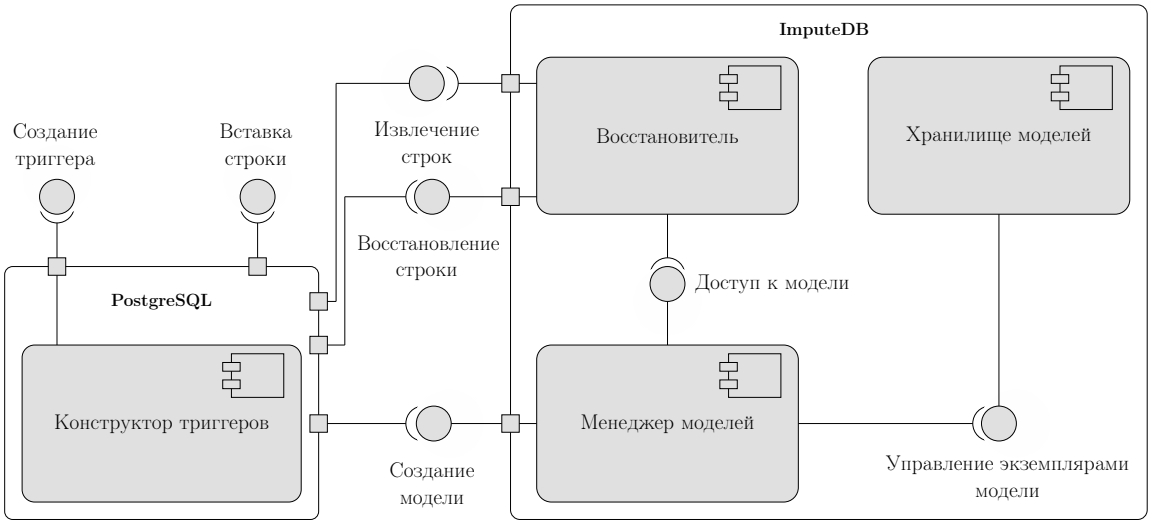


Рис. 1. Архитектура ImputeDB

Для внедрения нейросетевых моделей восстановления временных рядов в реляционные СУБД предлагается расширение ImputeDB, архитектура которого представлена на рис. 1. Процесс восстановления пропущенных значений реализуется с использованием механизма триггеров. При добавлении новой многомерной точки активируется триггер, в результате работы которого отсутствующие значения заменяются на синтетические данные, генерируемые обученной моделью восстановления. Предлагаемое расширение ImputeDB включает несколько ключевых компонентов, обеспечивающих управление моделями, их хранение и процесс восстановления. *Менеджер моделей* отвечает за полный жизненный цикл нейросетевых моделей, решая следующие задачи: создание, обучение и обновление нейросетевых моделей временного ряда. Данный компонент предоставляет интерфейс управления экземплярами, позволяя остальным компонентам системы запрашивать уже обученные модели для дальнейшего их применения. Обученные модели передаются в *Хранилище моделей*, которое отвечает за их долговременное хранение. Хранение реализуется путем записи данных в дисковое пространство, обеспечивая быстрый доступ к ним при необходимости. Восстановление пропущенных значений выполняется компонентом *Восстановитель*. Данный компонент системы принимает запросы на восстановление строк. После получения запроса на восстановление многомерной точки ряда $\boldsymbol{T}$ компонент *Восстановитель* выполняет следующие действия. Компонент запрашивает у *Менеджера моделей* обученную для ряда $\boldsymbol{T}$ модель. Далее *Восстановитель* выгружает из базы данных подпоследовательность $\boldsymbol{T}_{n-m+1,\ n}$, представляющую собой $m-1$ предыдущих строк восстанавливаемого ряда. Используя полученную модель, *Восстановитель* генерирует синтетические данные.

Для работы ImputeDB в исходной СУБД создается дополнительный компонент *Конструктор триггеров*. Данный компонент представляет собой хранимую UDF-процедуру, которая при вызове формирует и регистрирует триггеры, необходимые для реализации предлагаемого метода в СУБД. Сформированные триггеры активируются при вставке новых значений в таблицу. В момент их срабатывания выполняется обращение к интерфейсу *Восстановителя*, который отвечает за восстановление пропущенных данных в добавленной строке.

