

УДК 004.85

Шифр специальности ВАК: 2.3.1

В.В. Коломиец

Автоматическое индексирование баз данных с использованием обучения с подкреплением

Актуальность. Индексы существенно влияют на производительность систем управления базами данных, их автоматический выбор при изменяющейся нагрузке остается сложной многокритериальной задачей. **Цель исследования.** Разработать запрос-ориентированный метод автоматического управления индексами на основе обучения с подкреплением для работы в высоконагруженной среде. **Методы.** Инструменты и подходы, которые применял автор: постановка задачи в виде марковского процесса принятия решений, сравнительный анализ современных подходов к индексированию, оценочный механизм оптимизатора запросов, эксплуатационная телеметрия СУБД, проверка результата на нагрузках ТРС-Н. **Научная новизна.** Предложен метод автоматической индексации с использованием обучения с подкреплением, в котором индексное действие выбирается в локальном контексте оптимизируемого запроса, а суррогатная награда оптимизатора отделена от фактической награды, вычисляемой по статистике выполненных запросов. **Результаты.** В последовательном, конкурентном и синтетическом сценариях стратегия обучения с подкреплением превзошла базовую индексацию по времени выполнения и интегральным метрикам производительности. **Практическая значимость.** Метод может применяться как компонент безопасного контура автоматической настройки индексов в промышленных СУБД.

Ключевые слова: обучение с подкреплением, автоматическое индексирование, запрос-ориентированный выбор индексов, динамическая настройка индексов, физическое проектирование баз данных, статистика выполнения запросов, оценка стоимости запросов.

DOI: 10.21293/1818-0442-2026-29-2-116-128

Введение

Индексы являются одним из базовых механизмов физического проектирования баз данных и непосредственно определяют методы доступа к данным, объем чтений и время выполнения запросов [1–3]. Наличие подходящего индекса может на несколько порядков снижать объем обрабатываемых данных. Вместе с тем построение и сопровождение индексов сопряжены с дополнительными затратами вычислительных и дисковых ресурсов, а также с возможным возникновением блокировок на объекты базы данных. Вследствие этого возникает риск снижения доступности системы. По данной причине задача выбора индексов должна рассматриваться как многокритериальная.

Классические средства автоматического проектирования индексов, развивавшиеся со времен AutoAdmin, основаны, главным образом, на анализе фиксированной рабочей нагрузки и оценке полезности кандидатов в индексы через оценочный интерфейс оптимизатора запросов [1, 2]. Такие методы доказали практическую ценность и остаются важным инструментом администратора баз данных (англ. *Database Administrator*, DBA), однако имеют ряд ограничений. Во-первых, они ориентированы преимущественно на отложенную настройку по статическому снимку рабочей нагрузки. Во-вторых, они плохо отражают ситуацию, когда нагрузка дрейфует, набор запросов постепенно меняется, а цена ошибки оказывается накопительной: один неудачный индекс ухудшает записи, сужает бюджет и искажает последующие решения. В-третьих, само пространство индексных конфигураций комбинаторно велико, уже для умеренного числа таблиц и атрибутов задача выбора близка к NP-трудной постановке [4, 5].

Здесь рациональным становится применение обучения с подкреплением (*reinforcement learning*, RL). В отличие от одношаговых оптимизационных постановок, RL рассматривает систему как среду, в которой агент последовательно принимает решения, наблюдает реакцию среды и стремится максимизировать долгосрочную совокупную выгоду [6, 7]. Создание индекса – не изолированное действие, а шаг, меняющий состояние СУБД: текущее множество индексов, планы оптимизатора, доступный бюджет памяти, стоимость записи и набор будущих релевантных кандидатов. При этом эффект решения может быть отложенным: новый индекс строится некоторое время, затем начинает использоваться не всеми запросами сразу, а его реальная выгода проявляется лишь после накопления эксплуатационной статистики. RL позволяет описать именно такую последовательную, частично наблюдаемую и стохастическую динамику.

Содержательно RL опирается на несколько базовых понятий: агент, среда, состояние, действие, награда и политика [6]. Агентом в рассматриваемой задаче выступает модуль автоматического индексирования. Средой является сама СУБД вместе с данными, запросами, статистиками и эксплуатационными ограничениями. Состояние кодирует наблюдаемую конфигурацию среды в конкретный момент времени; действие соответствует изменению физического дизайна хранения данных, а награда измеряет, насколько полезным оказалось это изменение. Политика задает правило выбора действий по состоянию. В простейшем варианте политика может учиться выбирать один из нескольких индексов-кандидатов, а в более зрелых постановках – одновременно учитывать тип индекса, бюджет, стоимость обслуживания и риск блокировок. Таким образом, обучение с под-

креплением позволяет дать широкую постановку задачи автоматического индексирования.

Прямое применение классических алгоритмов обучения с подкреплением в промышленной СУБД требует существенной адаптации. Агент должен принимать решения безопасно по отношению к текущей нагрузке, учитывать, что каждый новый индекс повышает затраты на изменение данных и сам процесс создания индекса зачастую влияет на работу базы данных. Поэтому современные исследования развиваются сразу по нескольким направлениям – сокращение и обобщение описания рабочей нагрузки, уменьшение числа рассматриваемых действий, построение более содержательного описания состояния среды, учет ограниченности вычислительных ресурсов, учет особенностей текущей нагрузки [5, 8, 9–11]. В данной работе сформулирована широкая постановка задачи автоматического индексирования в терминах обучения с подкреплением, охватывающая как ранние, так и современные исследования. На этой основе будет проведен сравнительный обзор исследований по крупным смысловым блокам: как авторы описывают среду и состояние, формируют действия, считают награду, какие архитектуры агентов используют и как обучают модели. После этого предлагается новый алгоритм безопасного автоматического индексирования СУБД, основанный на продолжающейся задаче MDP с дискретными действиями, признаках, сформированных на основе нагрузки, локализованной по конкретному запросу и раздельной суррогатной и реальной награде.

Обобщенная постановка задачи автоматического индексирования в терминах обучения с подкреплением

В общем виде задача автоматического индексирования – это процесс последовательного управления физическим дизайном хранения данных. Пусть в момент времени t среда описывается как

$$\mathbf{E}_t = \mathbf{W}_t, \Sigma_t, \mathbf{I}_t, \Omega_t,$$

где $\mathbf{W}_t$ – актуальная рабочая нагрузка или ее релевантный фрагмент, Σ_t – статистическая информация о выполнении запросов, $\mathbf{I}_t$ – текущая конфигурация индексов, Ω_t – эксплуатационные ограничения: бюджет памяти, допустимость операций определения данных (англ. Data Definition Language, DDL), временные окна обслуживания и т.д. Тогда задача агента состоит в том, чтобы выбирать действия, переводящие систему в более выгодные состояния с учетом долгосрочных последствий.

Состояние должно быть достаточно информативным и универсальным, чтобы определять полезные действия и модель оставалась обучаемой и переносимой между разными схемами баз данных. Наблюдение $\mathbf{o}_t = \psi(\mathbf{E}_t)$ описывается как

$$\mathbf{o}_t = \psi(\mathbf{W}_t, \Sigma_t, \mathbf{I}_t, \Omega_t). \quad (1)$$

Действие a_t может соответствовать построению нового индекса, удалению или модификации суще-

ствующего индекса, выбору типа индекса или перераспределению бюджета. Поэтому пространство действий целесообразно описывать как множество допустимых физических преобразований $\mathbf{A}_t = \mathbf{A}(\mathbf{o}_t)$.

Для эпизодического обучения терминальным состоянием обычно служит исчерпание бюджета индексов или достижение предельного числа шагов. Для неэпизодических сценариев явного терминального состояния нет, а агент периодически переоценивает ситуацию и принимает очередное локальное решение.

$$a_t \in \mathbf{A}(\mathbf{o}_t), \pi(a_t | \mathbf{o}_t) \rightarrow \max E_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right], \quad (2)$$

где γ^t – коэффициент дисконтирования будущей награды r_t .