1.3. Реализация компонентов

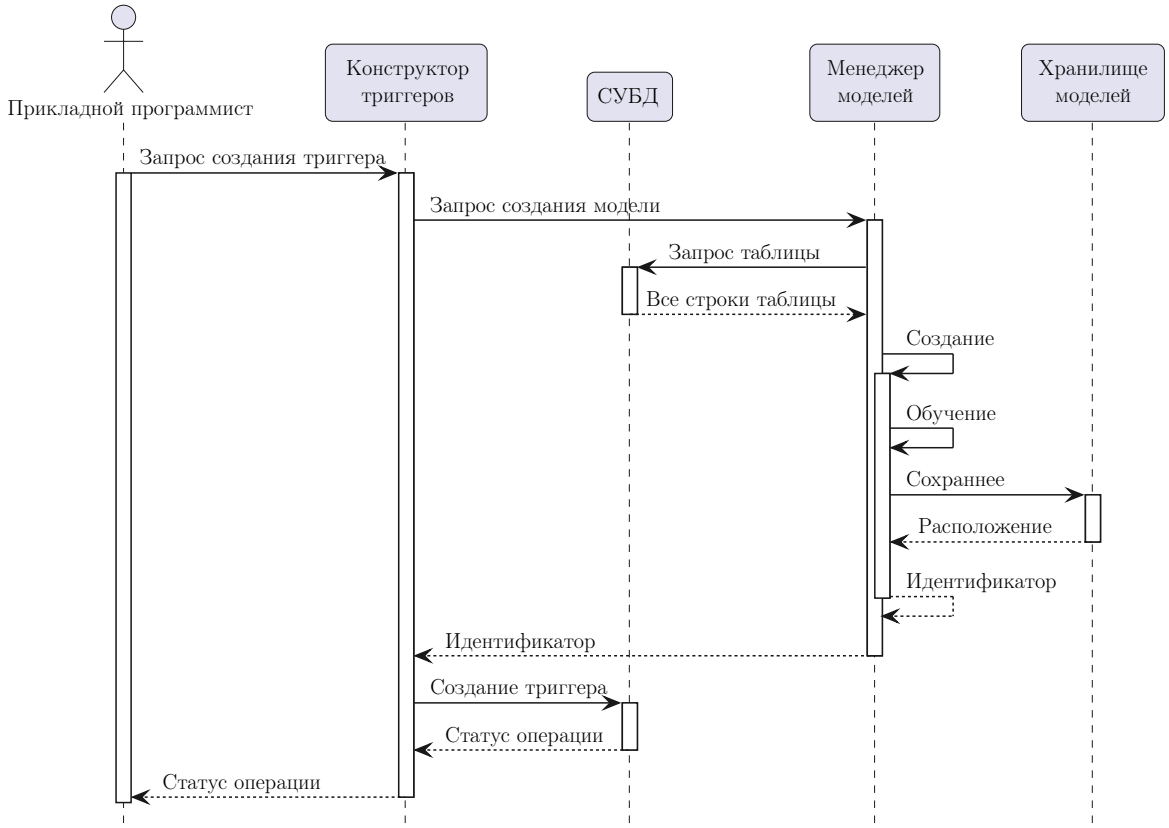


Рис. 2. Диаграмма последовательности создание триггера

На рис. 2 представлен процесс создания триггера. Для интеграции процесса восстановления данных прикладной программист отправляет запрос *Конструктору триггеров*, указывая необходимые параметры: название модели восстановления, целевую таблицу, содержащую временной ряд $\boldsymbol{T}$ и длину подпоследовательности m. *Конструктору триггеров* реализуется как набор пользовательских функций (UDF, user-defined functions), которые создают триггер для обработки событий вставки новой точки в таблицу. После получения запроса *Конструктор триггеров* формирует запрос для *Менеджера моделей*. Получив запрос, *Менеджера моделей* инициирует подготовку модели. *Менеджер моделей* выгружает все n строк временного ряда $\boldsymbol{T}$ из таблицы СУБД, указанной в запросе. Извлеченный временной ряд подвергается предварительной обработке и в дальнейшем используется для формирования обучающей выборки. Во время предварительной обработки ряда последовательно выполняются следующие операции: нормализация, разделение ряда на подпоследовательности фиксированной длинны m и разделение подпоследовательностей на обучающую и тестовую выборки. В качестве нормализации используется минимаксная нормализация, которая приводит данные в каждом измерении к диапазону $[0, 1]$ и определяется следующим образом:

$$\hat{t}_i^{(k)} = \frac{t_i^{(k)} - \min\limits_{1 \leqslant j \leqslant n} t_j^{(k)}}{\max\limits_{1 \leqslant j \leqslant n} t_j^{(k)} - \min\limits_{1 \leqslant j \leqslant n} t_j^{(k)}}, \quad 1 \leqslant k \leqslant d. \tag{6}$$

В зависимости от выбранной модели в процесс подготовки данных могут включаться различные дополнительные этапы предварительной обработки. Например, для моделей

SANNI [23] и SAETI [23] предварительная обработка должна включать выполнение алгоритма PSF [24], который реализует поиск типичных подпоследовательностей ряд (снипетов) [25].

Обучающая выборка формируется из нормализованной версии исходного временного ряда $\boldsymbol{T}$. Обозначим обучающую выборку как $D = \langle \boldsymbol{X}, \boldsymbol{Y} \rangle$, где $\boldsymbol{X}$ и $\boldsymbol{Y}$ представляют собой входные и выходные данные нейросетевых моделей соответственно. Входные данные $\boldsymbol{X}$ представляют собой многомерные подпоследовательности длинной m временного ряда $\boldsymbol{T}$, у которых последнее значение в каждом измерении было заменено на $\texttt{NIL} = -1$. Выходными данными полагаются последние многомерные точки входных подпоследовательностей до замены на NIL:

$$D_{\text{Reconstructor}} = \{\langle \boldsymbol{X}, \boldsymbol{Y} \rangle \mid X^{(k)} = T^{(k)}_{i,\,m-1} \bullet \texttt{NIL},\ \texttt{NIL} = -1,\ Y^{(k)} = t^{(k)}_{i+m},\ 1 \leqslant i \leqslant n - m + 1,\ 1 \leqslant k \leqslant d\}. \tag{7}$$

Подпоследовательности временного ряда $\boldsymbol{T}$ разделяются на обучающую и тестовую выборки в соотношении 75% и 25% соответственно. После формирования обучающих выборок *Менеджер модели* создает экземпляр модели и инициирует процесс ее обучения. По завершении обучения модель сохраняется в *Хранилище моделей* — централизованном файловом хранилище, предназначенном для хранения всех обученных моделей метода. *Хранилище моделей* генерирует уникальный идентификатор для каждой модели, который представляет собой строку, состоящую из названия таблицы, названия модели и длины подпоследовательности. После сохранения модели ее идентификатор передается обратно в *Конструктор триггеров*. Получив идентификатор, *Конструктор триггеров* формирует SQL-скрипт для сохранения триггера в СУБД, который реагирует на события вставки новых строк в таблицу временного ряда.

На рис. 3 представлена диаграмма последовательности, иллюстрирующая процесс восстановления данных с пропусками. Процесс восстановления пропущенных значений начинается с того, что *Источник данных* инициирует операцию вставки новой строки в таблицу временного ряда $\boldsymbol{T}$ в СУБД. Вставляемая строка может быть представлена как вектор длины m. В случае если хотя бы одно значение отсутствует, СУБД направляет запрос в компонент *Восстановитель* для генерации синтетических значений вместо пропусков. Получив запрос, *Восстановитель* извлекает из СУБД последние $m-1$ строк таблицы временного ряда. Извлеченные строки формируют подпоследовательность, которая используется как вход модели для восстановления пропущенных значений. Входная строка с пропуском становится m-м элементом этой подпоследовательности.

Сформировав подпоследовательность, *Восстановитель* запрашивает у *Менеджера моделей* модель, которая была обучена на данных временного ряда $\boldsymbol{T}$. Если запрашиваемая модель не была ранее загружена в оперативную память *Менеджера моделей*, направляется запрос в *Хранилище моделей* для извлечения модели. *Хранилище моделей* извлекает модель из файлового хранилища и передает ее экземпляр обратно *Менеджеру моделей*. После получения модели *Менеджер моделей* передает ее экземпляр в *Восстановитель*. *Восстановитель* применяет модель для восстановления пропущенных значений в строке. Затем восстановленные данные передаются в СУБД, которая выполняет вставку новой строки с восстановленными значениями в таблицу. В завершение СУБД возвращает статус операции, завершая процесс вставки данных.

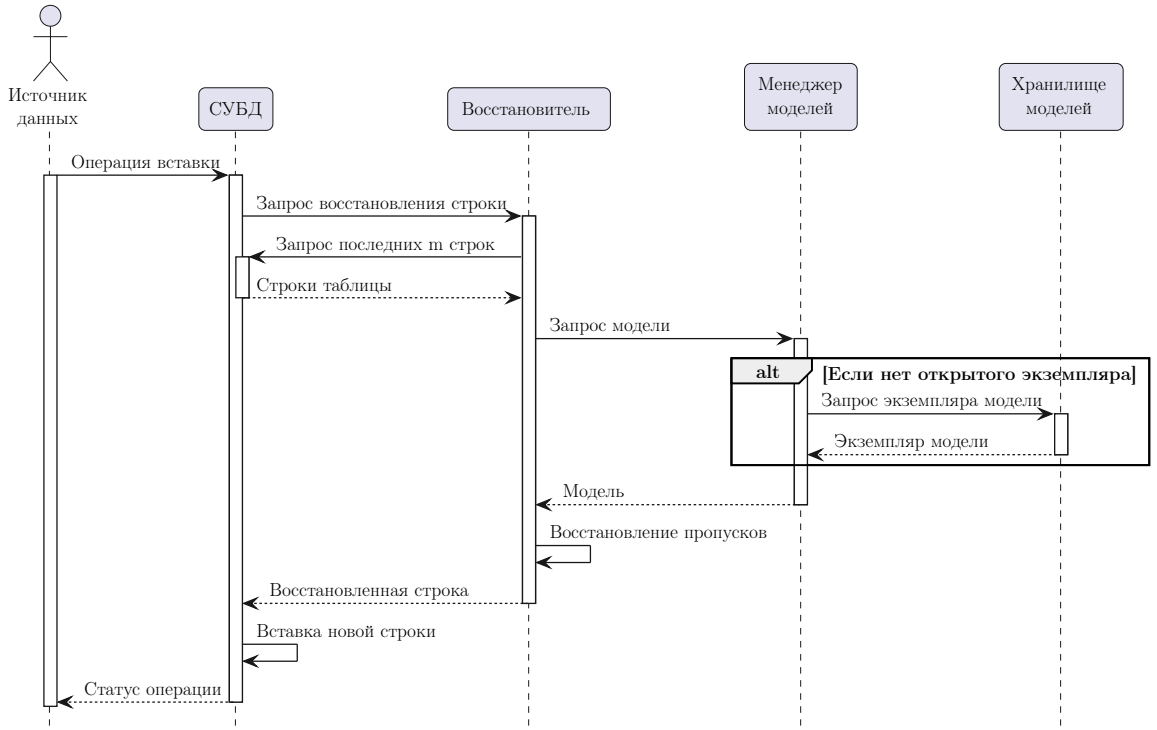


Рис. 3. Диаграмма последовательности востановления данных с пропусками

```
1  -- Запрос на создание триггера для таблицы TestTable
2  -- на основе среднего за последние 100 точек
3
4  SELECT create_impute_trigger(TestTable, mean, 100);
5
6  -- Последовательность запросов, выполняемых функцией create_impute_trigger
7  -- с заданными параметрами
8
9  SELECT create_model(TestTable, mean, 100);
10
11 CREATE OR REPLACE FUNCTION TestTable_mean_100_impute_func()
12   RETURNS TRIGGER AS $func_impute$
13     from plpy import execute
14     from sys import path
15     path.append('/opt/scripts/app')
16     from triggers.trigger_helpers import impute_point
17     TD['new'] = impute_point(TD['new'], mean, TestTable, 100)
18     return 'MODIFY'
19   $func_impute$ LANGUAGE plpython3u;
20
21 CREATE TRIGGER TestTable_mean_100_impute_trigger
22   BEFORE INSERT ON TestTable
23   FOR EACH ROW
24   EXECUTE FUNCTION TestTable_mean_100_impute_func();
25
```

Рис. 4. Пример выполнения запроса на создание триггера

На рис. 4 приведен пример выполнения запроса на создание триггера, который будет инициировать восстановление пропущенных значений в таблице *TestTable* с использова-