Награда r_t должна отражать не только увеличение производительности, но и сопутствующие издержки. Для СУБД лучше задавать награду как разность между приростом производительности одних запросов и штрафами за уменьшение производительности других, штрафами за занимаемое пространство и стоимость DDL-операций. В общем виде такую функцию можно записать следующим образом:

$$r(t) = \Delta U_t^{\text{выигр}} - \lambda_w \Delta U_t^{\text{пот}} - \lambda_s \Delta S_t - \lambda_d \Delta \text{DDL}_t, \quad (3)$$

здесь $\Delta U_t^{\text{выигр}}$ обозначает выигрыш в производительности; $\Delta U_t^{\text{пот}}$ – ухудшение производительности; ΔS_t – ограничения бюджета; ΔDDL_t – стоимость физического изменения конфигурации, а $\lambda_w, \lambda_s, \lambda_d$ – коэффициенты эксплуатационных предпочтений. В конкретных работах эти компоненты могут измеряться по-разному: через реальные времена выполнения, оценку стоимости оптимизатора, или через аппроксимации, полученные на основе моделей, обученных на результатах выполненных запросов. То есть под наградой понимается то, насколько выбранное действие улучшило выполнение рабочей нагрузки.

На рис. 1 показана обобщенная схема взаимодействия агента и среды в задаче автоматического индексирования – «наблюдение → действие → изменение конфигурации → награда → новое наблюдение», которая лежит в основе всех последующих исследований.

Сравнительный анализ исследований по автоматическому индексированию на основе RL

Ниже проведен обзор работ в исследуемой области за последние 10 лет в терминах, определенных в постановке задачи: представление среды и состояния, пространство действий, функции награды, архитектуры агентов и методология обучения.

Представление среды и состояния

Развитие методов особенно хорошо прослеживается по тому, как авторы описывают состояние среды.



Рис. 1. Обобщенная схема автоматического индексирования в терминах обучения с подкреплением

В ранних работах это описание было сравнительно компактным и зависело от конкретной структуры базы данных – набора таблиц, столбцов и допустимых индексов. Так, в rCOREIL состоянием служил текущий набор уже созданных вторичных индексов. Основное внимание в этом подходе уделялось тому, насколько выбранный набор индексов уменьшает стоимость выполнения запросов и насколько дорого обходится его сопровождение при вставке, обновлении и удалении данных. В самой модели это состояние представлялось через признаки, показывающие наличие или отсутствие индексов в текущей конфигурации [12].

В NoDBA описание состояния стало более содержательным: рабочая нагрузка задавалась матрицей селективностей по столбцам, а текущая конфигурация – двоичным вектором уже созданных индексов. То есть агент видел не только то, какие индексы существуют, но и то, насколько сильно разные запросы отбирают данные по различным столбцам. Исторически это был важный шаг, поскольку в одном описании впервые были объединены характеристики запросов и характеристики физической организации данных. Однако и такой подход оставался жестко связанным с заранее известным набором таблиц и столбцов, поэтому не переносился на новые базы данных с другой структурой [13].

SMARTIX логичнее рассматривать не как следующий шаг к более полному описанию состояния, а как отдельную упрощенную линию исследований. В этой работе состояние задаётся двоичным вектором по всем индексируемым столбцам базы данных: для каждого столбца фиксируется, существует на нём индекс или нет [14]. Иными словами, агент видит текущую конфигурацию индексирования на уровне отдельных столбцов, но не получает в самом состоянии подробного описания рабочей нагрузки. По сравнению с NoDBA это более простое представление: там в состоянии входили не только уже созданные индексы, но и характеристики запросов, выраженные через селективности по столбцам [13]. Поэтому значение SMARTIX состоит не в усложнении состояния среды, а в непрерывном управлении конфигурацией индексов с действиями создания и удаления, тогда как само описание состояния остаётся предельно компактным.

Следующий этап развития связан с тем, что в состоянии начали включать более подробные сведения о том, как именно запросы обращаются к данным. В DRLindex и ряде близких работ учитывалась не только информация о наличии индексов, но и признаки обращений к данным: какие таблицы и столбцы используются в запросах, по каким столбцам выполняются фильтрация и соединение, насколько избирательны условия отбора, а в системах с репликацией – на каких репликах размещаются данные [15, 16]. Такое описание состояния уже ближе к реальной задаче, поскольку агент получает информацию не только о текущем наборе индексов, но и о том, почему тот или иной индекс может быть полезен.

Часть работ решает ту же проблему иначе – не за счет усложнения состояния, а за счет предварительного ограничения множества рассматриваемых индексов. В работе [4] состояние задается относительно заранее подготовленного набора кандидатов: для каждого такого кандидата фиксируется, построен он или нет. Само состояние при этом получается достаточно простым, поскольку агенту не нужно анализировать все теоретически возможные индексы. Однако эта простота достигается тем, что значительная часть работы выносится на предыдущий этап: система заранее формирует разумный набор кандидатов по эвристическим правилам, в том числе для составных индексов. По существу, здесь действует следующий компромисс: чем сильнее пространство возможных индексов сокращено до начала обучения, тем проще можно описать состояние среды, но тем выше зависимость итогового качества от качества предварительного отбора.

MANTIS [17] и DRLISA [18] важны скорее не как работы с принципиально новым описанием состояния, а как работы, расширяющие саму задачу автоматического индексирования: в первом случае – за счет разделения выбора типа и выбора индекса, во втором – за счет перехода к NoSQL, настройке параметров индекса и работе при изменяющейся нагрузке.

В Indexer++ описание состояния становится еще ближе к реальной эксплуатации [19]. Система анализирует не один фиксированный набор запросов, а последовательность недавних окон рабочей нагрузки.

Это позволяет учитывать не только текущее состояние, но и изменение характера запросов во времени. Если новые окна нагрузки похожи на предыдущие, конфигурация индексов может не пересматриваться. Если же распределение запросов заметно меняется, состояние считается существенно изменившимся, и система получает основание для пересмотра текущего набора индексов. Таким образом, здесь состояние описывает уже не просто «снимок» системы в один момент времени, а недавнюю историю ее поведения.

В наиболее поздних работах основной акцент смещается на переносимость и обобщение. Исследователи стремятся построить такое описание состояния, которое не зависело бы слишком жестко от конкретных имен, таблиц, числа столбцов и заранее фиксированной схемы базы данных. В SWIRL для этого используется компактное описание рабочей нагрузки и метаданных [8]. В IA2 состояние строится вокруг связи между текущей нагрузкой и конкретными индексами-кандидатами [10]. Система оценивает не базу данных в целом, а то, насколько данный кандидат может быть полезен именно для текущего набора запросов. В AutoIndexer эта идея развивается дальше: описание рабочей нагрузки предварительно сжимается, дополняется сведениями из оптимизатора запросов и только после этого используется для принятия решения [11].

В целом можно выделить устойчивую тенденцию. Ранние работы описывали состояние в основном через конфигурацию индексов. Затем в состоянии начали включать свойства запросов и статистику данных. Далее появились подходы, в которых описание строится уже не для всей базы данных сразу, а относительно окна рабочей нагрузки. Появились подходы, решающие не только задачу построения индекса, но и смежные задачи, такие как выбор типа индекса.

Пространство действий и гранулярность управления

Эволюция работ в этой области хорошо видна по тому, как меняется множество действий агента. Например, в NoDBA агент на каждом шаге мог только создать индекс на одном столбце. Индексы добавлялись последовательно, пока их число не достигало заранее заданного предела, после чего эпизод завершался, все индексы удалялись, и обучение начиналось заново. Главное ограничение такой постановки в том, что агент не мог ни удалить неудачный индекс, ни пропустить ход, ни сохранить и постепенно улучшать конфигурацию. Следующий этап развития связан с расширением набора допустимых действий. В работах [4] и в SWIRL агент по-прежнему выбирает индекс-кандидат для создания, но пространство выбора предварительно сужается за счёт генерации кандидатов и исключения недопустимых действий [4, 8]. Иными словами, здесь прогресс связан с сокращением пространства поиска. Отдельная линия развития проявляется в SMARTIX, где агент уже может не только создавать, но и удалять индекс [14]. В более

поздних постановках того же направления появляется и явное действие пропуска шага. Наконец, в DRLISA пространство действий расширяется ещё дальше: система допускает изменение структуры индекса и его параметров, т.е. переходит от простого выбора очередного индекса к управлению текущей индексной конфигурацией [18]. В работе MANTIS рассматриваются уже не только составные индексы, но и несколько типов индексов в пределах заданного бюджета [17].