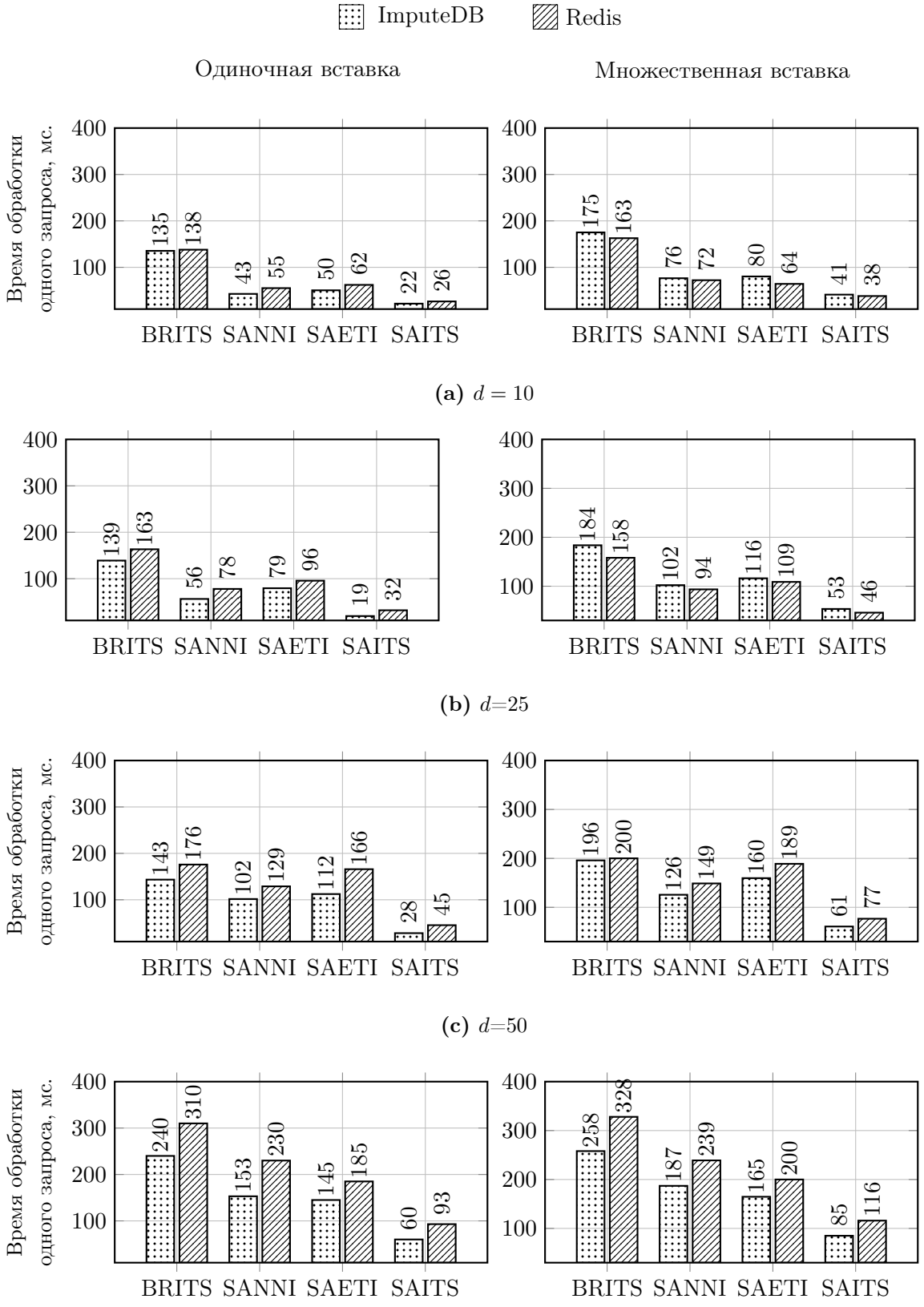
нием среднего значения последних 100 точек. Прикладной программист инициирует вызов функции `create_impute_trigger`, которая принимает в качестве аргументов следующие параметры: имя таблицы, метод восстановления (в данном случае среднее значение) и размер подпоследовательности $m = 100$. Во время выполнения функции вызывается вспомогательная функция `creat_model(TestTable, mean, 100)`, предназначенная для создания и обучения модель восстановления, которая будет использовать для восстановления переданный в качестве аргумента метод. После завершения обучения модель сохраняется в хранилище. Для реализации логики восстановления используется расширение PostgreSQL `plpython3u`, которое обеспечивает возможность выполнения кода на языке Python непосредственно внутри базы данных. Затем создается пользовательская функция восстановления, которая будет вызываться триггером. В процессе работы созданной функции происходит вызов процедуры восстановления значений, основанной на использовании обученной модели. Завершающим этапом является создание триггера `TestTable_mean_100_impute_trigger`, который будет активироваться при вставке новых строк в таблицу *TestTable*.

2. Вычислительные эксперименты

Разработанный подход был реализован для свободной СУБД PostgreSQL 13.2. Для исследования эффективности предложенного метода были проведены вычислительные эксперименты на оборудовании Лаборатории суперкомпьютерного моделирования ЮУрГУ [26]. Сбор результатов осуществлялся на платформе рабочей станции со следующими характеристиками: процессор Intel Xeon Gold 6254 4 ГГц, оперативная память 64 Гб, дисковая память 1 Тб, одно графическое ядро A30.

2.1. Исходные данные

В рамках вычислительных экспериментов была проведена оценка производительности операций вставки данных в таблицы с применением различных нейросетевых методов восстановления пропущенных значений. В ходе эксперимента измерялось время выполнения операции вставки (в секундах) в зависимости от количества измерений временного ряда d. В качестве d рассматривались следующие значение: 10, 25, 50 и 100. Эксперимент проводился в двух различных сценариях. В первом сценарии (одиночная вставка) новые точки данных добавлялись в таблицу из единственного источника. Во втором сценарии (множественная вставка) данные поступали одновременно от 1000 независимых источников в различные таблицы. В качестве моделей восстановления рассматривались нейросетевые методы с различными архитектурами, включая рекуррентные нейросети (BRITS [15] и SANNI [27]), автоэнкодеры (SAETI [23]) и трансформеры (SAITS [18]). Производительность предложенного метода сравнивалась с альтернативным подходом, основанным на использовании внешнего сервиса, предоставляющего API для восстановления пропусков в данных. Внешний сервис использовал идентичные модели восстановления. Для хранения обученных моделей внешний сервис использовал базу данных Redis. В сценарии, предполагающем использование внешнего сервиса, восстановление пропущенных значений выполнялось посредством HTTP-запросов, выполняемых непосредственно перед вставкой данных в основную базу. С целью минимизации влияния сетевых задержек на результаты эксперимента, внешняя сервисная инфраструктура и база данных размещались на одном сервере.