Награда и способы оценки эффекта индексирования

Функция награды, т.е. правило, по которому агент получает оценку полезности выбранного действия, является одним из ключевых элементов индексирования на основе обучения с подкреплением. В рассмотренных исследованиях можно выделить три основные группы способов задания награды. Первая группа связывает награду с уменьшением суммарной стоимости выполнения рабочей нагрузки. Так, в NoDBA качество конфигурации индексов оценивается по сравнению с исходным вариантом без индексов [13]. В работе [7] награда определяется относительным снижением стоимости рабочей нагрузки после добавления очередного индекса. В MANTIS также используется выигрыш по стоимости, но одновременно учитываются ограничения на доступный объем памяти [17]. Преимущество таких схем состоит в их понятности: агент поощряется за ускорение выполнения запросов. Однако их слабое место заключается в том, что они опираются на модель стоимости, а она не всегда точно отражает поведение реальной системы.

Вторая группа подходов использует награду, основанную не на оценке, а на фактической производительности. Наиболее показательной здесь является работа SMARTIX, где награда вычисляется по метрике QphH@Size, т.е. на основе реального выполнения тестовой нагрузки [14]. Такой подход дает более надежную обратную связь, поскольку агент ориентируется не на прогноз оптимизатора, а на реальные метрики системы. Вместе с тем его практическое применение затруднено высокой ценой такого измерения: для получения награды необходимо запускать нагрузку и наблюдать результат, что слишком дорого для оперативного управления индексами.

В DRLISA награда учитывает два эффекта одновременно – насколько возрастает число обрабатываемых запросов и насколько увеличиваются издержки на вставку, обновление и удаление данных [18]. Это делает постановку более практичной, поскольку индекс должен не только ускорять чтение, но и не создавать чрезмерные нагрузки при вставке и обновлении данных.

Третья группа объединяет подходы, в которых награда учитывает не только выигрыш в производительности, но и затраты ресурсов. В работе Welborn et al. награда определяется через соотношение между снижением стоимости запросов и ростом пространства рассматриваемых решений [20].

В SWIRL выигрыш от индекса сопоставляется с увеличением занимаемого им объема памяти [8]. В BAIT оптимизируется не только качество найденной конфигурации индексов, но и эффективность использования бюджета на её поиск. При этом бюджет измеряется числом вызовов оценочного интерфейса оптимизатора, т.е. числом оценок стоимости для пар «запрос – конфигурация индексов» без реального построения индексов [9].

Почти все существующие работы сталкиваются с одной и той же фундаментальной трудностью: в задачах индексирования награда обычно является отложенной и неоднозначной. Индекс не начинает влиять на систему мгновенно: сначала он должен быть физически построен, затем его эффект может проявиться лишь для части запросов. Кроме того, наблюдаемое изменение времени выполнения запросов не всегда связано именно с новым индексом. Оно может быть вызвано конкуренцией за ресурсы, блокировками, изменением статистик или другими внешними факторами. Во многих исследовательских работах эта проблема упрощается: предполагается, что после каждого действия можно почти сразу получить корректную и однозначно интерпретируемую награду. Для академического эксперимента такое допущение допустимо, однако при переходе к промышленному автоматическому индексированию именно здесь возникает одно из главных расхождений между теоретической моделью и реальной эксплуатацией.

Архитектуры агентов и стратегии обучения

Архитектуры агентов в работах по автоматическому индексированию развивались поэтапно. Сначала использовались методы без глубоких нейронных сетей. Затем появились модели, в которых нейронная сеть стала использоваться для выбора индексов. После этого основной линией развития стало глубокое Q-обучение, а в более поздних работах начали применяться и другие алгоритмы: методы, напрямую обучающие стратегию выбора действий, методы с двумя взаимосвязанными моделями выбора и оценки, а также поиск по дереву [12].

Первый этап представлен работами, такими как rCOREIL. В них агент не использует глубокую нейронную сеть. Вместо этого для каждой конфигурации индексов заранее задается набор признаков, а полезность конфигурации и возможных переходов оценивается как линейная комбинация этих признаков. Обучение в таких работах состоит в том, что по наблюдаемым переходам система постепенно уточняет веса при признаках и тем самым улучшает правлю выбора действий [12].

NoDBA относится к ранним работам по глубокому обучению с подкреплением для индексирования, в качестве обучающего алгоритма используется СЕМ. Для повышения устойчивости обучения авторы используют повторное использование накопленного опыта [13].

Основная группа последующих работ использует глубокое Q-обучение. В этой схеме нейронная сеть получает описание текущего состояния базы

данных и оценивает ожидаемую полезность каждого допустимого действия. Затем агент выбирает действие с наибольшей оценкой. В [4] эта схема сочетается с эвристической генерацией кандидатов, что позволяет применять глубокое Q-обучение не только к одноклоночным, но и к составным индексам. В MANTIS задача разбивается на два этапа: одна модель по рабочей нагрузке определяет тип индекса, а сеть глубокого Q-обучения выбирает конкретный индекс в рамках этого типа [17]. В DRLISA также используется схема обучения по Q-функции, но уже в более широкой постановке, где допускается изменение конфигурации индекса и его параметров [18].

По мере роста числа кандидатов одной только замены линейной модели на нейронную сеть оказалось недостаточно. Поэтому вместе с глубоким Q-обучением стали использоваться дополнительные приемы, которые уменьшают число рассматриваемых действий и делают обучение устойчивее. В [4] эту роль играет предварительная генерация кандидатов. В DQN-AMTAS пространство действий организуется иерархически, а из рассмотрения исключаются недопустимые действия [21]. Тем самым развитие идет не только по линии изменения алгоритма обучения, но и по линии сокращения пространства поиска [4, 21].

После этого в литературе появляются работы, использующие другие алгоритмы. В SWIRL применяется метод, в котором обучается сама стратегия выбора действий, а не только оценка их полезности [8].

В BAIT обучение с подкреплением используется не для прямого выбора индекса, а для управления поиском: агент распределяет ограниченный бюджет вызовов механизма гипотетической оценки индексов и решает, какие конфигурации следует проверить следующими [9]. Таким образом, развитие архитектур агентов идет от ранних методов без глубоких нейронных сетей к моделям с нейронной сетью, затем к глубокому Q-обучению и далее к другим алгоритмам, лучше приспособленным к большим пространствам действий и сложным ограничениям задачи [3, 4, 7–21].

Экспериментальная методология и ограничения практического внедрения

Почти все рассмотренные работы проверяют свои идеи на стандартных тестовых наборах запросов. Чаще всего для этого используются TPC-H и TPC-DS; в задачах, связанных с оперативной обработкой и нереляционными системами, применяют YCSB и наборы на основе IMDB/JOB или специально сгенерированные нагрузки [8–11, 14, 17–19, 22]. Такой подход имеет очевидное достоинство: результаты можно воспроизвести и сопоставить между собой. Однако одновременно он задает и существенное ограничение. Во многих исследованиях обучение и проверка проводятся на сравнительно небольшом числе запросов, на статических или частично искусственных нагрузках и в условиях, где неудачное действие агента не приводит к серьезным последствиям для системы. Поэтому метод, который показывает хорошие результаты в лабораторном эксперименте, не гарантирует надежность при длительной работе на

реальной производственной нагрузке, где одновременно действуют жесткие требования к времени отклика, пропускной способности и стабильности сервиса. По этой причине в современных работах все заметнее смещается акцент от полного многократного перестроения конфигурации индексов к более реалистичным сценариям. Исследователи используют предварительное сжатие и анализ рабочей нагрузки, безопасный отбор кандидатов, ограниченный бюджет гипотетических проверок, проверку способности метода работать на новых нагрузках и гибридные схемы, в которых реальные изменения в рабочей базе данных выполняются редко и только после дополнительной верификации [3, 5, 8, 9, 11].

Краткие выводы по анализу существующих исследований

Проведенный обзор показывает, что сильной стороной RL-подходов является способность моделировать индексирование как последовательное принятие решений. Это особенно важно там, где между индексами существуют сложные взаимодействия, а локально выгодное действие может ухудшить дальнейшую конфигурацию. Работы последних лет показывают, что за счет более содержательного описания состояния среды, исключения заведомо недопустимых действий, сжатого описания рабочей нагрузки и поиска с учетом ограниченности вычислительных ресурсов методы обучения с подкреплением способны выходить за рамки простых жадных схем и находить конкурентоспособные конфигурации индексов [8–11].

В то же время почти все исследовательские линии имеют общие слабые места. Во-первых, недостаточно явно учитывается эксплуатационная безопасность: работа с индексом рассматривается как незатратное действие, хотя в реальной СУБД – это дорогостоящая физическая операция. Во-вторых, отложенная награда обычно упрощается до мгновенной оценки после шага. В-третьих, многие представления состояния остаются либо слишком бедными, либо слишком привязанными к конкретной схеме и набору кандидатов [3, 8, 14, 18].