(a) $d = 10$

(b) d=25

(c) d=50

(d) d=100

Рис. 5. Сравнение производительность различных способов организации восстановления

2.2. Анализ производительности

На рис. 5 представлено сравнительное исследование производительности различных методов восстановления пропущенных данных в базе данных. Рассмотрим более подробно результаты для одиночной вставки. Для временных рядов с небольшим числом измерений, например при $d = 10$, разница во времени обработки между ImputeDB и Redis минимальна. В данном случае объем передаваемых данных через сеть незначителен, что означает, что накладные расходы на сетевое взаимодействие имеют минимальное влияние. Вследствие этого преимущество, которое демонстрирует предлагаемый метод, незначительное и составляет в среднем 15%. С увеличением размерности временного ряда, например при $d = 25$, разница в производительности становится более выраженной. Для Redis накладные расходы на отправку HTTP-запросов начинают существенно расти, что приводит к увеличению времени обработки. В результате ImputeDB демонстрирует прирост производительности на уровне 19%. При дальнейшем увеличении размерности временного ряда разница в производительности становится еще более заметной: при $d = 50$ достигается прирост на 27%, а при $d = 100$ на 28%. С ростом размерности временного ряда накладные расходы, связанные с сетевыми запросами и процессами сериализации и десериализации данных, оказывают все более значительное влияние на работу сервиса на, использующего Redis. В то время как ImputeDB сохраняет стабильность работы и значительно выигрывает по времени обработки.

Для сценария множественной вставки результаты демонстрируют, что влияние размерности временного ряда на производительность ImputeDB и внешнего сервиса, использующего Redis, различается. При малых значениях d, таких как $d = 10$ и $d = 25$, внешний сервис показывает более высокую эффективность, обеспечивая ускорение обработки на 11% и 12% соответственно. В случае использования Redis, модели загружаются непосредственно из оперативной памяти, что обеспечивает минимальное время доступа. В отличие от этого, в ImputeDB модели хранятся в файловой системе, что приводит к дополнительным задержкам при их извлечении. Однако с увеличением размерности временного ряда эффективность ImputeDB возрастает. Для $d = 50$ производительность ImputeDB оказывается выше на 13.5%, для $d = 100$ преимущество увеличивается до 21.48%. Такой рост обусловлен тем, что при больших d накладные расходы на загрузку моделей из файлового хранилища становятся несущественными в сравнении с задержками, возникающими при передаче данных через сеть во внешнем сервисе.

Таким образом, полученные в ходе экспериментов результаты, демонстрируют зависимость производительности ImputeDB и внешнего сервиса, использующего Redis, от размерности временного ряда и характеристик операции вставки данных. Для малых значений d (например, $d = 10$ и $d = 25$) внешний сервис показывает более высокую эффективность за счет использования оперативной памяти для хранения моделей, что ускоряет их доступ и обработку. Однако при увеличении размерности временного ряда ImputeDB демонстрирует значительное преимущество. Рост значения d приводит к увеличению накладных расходов на обработку HTTP-запросов и передачу данных во внешнем сервисе, тогда как ImputeDB, работая локально, минимизирует сетевые задержки. Средний прирост производительности ImputeDB по сравнению с Redis составляет 22.25%. Возможным направлением дальнейшей оптимизации ImputeDB является интеграция Redis в качестве механизма кэширования моделей. Внедрение Redis позволит снизить накладные расходы на их извлечение и повысить эффективность системы при обработке временных рядов малой размерности, объединяя преимущества локального хранения и быстрого доступа к данным.

Заключение

В данном исследовании затронута проблема интеграции методов восстановления временных рядов в СУБД, которая является актуальной в широком спектре предметных областей. Предложен метод ImputeDB, обеспечивающий интеграцию нейросетевых моделей восстановления пропущенных значений в реляционную СУБД PostgreSQL. Процесс восстановления пропущенных значений в ImputeDB реализуется с использованием триггеров, которые активируются при добавлении новых многомерных точек данных в таблицу. Предложенный метод включает следующие ключевые компоненты: Менеджер моделей, Хранилище моделей и Восстановитель. Менеджер моделей управляет жизненным циклом нейросетевых моделей, включая их создание, обучение и обновление. Обученные модели сохраняются в Хранилище моделей для долговременного хранения. Восстановитель отвечает за обработку запросов на восстановление пропущенных значений, применяя обученные модели для генерации синтетических данных. Процесс создания триггера начинается с запроса прикладного программиста баз данных в Конструктор триггеров, который генерирует соответствующий SQL-скрипт для создания триггера в СУБД. При вставке новых строк в таблицу, если в данных обнаружены пропущенные значения, СУБД инициирует запрос в компонент Восстановитель. Данный компонент извлекает последние строки временного ряда и использует заранее обученную модель для восстановления пропущенных значений. В ходе экспериментов была проведена оценка производительности операций вставки данных при использовании различных нейросетевых методов восстановления пропущенных значений, таких как рекуррентные сети, автоэнкодеры и трансформеры. Проведен сравнительный анализ с внешним сервисом, использующим Redis и те же модели восстановления через API. Результаты показали, что при малых значениях размерности временного ряда внешний сервис демонстрирует меньшее время отклика за счет быстрого извлечения моделей из оперативной памяти. Однако с увеличением размерности временного ряда, ImputeDB показал значительный прирост производительности, так как минимизирует сетевые задержки, работая локально. Средний прирост производительности ImputeDB составляет 22.25%. Будущие направления исследований включают адаптацию метода для работы с другими задачи интеллектуального анализа данных, расширение набора поддерживаемых моделей и улучшения производительности на малых значениях размерности временного ряда за счет использование Redis в качестве кэширования моделей.

Работа выполнена при финансовой поддержке Российского научного фонда (грант № 23-21-00465).

Литература

1. Majumdar S., Laha A.K. Clustering and classification of time series using topological data analysis with applications to finance // Expert Syst. Appl. 2020. Vol. 162. P. 113868. DOI: 10.1016/j.eswa.2020.113868.
2. Kumar S., Tiwari P., Zymbler M.L. Internet of Things is a revolutionary approach for future technology enhancement: a review // J. Big Data. 2019. Vol. 6. P. 111. DOI: 10.1186/S40537-019-0268-2.
3. Seyedan M., Mafakheri F. Predictive big data analytics for supply chain demand forecasting: methods, applications, and research opportunities // J. Big Data. 2020. Vol. 7, no. 1. P. 53. DOI: 10.1186/S40537-020-00329-2.

4. Jensen S.K., Pedersen T.B., Thomsen C. Time Series Management Systems: A Survey // IEEE Trans. Knowl. Data Eng. 2017. Vol. 29, no. 11. P. 2581–2600. DOI: 10.1109/TKDE.2017.2740932.

5. Иванова Е.В., Цымблер М.Л. Обзор современных систем обработки временных рядов // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2020. Т. 9, № 4. С. 79–97. DOI: 10.14529/cmse200406.

6. Shen C., Ouyang Q., Li F., *et al.* Lindorm TSDB: A Cloud-native Time-series Database for Large-scale Monitoring Systems // Proc. VLDB Endow. 2023. Vol. 16, no. 12. P. 3715–3727. DOI: 10.14778/3611540.3611559.

7. Иванова Е.В., Цымблер М.Л. Внедрение концепции матричного профиля в реляционную СУБД для интеллектуального анализа временных рядов // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2021. Т. 10, № 3. С. 72–87. DOI: 10.14529/cmse210305.

8. Khalefa M.E., Fischer U., Pedersen T.B., Lehner W. Model-based Integration of Past & Future in TimeTravel // Proc. VLDB Endow. 2012. Vol. 5, no. 12. P. 1974–1977. DOI: 10.14778/2367502.2367551.

9. Fischer U., Rosenthal F., Lehner W. F2DB: The Flash-Forward Database System // IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012 / ed. by A. Kementsietsidis, M.A.V. Salles. IEEE Computer Society, 2012. P. 1245–1248. DOI: 10.1109/ICDE.2012.117.

10. Agarwal A., Alomar A., Shah D. tspDB: Time Series Predict DB // NeurIPS 2020 Competition and Demonstration Track, 6-12 December 2020, Virtual Event, Vancouver, BC, Canada. Vol. 133 / ed. by H.J. Escalante, K. Hofmann. PMLR, 2020. P. 27–56. Proceedings of Machine Learning Research. URL: http://proceedings.mlr.press/v133/agarwal21a.html.

11. Arous I., Khayati M., Cudré-Mauroux P., *et al.* RecovDB: Accurate and Efficient Missing Blocks Recovery for Large Time Series // 35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019. IEEE, 2019. P. 1976–1979. DOI: 10.1109/ICDE.2019.00218.

12. Ariyo A.A., Adewumi A.O., Ayo C.K. Stock Price Prediction Using the ARIMA Model // UKSim-AMSS 16th International Conference on Computer Modelling and Simulation, UKSim 2014, Cambridge, United Kingdom, March 26-28, 2014 / ed. by D. Al-Dabass, A. Orsoni, R.J. Cant, *et al.* IEEE, 2014. P. 106–112. DOI: 10.1109/UKSIM.2014.67.

13. Salinas D., Flunkert V., Gasthaus J., Januschowski T. DeepAR: Probabilistic forecasting with autoregressive recurrent networks // International Journal of Forecasting. 2020. Vol. 36, no. 3. P. 1181–1191. DOI: 10.1016/j.ijforecast.2019.07.001.

14. Lim B., Arik S.Ö., Loeff N., Pfister T. Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting // CoRR. 2019. Vol. abs/1912.09363. arXiv: 1912.09363. URL: http://arxiv.org/abs/1912.09363.

15. Cao W., Wang D., Li J., *et al.* BRITS: Bidirectional Recurrent Imputation for Time Series // Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada / ed. by S. Bengio, H.M. Wallach, H. Larochelle, *et al.* 2018. P. 6776–6786. URL: https://proceedings.neurips.cc/paper/2018/hash/734e6bfcd358e25ac1db0a4241b95651-Abstract.html.

16. Yoon J., Zame W.R., Schaar M. van der Estimating Missing Data in Temporal Data Streams Using Multi-Directional Recurrent Neural Networks // IEEE Trans. Biomed. Eng. 2019. Vol. 66, no. 5. P. 1477–1490. DOI: 10.1109/TBME.2018.2874712.

17. Fortuin V., Baranchuk D., Rätsch G., Mandt S. GP-VAE: Deep Probabilistic Time Series Imputation // The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]. Vol. 108 / ed. by S. Chiappa, R. Calandra. PMLR, 2020. P. 1651–1661. Proceedings of Machine Learning Research. URL: http://proceedings.mlr.press/v108/fortuin20a.html.

18. Du W., Côté D., Liu Y. SAITS: Self-attention-based imputation for time series // Expert Syst. Appl. 2023. Vol. 219. P. 119619. DOI: 10.1016/J.ESWA.2023.119619.

19. Oh E., Kim T., Ji Y., Khyalia S. STING: Self-attention based Time-series Imputation Networks using GAN // CoRR. 2022. Vol. abs/2209.10801. DOI: 10.48550/ARXIV.2209.10801. arXiv: 2209.10801.