Следовательно, перспективный путь состоит в том, чтобы сделать агента наблюдательнее и безопаснее: опираться на состояния, ограниченные оптимизируемым запросом, отдельно учитывать суррогатную и реальную награду, ограничивать DDL-операции и применять обучение с подкреплением не как автономный механизм, а как компонент принятия решений в составе более широкой системы, основанной на сборе и обработке эксплуатационной телеметрии.

Запрос-ориентированный метод автоматического индексирования

Метод предназначен для автоматического управления индексной конфигурацией базы данных на основе наблюдаемой рабочей нагрузки. Решение последовательно принимается в рамках конкретного оптимизируемого запроса. В каждом шаге выбирается запрос, дающий наибольший вклад в текущую стоимость нагрузки, вокруг него формируется локальный контекст связанных запросов, после чего происходит

построение множества индексов кандидатов и для каждого индексного кандидата выбирается действие CREATE, DROP или SKIP. Такая постановка уменьшает размер пространства решений и при этом учитывает влияние индекса не только на оптимизируемый запрос, но и на другие запросы, которые обращаются к тем же объектам и используют те же столбцы.

Рабочая нагрузка в момент времени t рассматривается в окне наблюдения $\mathbf{W}_t$. Для запроса q используется оценка стоимости оптимизатора $C_{\text{опт}}(q, t)$. Чтобы выбирать запросы с учетом количества вызовов, вводится нагрузочный вес запроса

$$H(q, t) = \tilde{C}_{\text{опт}}(q, t) \cdot \log(1 + f(q, t)), \quad (4)$$

где $f(q, t)$ – число выполнений запроса q в окне $\mathbf{W}_t$, а $\tilde{C}_{\text{опт}}(q, t)$ – нормированная оценочная стоимость оптимизатора. Нормирование числовых признаков выполняется внутри окна наблюдения по логарифмической шкале

$$\tilde{x} = \frac{\log(1 + x)}{\max_{y \in \mathbf{X}_t} \log(1 + y) + \varepsilon}. \quad (5)$$

Фактические метрики выполнения – логические чтения, процессорное время, длительность, ожидания и признаки сканирований – включаются в состояние агента и используются при расчете фактической награды после совершения действий.

Выбор запроса, локального контекста и индексных кандидатов

Вначале выбирается запрос с максимальным весом. Такой запрос рассматривается как инициатор цикла оптимизации посредством построения индекса. После чего оценка индексного действия формируется на основе локального подмножества рабочей нагрузки, связанного с теми же объектами:

$$q_t^* = \arg \max_{q \in \mathbf{W}_t} H(q, t). \quad (6)$$

Для запроса q_t^* строится локальный контекст $\mathbf{L}_t(q_t^*)$, содержащий запросы из текущего окна наблюдения, которые обращаются к тем же базовым объектам. Контекст нужен для того, чтобы индексное решение оценивалось по группе связанных запросов, работающих с одинаковыми объектами:

$$\mathbf{L}_t(q_t^*) = \{q \in \mathbf{W}_t \mid \mathbf{O}(q) \cap \mathbf{O}(q_t^*) \neq \emptyset\}, \quad (7)$$

где $\mathbf{O}(q)$ – множество базовых объектов, к которым обращается запрос q . На основе иницирующего запроса, локального контекста и текущей индексной конфигурации $\mathbf{I}_t$ формируется множество кандидатов

$$\mathbf{C}_t = G(q_t^*, \mathbf{L}_t(q_t^*), \mathbf{I}_t). \quad (8)$$

Здесь G – генератор новых индексных структур. Он формирует кандидаты по столбцам, участвующим в аргументах поиска, предикатах фильтрации, соеди-

нениях, сортировке и группировке. В множество C_t также включаются существующие индексы I_t .

Каждый кандидат c описывает индексную структуру и имеет признак присутствия $p(c, t) \in \{0, 1\}$: $p(c, t) = 1$ означает, что такая структура уже существует в конфигурации I_t , а $p(c, t) = 0$ означает, что она является потенциальным новым индексом. Для каждого кандидата используется единое пространство действий

$$A = \{\text{CREATE}, \text{DROP}, \text{SKIP}\}. \quad (9)$$

Действие CREATE означает переход к конфигурации, в которой кандидат присутствует; действие DROP – переход к конфигурации, в которой кандидат отсутствует; действие SKIP сохраняет текущую конфигурацию. Формально целевое значение признака присутствия после действия задается как

$$p^a(c, t+1) = \begin{cases} 1, & a = \text{CREATE}, \\ 0, & a = \text{DROP}, \\ p(c, t), & a = \text{SKIP}. \end{cases} \quad (10)$$

Такая запись позволяет использовать одно и то же множество действий для новых и уже существующих индексных структур.

Множество затронутых запросов и кодирование состояния

Для оценки конкретного кандидата используется не весь локальный контекст, а только те запросы, на которые данный кандидат потенциально способен повлиять. Для кандидата c вводится множество смежных запросов:

$$L(c, t) = \{q \in W_t \mid O(q) \cap O(c) \neq \emptyset \wedge R(q, c) \neq \emptyset\}, \quad (11)$$

где $O(c)$ – объект или объекты, для которых определен кандидат c , а $R(q, c)$ – множество ролей столбцов кандидата в запросе q . В $R(q, c)$ включаются признаки того, что столбец используется как аргумент поиска, остаточный предикат, столбец соединения, сортировки, группировки или покрытия результата. Если запрос обращается к тому же объекту, но не использует столбцы кандидата ни в одной из этих ролей, он не включается в $L(c, t)$.

Состояние среды для кандидата c задается вектором признаков

$$s(c, t) = [x_q, x_L, x_T, x_I, x_C, x_E]. \quad (12)$$

Блок x_q описывает инициирующий запрос q_t^* : оценочную стоимость оптимизатора, логические чтения, процессорное время, полную длительность, частоту выполнений, наличие сортировок, выгрузок во временные структуры и сканирований. Ожидания кодируются по нескольким устойчивым классам: блокировки, ввод-вывод, ожидание процессора и сетевые ожидания. Все временные величины выражаются в миллисекундах и нормируются внутри окна наблюдения.

Блок x_L содержит агрегированные характеристики множества $L(c, t)$: суммарную частоту выполнений, суммарную оценочную стоимость, средние и максимальные значения логических чтений, процессорного времени и длительности, а также доли запросов, для которых кандидат может использоваться как индекс поиска, как покрывающий индекс или как индекс, сохраняющий требуемый порядок для сортировки либо группировки.

Блок x_T описывает объект данных: размер таблицы в байтах, число строк, интенсивность чтения, а также интенсивность операций INSERT, UPDATE и DELETE. В методе различаются общая интенсивность обращений к объекту и DML-интенсивность: первая учитывает все запросы к объекту, вторая – только операции модификации данных.

Блок x_I описывает текущую индексную конфигурацию на объекте: число существующих индексов, их суммарный размер, наличие индекса с тем же ведущим ключевым столбцом, максимальную длину совпадающего ключевого префикса, долю столбцов кандидата, уже покрытых существующими индексами, и признак существования почти дублирующей структуры.

Блок x_C описывает сам кандидат: число ключевых столбцов, число включенных столбцов, предполагаемый размер, оценочную стоимость построения, ожидаемый прирост стоимости операций записи, признак покрытия инициирующего запроса и признак сохранения порядка данных. Для существующего индекса эти признаки вычисляются по его текущему определению; для нового кандидата – по его предполагаемому определению.

Блок x_E содержит эксплуатационные условия, влияющие на допустимость изменения: доступное место, оценочную нагрузку на журнал транзакций, текущий уровень параллелизма, наличие конкурирующих операций изменения структуры и признак допустимого эксплуатационного окна. Эти признаки входят в состояние агента, но окончательное разрешение на изменение все равно выдается отдельным контуром безопасного применения.