20. Fang C., Wang C. Time Series Data Imputation: A Survey on Deep Learning Approaches // CoRR. 2020. Vol. abs/2011.11347. arXiv: 2011.11347. URL: https://arxiv.org/abs/2011.11347.

21. Wang J., Du W., Cao W., *et al.* Deep Learning for Multivariate Time Series Imputation: A Survey // CoRR. 2024. Vol. abs/2402.04059. DOI: 10.48550/ARXIV.2402.04059. arXiv: 2402.04059.

22. Silberschatz A., Korth H.F., Sudarshan S. Database System Concepts, Seventh Edition. McGraw-Hill Book Company, 2020. URL: https://www.db-book.com/.

23. Юртин А.А. Восстановление многомерных временных рядов на основе выявления поведенческих шаблонов и применения автоэнкодеров // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2024. Т. 13, № 2. С. 39–55. DOI: 10.14529/cmse240203.

24. Zymbler M., Goglachev A. Fast summarization of long time series with graphics processor // Mathematics. 2022. Vol. 10, no. 10. P. 1781. DOI: 10.3390/math10101781.

25. Imani S., Madrid F., Ding W., *et al.* Introducing time series snippets: A new primitive for summarizing long time series // Data Min. Knowl. Discov. 2020. Vol. 34, no. 6. P. 1713–1743. DOI: 10.1007/s10618-020-00702-y.

26. Биленко Р.В., Долганина Н.Ю., Иванова Е.В., Рекачинский А.И. Высокопроизводительные вычислительные ресурсы Южно-Уральского государственного университет // Вычислительные методы и программирование. 2022. Т. 11, № 1. С. 15–30. DOI: 10.14529/cmse220102.

27. Цымблер М.Л., Полонский В.А., Юртин А.А. Об одном методе восстановления пропущенных значений потокового временного ряда в режиме реального времени // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2021. Т. 10, № 4. С. 5–25. DOI: 10.14529/cmse210401.

Юртин Алексей Артемьевич, программист, Лаборатория больших данных и машинного обучения, аспирант кафедры системного программирования, Южно-Уральский государственный университет (национальный исследовательский университет) (Челябинск, Российская Федерация)

DOI: 10.14529/cmse250102

INTEGRATION OF MISSING DATA IMPUTATION TOOLS FOR TIME SERIES IN REAL-TIME MODE INTO A RELATIONAL DBMS

© 2025 A.A. Yurtin

South Ural State University (pr. Lenina 76, Chelyabinsk, 454080 Russia)

E-mail: iurtinaa@susu.ru

Received: 15.10.2024

The article addresses the problem of integrating time series imputation into relational database management systems (RDBMS). A method called ImputeDB is proposed, which enables the real-time integration of neural network-based imputation models into the PostgreSQL RDBMS. The imputation of missing values is carried out through triggers (stored functions automatically executed by the RDBMS kernel when new data is inserted). When a trigger is activated, missing values are replaced by synthetic ones generated by a neural network model. Using the proposed method, a database application programmer can integrate the process of imputing missing values into the standard time series processing pipeline within the PostgreSQL RDBMS, without relying on external services. The proposed approach includes a set of components implemented as user-defined functions (UDFs) in Python and PL/Python: Trigger Constructor, Model Manager, Model Storage, and Imputer. The Trigger Constructor is used to create triggers that automatically perform imputation of missing values in inserted data. The Model Manager is responsible for training neural network models, while the Model Storage is used to save these models in a file-based repository. The Imputer, in turn, synthesizes missing values using the trained models. Experiments were conducted to evaluate the performance of the ImputeDB method. The experiments measured the processing time of data insertion with automatic gap imputation as a function of the time series dimensionality. Experiments were performed under two scenarios: single and multiple insertions. Neural network-based imputation models with various architectures, including recurrent neural networks, autoencoders, and transformers, were employed. The experimental results demonstrated that under conditions of increasing time series dimensionality and rising overhead from network requests and data transfer, ImputeDB exhibits superior performance. Specifically, the system achieved an efficiency gain of 22.5% compared to another approach, while maintaining the accuracy of the employed imputation methods.

Keywords: time series, DBMS, PostgreSQL, missing value imputation, neural networks.

FOR CITATION

Yurtin A.A. Integration of Missing Data Imputation Tools for Time Series in Real-Time Mode into a Relational DBMS. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2025. Vol. 14, no. 1. P. 30–46. (in Russian) DOI: 10.14529/cmse250102.