Для запросов из $L(c, t)$ дополнительно определяется степень пригодности кандидата:

$$\psi(q, c) = \begin{cases} 0, & \text{нет изменения доступа,} \\ 1, & \text{поиск без покрытия,} \\ 2, & \text{покрытие запроса,} \\ 3, & \text{покрытие и порядок.} \end{cases} \quad (13)$$

Уровень 0 означает, что кандидат не меняет способ доступа; уровень 1 – что кандидат позволяет выполнить поиск по ключу, но требует последующего обращения к данным таблицы; уровень 2 – что кандидат покрывает запрос; уровень 3 – что кандидат одновременно покрывает запрос и сохраняет требуемый порядок для сортировки или группировки. На основе этой шкалы задается вес запроса при оценке кандидата

$$w(q, c, t) = \frac{f(q, t) C_{\text{опт}}(q, t) (1 + \psi(q, c))}{\sum_{u \in L(c, t)} f(u, t) C_{\text{опт}}(u, t) (1 + \psi(u, c)) + \varepsilon}. \quad (14)$$

Чтобы выявлять дублирование и близкие замены существующих индексов, вычисляется мера структурной близости кандидата к текущей конфигурации

$$D(c, I_t) = \max_{i \in I_t} (\alpha_p P(c, i) + \alpha_{\text{вкл}} \text{Inc}(c, i) + \alpha_o \text{Ord}(c, i)), \quad (15)$$

где $P(c, i)$ – доля совпадающего ключевого префикса; $\text{Inc}(c, i)$ – доля включенных столбцов кандидата, уже присутствующих в индексе i , а $\text{Ord}(c, i)$ – совместимость порядка ключевых столбцов с требованиями сортировки и группировки. Значение $D(c, I_t)$ используется как признак состояния и как возможный штраф при создании почти дублирующего индекса.

Суррогатная и фактическая награда

Индексное действие сначала оценивается в гипотетической конфигурации, а затем проверяется по фактической статистике. Пусть $I_t^{c,a}$ – конфигурация, полученная из I_t после применения действия a к кандидату c . Для каждого запроса $q \in L(c, t)$ оптимизатор оценивает стоимость выполнения в конфигурациях I_t и $I_t^{c,a}$. Относительный ожидаемый выигрыш определяется как

$$\Delta \hat{C}(q, c, a, t) = \frac{\hat{C}(q, I_t) - \hat{C}(q, I_t^{c,a})}{\max(\hat{C}(q, I_t), \varepsilon)}. \quad (16)$$

Суррогатная награда действия задается выражением

$$\hat{r}(c, a, t) = \sum_{q \in L(c, t)} w(q, c, t) \Delta \hat{C}(q, c, a, t) - \lambda_m \Delta M(c, a, t) - \lambda_s \Delta S(c, a, t) - \lambda_b B(c, a, t). \quad (17)$$

Здесь $\Delta M(c, a, t)$ – ожидаемое изменение стоимости операций INSERT, UPDATE и DELETE; $\Delta S(c, a, t)$ – изменение занимаемого пространства; $B(c, a, t)$ – стоимость и операционный риск физического изменения структуры. Величины ΔM и ΔS имеют знак: при создании индекса они обычно положительны, а при удалении индекса могут быть отрицательны, если действие снижает стоимость записи и освобождает место. Для минимально значимого выигрыша $G_{\min} = 0,10$, т.е. 10% взвешенного относительного снижения оценочной стоимости затронутых запросов по сравнению с текущей конфигурацией, допустимое относительное увеличение стоимости операций записи M принимается равным 10%, а бюджет дополнительного пространства S равным 20% от суммарного начального размера данных таблиц. Показатель стоимости и риска B нормируется к

интервалу $([0; 1])$ по фиксированной шкале. Коэффициенты штрафов соответственно рассчитываются по общему правилу

$$\lambda_j = \frac{G_{\min}}{L_j}, \quad j \in m, s, b.$$

После применения изменения система переходит в режим наблюдения. Фактическая награда вычисляется не сразу, а после накопления не менее N выполнений затронутых запросов. Момент расчета определяется как

$$\tau_N = \min \{ \tau > t : \sum_{q \in L(c, t)} n_q(t, \tau) \geq N \}, \quad (18)$$

где $n_q(t, \tau)$ – число выполнений запроса q на интервале от t до τ . Если часть запросов из $L(c, t)$ не выполнялась в период наблюдения, фактическая награда рассчитывается по выполненным запросам с весами, пересчитанными по фактическим частотам.

Для выполненного запроса q используются относительные изменения длительности T , процессорного времени P и логических чтений R

$$\begin{aligned} \Delta T_q &= \frac{T_q^{\text{до}} - T_q^{\text{после}}}{\max(T_q^{\text{до}}, \varepsilon)}, \\ \Delta P_q &= \frac{P_q^{\text{до}} - P_q^{\text{после}}}{\max(P_q^{\text{до}}, \varepsilon)}, \\ \Delta R_q &= \frac{R_q^{\text{до}} - R_q^{\text{после}}}{\max(R_q^{\text{до}}, \varepsilon)}. \end{aligned} \quad (19)$$

Фактическая награда имеет вид

$$\begin{aligned} r(c, a, \tau_N) &= \\ &= \sum_{q \in L_N(c, t)} \bar{w}(q, c, \tau_N) \kappa_q (\beta_T \Delta T_q + \beta_P \Delta P_q + \beta_R \Delta R_q) - \\ &- \mu_m \Delta M^{\text{факт}}(c, a) - \mu_s \Delta S^{\text{факт}}(c, a) - \mu_b B^{\text{факт}}(c, a). \end{aligned} \quad (20)$$

Здесь $L_N(c, t)$ – подмножество затронутых запросов, реально выполненных за период наблюдения, $\bar{w}(q, c, \tau_N)$ – веса, пересчитанные по фактическим частотам, а коэффициент $\kappa_q \in [0, 1]$ снижает вклад запроса, если изменение его длительности в основном связано не с индексом, а с внешними ожиданиями: блокировками, насыщением дисковой подсистемы, ожиданием процессора или сетевыми задержками. После появления фактической награды она заменяет суррогатную награду в обучающей выборке. В фактической награде используются те же коэффициенты, что и для суррогатной награды: $\mu_m = \lambda_m$, $\mu_s = \lambda_s$, $\mu_b = \lambda_b$.

Обучающий агент

Задача выбора индексного действия рассматривается как повторяемая задача принятия решений в

изменяющемся состоянии рабочей нагрузки и текущей индексной конфигурации. На вход агенту подается состояние $s(c, t)$, описывающее кандидата, затронутые запросы, объект данных, текущие индексы и эксплуатационные условия. На выходе агент выбирает одно из трех действий: CREATE, DROP или SKIP.

В качестве обучающей модели используется двойная дуэльная глубокая Q-сеть (англ. Double Dueling Deep Q-Network, Double Dueling DQN). Дуэльная архитектура применяется потому, что для индексного кандидата полезно раздельно оценивать общую перспективность состояния и относительное преимущество конкретного действия. Оценка Q-функции задается как

$$Q(s, a; \theta) = V(s; \theta_V) + \left(A(s, a; \theta_A) - \frac{1}{|A|} \sum_{a' \in A} A(s, a'; \theta_A) \right). \quad (21)$$

Здесь $V(s; \theta_V)$ – оценка ценности состояния, а

$A(s, a; \theta_A)$ – оценка преимущества действия a относительно других действий в том же состоянии. Такая архитектура полезна, когда часть кандидатов заведомо малоперспективна независимо от выбранного действия, например, из-за малого размера таблицы, высокой DML-интенсивности или почти полного дублирования существующей структуры.

Двойная схема используется для уменьшения завышения оценок Q при обучении на шумных данных. Гипотетические оценки оптимизатора могут отличаться от фактического эффекта, а рабочая нагрузка может изменяться между моментом выбора действия и моментом расчета фактической награды. Поэтому выбор следующего действия и оценка его целевого значения разделяются:

$$y_t = r_t + \gamma Q(s_{t+1}, a^*, \theta_t^-), \\ a^* = \arg \max_{a' \in A} Q(s_{t+1}, a'; \theta_t), \quad (22)$$

где θ_t – параметры основной сети; θ_t^- – параметры целевой сети; γ – коэффициент дисконтирования. Обучение выполняется по накопленным переходам вида

$$(s_t, a_t, \hat{r}_t, s_{t+1}) \rightarrow (s_t, a_t, r_t, s_{t+1}), \quad (23)$$

т.е. сначала в буфер помещается переход с суррогатной наградой, а после завершения периода наблюдения эта награда заменяется фактической. Агент не выполняет DDL-операции напрямую. Его результат является рекомендацией, которая передается в контур безопасного применения.