References

1. Majumdar S., Laha A.K. Clustering and classification of time series using topological data analysis with applications to finance. Expert Syst. Appl. 2020. Vol. 162. P. 113868. DOI: 10.1016/j.eswa.2020.113868.
2. Kumar S., Tiwari P., Zymbler M.L. Internet of Things is a revolutionary approach for future technology enhancement: a review. J. Big Data. 2019. Vol. 6. P. 111. DOI: 10.1186/S40537-019-0268-2.
3. Seyedan M., Mafakheri F. Predictive big data analytics for supply chain demand forecasting: methods, applications, and research opportunities. J. Big Data. 2020. Vol. 7, no. 1. P. 53. DOI: 10.1186/S40537-020-00329-2.
4. Jensen S.K., Pedersen T.B., Thomsen C. Time Series Management Systems: A Survey. IEEE Trans. Knowl. Data Eng. 2017. Vol. 29, no. 11. P. 2581–2600. DOI: 10.1109/TKDE.2017.2740932.
5. Ivanova E.V., Zymbler M.L. Overview of Modern Time Series Management Systems. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2020. Vol. 9, no. 4. P. 79–97. DOI: 10.14529/cmse200406.
6. Shen C., Ouyang Q., Li F., *et al.* Lindorm TSDB: A Cloud-native Time-series Database for Large-scale Monitoring Systems. Proc. VLDB Endow. 2023. Vol. 16, no. 12. P. 3715–3727. DOI: 10.14778/3611540.3611559.
7. Ivanova E.V., Zymbler M.L. Embedding of the Matrix Profile Concept Into a Relational DBMS for Time Series Mining. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2021. Vol. 10, no. 3. P. 72–87. DOI: 10.14529/cmse210305.
8. Khalefa M.E., Fischer U., Pedersen T.B., Lehner W. Model-based Integration of Past & Future in TimeTravel. Proc. VLDB Endow. 2012. Vol. 5, no. 12. P. 1974–1977. DOI: 10.14778/2367502.2367551.
9. Fischer U., Rosenthal F., Lehner W. F2DB: The Flash-Forward Database System. IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012 / ed. by A. Kementsietsidis, M.A.V. Salles. IEEE Computer Society, 2012. P. 1245–1248. DOI: 10.1109/ICDE.2012.117.
10. Agarwal A., Alomar A., Shah D. tspDB: Time Series Predict DB. NeurIPS 2020 Competition and Demonstration Track, 6-12 December 2020, Virtual Event, Vancouver, BC, Canada. Vol. 133 / ed. by H.J. Escalante, K. Hofmann. PMLR, 2020. P. 27–56. Proceedings of Machine Learning Research. URL: http://proceedings.mlr.press/v133/agarwal21a.html.
11. Arous I., Khayati M., Cudré-Mauroux P., *et al.* RecovDB: Accurate and Efficient Missing Blocks Recovery for Large Time Series. 35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019. IEEE, 2019. P. 1976–1979. DOI: 10.1109/ICDE.2019.00218.

12. Ariyo A.A., Adewumi A.O., Ayo C.K. Stock Price Prediction Using the ARIMA Model. UKSim-AMSS 16th International Conference on Computer Modelling and Simulation, UKSim 2014, Cambridge, United Kingdom, March 26-28, 2014 / ed. by D. Al-Dabass, A. Orsoni, R.J. Cant, *et al.* IEEE, 2014. P. 106–112. DOI: 10.1109/UKSIM.2014.67.

13. Salinas D., Flunkert V., Gasthaus J., Januschowski T. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. International Journal of Forecasting. 2020. Vol. 36, no. 3. P. 1181–1191. DOI: 10.1016/j.ijforecast.2019.07.001.

14. Lim B., Arik S.Ö., Loeff N., Pfister T. Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting. CoRR. 2019. Vol. abs/1912.09363. arXiv: 1912.09363. URL: http://arxiv.org/abs/1912.09363.

15. Cao W., Wang D., Li J., *et al.* BRITS: Bidirectional Recurrent Imputation for Time Series. Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada / ed. by S. Bengio, H.M. Wallach, H. Larochelle, *et al.* 2018. P. 6776–6786. URL: https://proceedings.neurips.cc/paper/2018/hash/734e6bfcd358e25ac1db0a4241b95651-Abstract.html.

16. Yoon J., Zame W.R., Schaar M. van der Estimating Missing Data in Temporal Data Streams Using Multi-Directional Recurrent Neural Networks. IEEE Trans. Biomed. Eng. 2019. Vol. 66, no. 5. P. 1477–1490. DOI: 10.1109/TBME.2018.2874712.

17. Fortuin V., Baranchuk D., Rätsch G., Mandt S. GP-VAE: Deep Probabilistic Time Series Imputation. The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]. Vol. 108 / ed. by S. Chiappa, R. Calandra. PMLR, 2020. P. 1651–1661. Proceedings of Machine Learning Research. URL: http://proceedings.mlr.press/v108/fortuin20a.html.

18. Du W., Côté D., Liu Y. SAITS: Self-attention-based imputation for time series. Expert Syst. Appl. 2023. Vol. 219. P. 119619. DOI: 10.1016/J.ESWA.2023.119619.

19. Oh E., Kim T., Ji Y., Khyalia S. STING: Self-attention based Time-series Imputation Networks using GAN. CoRR. 2022. Vol. abs/2209.10801. DOI: 10.48550/ARXIV.2209.10801. arXiv: 2209.10801.

20. Fang C., Wang C. Time Series Data Imputation: A Survey on Deep Learning Approaches. CoRR. 2020. Vol. abs/2011.11347. arXiv: 2011.11347. URL: https://arxiv.org/abs/2011.11347.

21. Wang J., Du W., Cao W., *et al.* Deep Learning for Multivariate Time Series Imputation: A Survey. CoRR. 2024. Vol. abs/2402.04059. DOI: 10.48550/ARXIV.2402.04059. arXiv: 2402.04059.

22. Silberschatz A., Korth H.F., Sudarshan S. Database System Concepts, Seventh Edition. McGraw-Hill Book Company, 2020. URL: https://www.db-book.com/.

23. Yurtin A.A. Imputation of Multivariate Time Series Based on the Behavioral Patterns and Autoencoders. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2024. Vol. 13, no. 2. P. 39–55. DOI: 10.14529/cmse240203.

24. Zymbler M., Goglachev A. Fast summarization of long time series with graphics processor. Mathematics. 2022. Vol. 10, no. 10. P. 1781. DOI: 10.3390/math10101781.

25. Imani S., Madrid F., Ding W., *et al.* Introducing time series snippets: A new primitive for summarizing long time series. Data Min. Knowl. Discov. 2020. Vol. 34, no. 6. P. 1713–1743. DOI: 10.1007/s10618-020-00702-y.

26. Bilenko R.V., Dolganina N.Y., Ivanova E.V., Rekachinsky A.I. High-performance Computing Resources of South Ural State University. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2022. Vol. 11, no. 1. P. 15–30. DOI: 10.14529/cmse220102.

27. Zymbler M.L., Polonsky V.A., Yurtin A.A. On One Method of Imputation Missing Values of a Streaming Time Series in Real Time. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2021. Vol. 10, no. 4. P. 5–25. DOI: 10.14529/cmse210401.