Безопасное применение индексных изменений

Рекомендация агента применяется только после проверки эксплуатационных ограничений. Контур безопасного применения проверяет доступное место, допустимую нагрузку на журнал транзакций, нали-

чие конкурирующих DDL-операций, текущее состояние блокировок, допустимый уровень параллелизма и наличие эксплуатационного окна. Если хотя бы одно ограничение нарушено, действие не применяется, а система переходит к наблюдению без изменения индексной конфигурации.

Физическое изменение конфигурации выполняется не чаще одного раза за цикл. После CREATE или DROP система не принимает новое индексное решение до завершения периода наблюдения и накопления достаточного числа выполнений затронутых запросов. Это ограничение нужно для того, чтобы фактическая награда могла быть связана с конкретным действием, а не с несколькими одновременными изменениями.

Построение индекса выполняется в онлайн-режиме, если такая возможность поддерживается используемой СУБД. При отсутствии безопасного режима выполнения контур применения отклоняет действие или переносит его до подходящего эксплуатационного окна. В результате предложенный метод не является автономным механизмом бесконтрольного изменения схемы. Обучающий агент решает задачу ранжирования индексных действий в локальном контексте нагрузки, а фактическое изменение конфигурации выполняется только через безопасный контур. За счет этого метод сочетает адаптацию к текущей рабочей нагрузке, учет реального эффекта индекса и эксплуатационные ограничения промышленной СУБД.

Результаты

Для оценки производительности стратегии индексирования использовался тестовый набор TPC-H. В качестве системы управления базами данных был выбран Microsoft SQL Server 2022. Цель эксперимента состояла в сравнительной оценке трёх стратегий индексирования в одинаковых условиях: базовой индексации, отсутствия индексов и стратегии, сформированной методом обучения с подкреплением.

Рассматривались три сценария нагрузки. Первый сценарий соответствовал последовательному выполнению 22 аналитических запросов TPC-H в одном потоке.

Второй сценарий моделировал конкурентную нагрузку: два потока параллельно выполняли аналитические запросы, а третий поток выполнял стандартные операции обновления данных RF1 и RF2.

Третий сценарий представлял собой дополнительную синтетическую нагрузку из 100 SQL-запросов к тем же данным. Она использовалась для проверки поведения стратегий индексирования за пределами стандартного набора запросов TPC-H. На рис. 2 представлено сравнение трёх стратегий индексирования при последовательном выполнении 22 запросов TPC-H по суммарному времени выполнения и метрике Power@Size.

Метрика Power@Size используется для оценки последовательного однопоточного выполнения запросов. Времена выполнения 22 запросов агрегируются с использованием геометрического среднего, после чего результат нормируется.

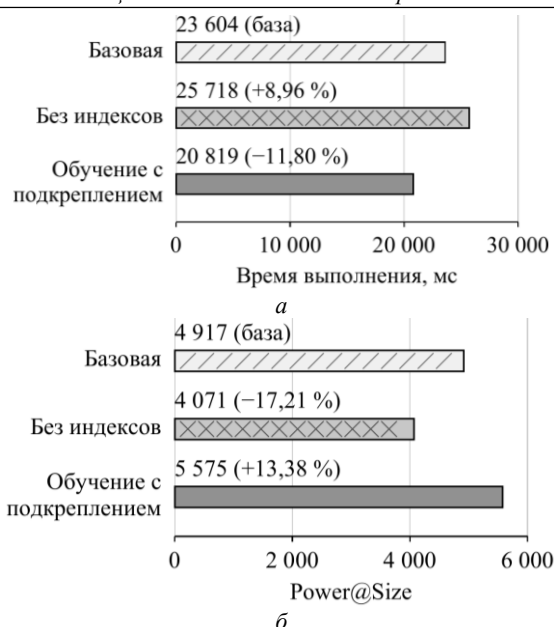


Рис. 2. Результаты последовательного выполнения 22 аналитических запросов TPC-H: а – суммарное время выполнения; б – метрика Power@Size

Этот показатель используется как сравнительная оценка сценариев индексирования.

В первом сценарии при базовой индексации набор из 22 запросов был выполнен за 23604 мс. Среднее время выполнения одного запроса составило 1072,91 мс, а значение интегральной метрики Power@Size – 4917. В сценарии без индексов суммарное время увеличилось до 25718 мс, среднее время запроса – до 1169,00 мс, а значение Power@Size снизилось до 4071.

Таким образом, отказ от индексов в режиме только чтения привёл к ухудшению на 8,96% по общему времени выполнения и на 17,21% по Power@Size. При использовании стратегии обучения с подкреплением общее время выполнения составило 20819 мс, среднее время запроса – 946,32 мс, а значение Power@Size – 5575. Это соответствует улучшению на 11,80% по времени выполнения и на 13,38% по Power@Size относительно базовой индексации.

Метрика Throughput@Size характеризует способность системы обрабатывать конкурентную нагрузку. Она рассчитывается по результатам многопоточного испытания как количество выполненных запросов в час с учётом числа параллельных потоков, общего времени выполнения испытания и масштаба набора данных.

Во втором сценарии при базовой индексации общее время прогона составило 22361 мс. На рис. 3 представлено сравнение трёх стратегий при конкурентном выполнении аналитических запросов и операций обновления по длительности выполнения и метрике Throughput@Size.

Поскольку в испытании использовались два параллельных потока чтения, среднее время одного аналитического выполнения рассчитывалось по 44 выполненным запросам и составило 508,20 мс. Итоговая метрика Throughput@Size составила 7083,60.

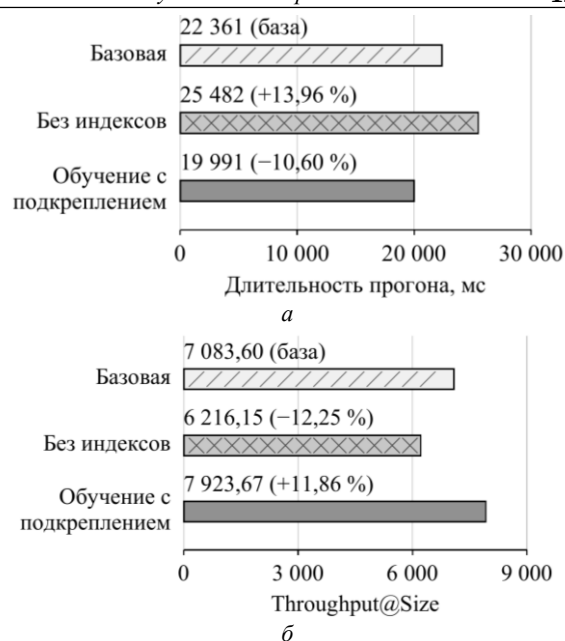


Рис. 3. Результаты выполнения конкурентной нагрузки с двумя потоками аналитических запросов и потоком операций обновления RF1 и RF2: а – длительность прогона; б – метрика Throughput@Size

В сценарии без индексов общее время увеличилось до 25482 мс, среднее время запроса – до 579,14 мс, а значение Throughput@Size снизилось до 6216,15.

Это соответствует ухудшению на 13,96% по общему времени выполнения и на 12,25% по метрике пропускной способности относительно базовой индексации.

Для стратегии обучения с подкреплением общее время выполнения составило 19991 мс, среднее время запроса – 454,34 мс, а значение Throughput@Size – 7923,67. Это соответствует улучшению на 10,60% по времени прогона и на 11,86% по метрике пропускной способности относительно базовой индексации.

В качестве третьего сценария была добавлена дополнительная синтетическая нагрузка из 100 SQL-запросов к тем же данным TPC-H. Нагрузка формировалась шаблонным генератором: были заданы 10 групп запросов, и для каждой группы создано по 10 параметризованных экземпляров, что дало итоговый набор из 100 SQL-запросов. В шаблоны входили фильтры по датам и справочникам, соединения таблиц, агрегирование с GROUP BY, сортировки с выбором первых N строк, подзапросы с EXISTS и NOT EXISTS, а также запросы, в которых соединения порождают большое число связанных строк.

Параметры выбирались из доменов, присутствующих в данных. На рис. 4 представлено сравнение трёх стратегий на синтетической нагрузке из 100 SQL-запросов по суммарному времени выполнения и метрике Power@Size.

Для синтетической нагрузки из 100 запросов при базовой индексации были получены следующие показатели производительности: общее время выполнения – 156 с, среднее время запроса – 1,56 с, условная интегральная метрика Power@Size – 2950.

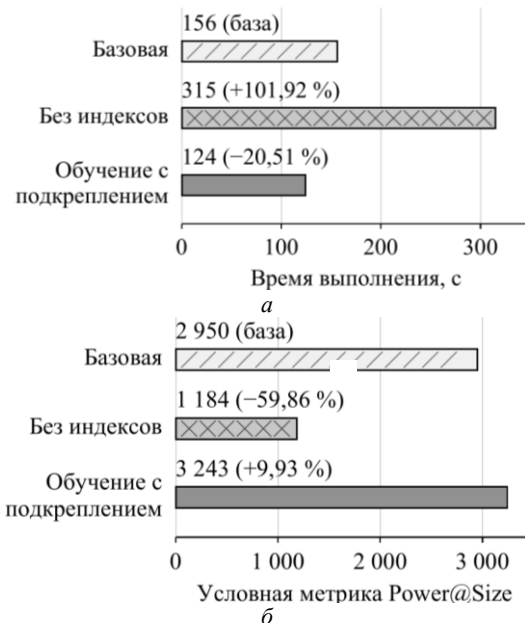


Рис. 4. Результаты выполнения синтетической нагрузки из 100 SQL-запросов: а – суммарное время выполнения; б – условная интегральная метрика Power@Size

Сценарий без индексов оказался наиболее медленным: общее время выполнения составило 315 с, среднее время запроса – 3,15 с, а значение условной интегральной метрики снизилось до 1184. Таким образом, выполнение без индексов оказалось в 2,02 раза медленнее по времени выполнения и хуже на 59,86% по интегральной метрике относительно базового набора индексов. Для стратегии обучения с подкреплением общее время выполнения составило 124 с, среднее время запроса – 1,24 с, а значение условной интегральной метрики – 3243. Это соответствует улучшению на 20,51% по времени выполнения и на 9,93% по производительности относительно базового набора индексов.

Заключение

В работе рассмотрена задача автоматического индексирования баз данных как задача последовательного управления физической структурой хранения данных с использованием обучения с подкреплением. Показано, что выбор индексной конфигурации не сводится к одноразовой настройке для фиксированной рабочей нагрузки: каждое индексное действие изменяет последующее состояние системы, влияет на планы выполнения запросов, занимаемое пространство и множество дальнейших допустимых решений.

Проведенный анализ существующих исследований показал, что методы обучения с подкреплением учитывают последовательный характер задачи индексирования и выходят за пределы жадных стратегий. Вместе с тем выявлены ограничения большинства рассмотренных подходов: недостаточный учет эксплуатационной безопасности, упрощенное представление отложенной награды, зависимость состояния от конкретной схемы базы данных и слабая связь между гипотетической оценкой индекса и фактическим поведением рабочей системы.

Предложен запрос-ориентированный метод автоматического индексирования. Цикл оптимизации начинается с выбора запроса, вносящего наибольший вклад в текущую стоимость рабочей нагрузки. Для каждого индексного кандидата агент выбирает одно из дискретных действий, а оценка действия строится не только по исходному запросу, но и по затронутым запросам, для которых данный индекс может изменить план выполнения. Особенностью метода является разделение суррогатной и фактической награды: первая вычисляется на основе гипотетической оценки оптимизатора, вторая – по статистике выполненных запросов.

Экспериментальная проверка показала, что стратегия, сформированная методом обучения с подкреплением, превзошла базовую индексацию TPC-H во всех рассмотренных сценариях. В последовательном выполнении 22 аналитических запросов получено улучшение на 11,80% по общему времени выполнения и на 13,38% по метрике Power@Size. В конкурентном сценарии улучшение составило 10,60% по времени прогона и 11,86% по Throughput@Size.

На дополнительной синтетической нагрузке стратегия снизила общее время выполнения на 20,51% и повысила условную интегральную метрику производительности на 9,93% относительно базовой индексации.

Полученные результаты подтверждают, что обучение с подкреплением может использоваться как основа для динамического управления индексной конфигурацией базы данных.

Дальнейшее развитие метода связано с расширением экспериментальной проверки на реальные производственные нагрузки, увеличением длительности наблюдения за изменениями запросов, сравнением с промышленными средствами автоматической настройки индексов и переносом подхода на другие типы СУБД.

Литература

1. Agrawal S. AutoAdmin: Self-Tuning Database Systems Technology / S. Agrawal, N. Bruno, S. Chaudhuri, V. Narasayya // IEEE Data Engineering Bulletin. – 2006. – Vol. 29, No. 3. – P. 7–15.
2. Chaudhuri S. Self-Tuning Database Systems: A Decade of Progress / S. Chaudhuri, V. Narasayya // VLDB '07: Proceedings of the 33rd international conference on Information and Knowledge Management (CIKM). – New York: ACM, 2007. – P. 3–14.
3. Wu Y. Automatic Index Tuning: A Survey / Y. Wu, X. Zhou, Y. Zhang, G. Li // IEEE Transactions on Knowledge and Data Engineering. – 2024. – Vol. 36, No. 12. – P. 7657–7676.
4. Lan H. An Index Advisor Using Deep Reinforcement Learning / H. Lan, Z. Bao, Y. Peng // Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM). – New York: ACM, 2020. – P. 2105–2108.
5. Shi J. Learned Index Benefits: Machine Learning Based Index Performance Estimation / J. Shi, G. Cong, X.-L. Li // Proceedings of the VLDB Endowment. – 2022. – Vol. 15, No. 13. – P. 3950–3962.
6. Sutton R.S. Reinforcement Learning: An Introduction / R.S. Sutton, A.G. Barto. – 2nd ed. – Cambridge, MA: MIT Press, 2018. – 552 p.

7. Human-level Control through Deep Reinforcement Learning / V. Mnih, K. Kavukcuoglu, D. Silver et al. // *Nature*. – 2015. – Vol. 518. – P. 529–533.

8. Kossmann J. SWIRL: Selection of Workload-aware Indexes Using Reinforcement Learning / J. Kossmann, A. Kastius, R. Schlosser // *Proceedings of the 25th International Conference on Extending Database Technology (EDBT)*. – Edinburgh, UK, 2022. – P. 155–168.

9. Budget-aware Index Tuning with Reinforcement Learning / W. Wu, C. Wang, T. Siddiqui et al. // *Proceedings of the 2022 International Conference on Management of Data (SIGMOD)*. – USA: ACM, 2022. – P. 1528–1541.

10. Wang T. IA2: Leveraging Instance-Aware Index Advisor with Reinforcement Learning for Diverse Workloads / T. Wang, E. Yoneki // *Proceedings of the 4th Workshop on Machine Learning and Systems (EuroMLSys)*. – New York: ACM, 2024. – P. 10–17.

11. Wang T. AutoIndexer: A Reinforcement Learning-Enhanced Index Advisor Towards Scaling Workloads / T. Wang, E. Yoneki // *arXiv e-prints*. – 2025. – Art. no. arXiv:2507.23084.

12. Basu D. Regularized Cost-Model Oblivious Database Tuning with Reinforcement Learning / D. Basu, Q. Lin, W. Chen et al. // *Transactions on Large-Scale Data- and Knowledge-Centered Systems: Lecture Notes in Computer Science*. – Cham: Springer, 2016. – Vol. 28. – P. 96–132.

13. Sharma A. The Case for Automatic Database Administration using Deep Reinforcement Learning / A. Sharma, F.M. Schuhknecht, J. Dittrich // *arXiv e-prints*. – 2018. – Art. no. arXiv:1801.05643.

14. SMARTIX: A Database Indexing Agent Based on Reinforcement Learning / G.P. Licks, J.C. Couto, P. Mieke et al. // *Applied Intelligence*. – 2020. – Vol. 50, No. 8. – P. 2575–2588.

15. Sadri Z. Online Index Selection Using Deep Reinforcement Learning for a Cluster Database / Z. Sadri, L. Gruenwald, E. Leal // *2020 IEEE 36th International Conference on Data Engineering Workshops (ICDEW)*. – Dallas, TX: IEEE, 2020. – P. 158–161.

16. Sadri Z. DRLindex: Deep Reinforcement Learning Index Advisor for a Cluster Database / Z. Sadri, L. Gruenwald, E. Leal // *Proceedings of the 24th International Database Engineering & Applications Symposium (IDEAS)*. – Seoul, Republic of Korea: ACM, 2020. – Article 11. – P. 1–8.

17. Sharma V. MANTIS: Multiple Type and Attribute Index Selection using Deep Reinforcement Learning / V. Sharma, C. Dyreson, N. Flann // *Proceedings of the 25th International Database Engineering & Applications Symposium (IDEAS)*. – New York, USA, 2021. – P. 56–64.

18. Yan Y. Index Selection for NoSQL Database with Deep Reinforcement Learning / Y. Yan, S. Yao, H. Wang, M. Gao // *Information Sciences*. – 2021. – Vol. 561. – P. 20–30.

19. Sharma V. Indexer++: Workload-Aware Online Index Tuning with Transformers and Reinforcement Learning / V. Sharma, C. Dyreson // *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing (SAC)*. – New York: ACM, 2022. – P. 372–380.

20. Welborn J. Learning Index Selection with Structured Action Spaces / J. Welborn, M. Schaarschmidt, E. Yoneki // *arXiv e-prints*. – 2019. – Art. no. arXiv:1909.07440.

21. Wu Y. Index Advisor via DQN with Invalid Action Mask in Tree-Structured Action Space / Y. Wu, Y. Zhang, N. Li // *Advanced Data Mining and Applications. ADMA 2022. Lecture Notes in Computer Science*. – Cham: Springer, 2022. – Vol. 13781. – P. 434–445.

22. Zhou W. Breaking It Down: An In-depth Study of Index Advisors / W. Zhou, C. Lin, X. Zhou, G. Li // *Proceedings of the VLDB Endowment*. – 2024. – Vol. 17, No. 10. – P. 2405–2418. DOI: 10.14778/3675034.3675035.

Коломиец Виктор Владимирович

Аспирант каф. информационных технологий на транспорте Сибирского государственного университета путей сообщения
Дуси Ковальчук ул., 191, г. Новосибирск, Россия, 630049
Тел.: +7-953-777-72-71

Эл. почта: viccont65@gmail.com

Поступила в редакцию: 17.06.2026.

Принята к публикации: 11.09.2026.

Kolomiets V.V.

Automatic database indexing using reinforcement learning

Relevance. Indexes have a significant impact on the performance of database management systems, and their automatic selection under changing workloads remains a challenging multicriteria problem. **Purpose of the study.** The purpose of the study is to develop a query-oriented method for automatic index management based on reinforcement learning for operation in a high-load environment. **Methods.** The study employs the following tools and approaches: formulation of the problem as a Markov decision process, comparative analysis of modern indexing approaches, the query optimizer's costing mechanism, DBMS operational telemetry, and validation of results on TPC-H workloads. **Scientific novelty.** An automatic indexing method based on reinforcement learning is proposed, in which the indexing action is selected in the local context of the query being optimized, while the optimizer's surrogate reward is separated from the actual reward calculated from the statistics of executed queries. **Results.** In sequential, concurrent, and synthetic scenarios, the reinforcement learning strategy outperformed baseline indexing in terms of query execution time and aggregate performance metrics. **Practical significance.** The proposed method can be used as a component of a safe automatic indexing loop in industrial DBMSs.

Keywords: reinforcement learning, automatic indexing, query-oriented index selection, dynamic index tuning, physical database design, query execution statistics, query cost estimation.

DOI: 10.21293/1818-0442-2026-29-2-116-128

References

1. Agrawal S., Bruno N., Chaudhuri S., Narasayya V. AutoAdmin: Self-Tuning Database Systems Technology. *IEEE Data Engineering Bulletin*, 2006, vol. 29, no. 3, pp. 7–15.
2. Chaudhuri S., Narasayya V. Self-Tuning Database Systems: A Decade of Progress. *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*. Vienna Austria, ACM, 2007, pp. 3–14.
3. Wu Y., Zhou X., Zhang Y., Li G. Automatic Index Tuning: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 2024, vol. 36, no. 12, pp. 7657–7676.
4. Lan H., Bao Z., Peng Y. An Index Advisor Using Deep Reinforcement Learning. *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM)*, New York, ACM, 2020, pp. 2105–2108.
5. Shi J., Cong G., Li X.-L. Learned Index Benefits: Machine Learning Based Index Performance Estimation. *Proceedings of the VLDB Endowment*, 2022, vol. 15, no. 13, pp. 3950–3962.
6. Sutton R.S., Barto A.G. Reinforcement Learning: An Introduction. 2nd ed. Cambridge, MA, MIT Press, 2018, 552 p.
7. Mnih V., Kavukcuoglu K., Silver D. et al. Human-level Control through Deep Reinforcement Learning. *Nature*, 2015, vol. 518, pp. 529–533.

8. Kossmann J., Kastius A., Schlosser R. SWIRL: Selection of Workload-aware Indexes Using Reinforcement Learning. Proceedings of the 25th International Conference on Extending Database Technology (EDBT), Edinburgh, UK, 2022, pp. 155–168.
9. Wu W., Wang C., Siddiqui T. et al. Budget-aware Index Tuning with Reinforcement Learning. Proceedings of the 2022 International Conference on Management of Data (SIGMOD), USA, ACM, 2022, pp. 1528–1541.
10. Wang T., Yoneki E. IA2: Leveraging Instance-Aware Index Advisor with Reinforcement Learning for Diverse Workloads. Proceedings of the 4th Workshop on Machine Learning and Systems (EuroMLSys), New York, ACM, 2024, pp. 10–17.
11. Wang T., Yoneki E. AutoIndexer: A Reinforcement Learning-Enhanced Index Advisor Towards Scaling Workloads. *arXiv e-prints*, 2025, art. no. arXiv:2507.23084.
12. Basu D., Lin Q., Chen W. et al. Regularized Cost-Model Oblivious Database Tuning with Reinforcement Learning. Transactions on Large-Scale Data- and Knowledge-Centered Systems, *Lecture Notes in Computer Science*, Cham, Springer, 2016, vol. 28, pp. 96–132.
13. Sharma A., Schuhknecht F.M., Dittrich J. The Case for Automatic Database Administration using Deep Reinforcement Learning. *arXiv e-prints*, 2018, art. no. arXiv:1801.05643.
14. Licks G.P., Couto J.C., Mieke P. et al. SMARTIX: A Database Indexing Agent Based on Reinforcement Learning. *Applied Intelligence*, 2020, vol. 50, no. 8, pp. 2575–2588.
15. Sadri Z., Gruenwald L., Leal E. Online Index Selection Using Deep Reinforcement Learning for a Cluster Database. 2020 IEEE 36th International Conference on Data Engineering Workshops (ICDEW), Dallas, TX, IEEE, 2020, pp. 158–161.
16. Sadri Z., Gruenwald L., Leal E. DRLindex: Deep Reinforcement Learning Index Advisor for a Cluster Database. Proceedings of the 24th International Database Engineering & Applications Symposium (IDEAS), Seoul, Republic of Korea, ACM, 2020, article 11, pp. 1–8.
17. Sharma V., Dyreson C., Flann N. MANTIS: Multiple Type and Attribute Index Selection using Deep Reinforcement Learning. Proceedings of the 25th International Database Engineering & Applications Symposium (IDEAS), New York, USA, 2021, pp. 56–64.
18. Yan Y., Yao S., Wang H., Gao M. Index Selection for NoSQL Database with Deep Reinforcement Learning. *Information Sciences*, 2021, vol. 561, pp. 20–30.
19. Sharma V., Dyreson C. Indexer++: Workload-Aware Online Index Tuning with Transformers and Reinforcement Learning. Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing (SAC), New York, ACM, 2022, pp. 372–380.
20. Welborn J., Schaarschmidt M., Yoneki E. Learning Index Selection with Structured Action Spaces. *arXiv e-prints*, 2019, art. no. arXiv:1909.07440.
21. Wu Y., Zhang Y., Li N. Index Advisor via DQN with Invalid Action Mask in Tree-Structured Action Space. Advanced Data Mining and Applications. ADMA 2022. *Lecture Notes in Computer Science*, Cham, Springer, 2022, vol. 13781, pp. 434–445.
22. Zhou W., Lin C., Zhou X., Li G. Breaking It Down: An In-depth Study of Index Advisors. *Proceedings of the VLDB Endowment*, 2024, vol. 17, no. 10, pp. 2405–2418. DOI:10.14778/3675034.3675035.

Viktor V. Kolomiets

Postgraduate student, Department of Information Technologies in Transport, Siberian Transport University
191, Dusi Kovalchuk St., Novosibirsk, Russia, 630049
Phone: +7-953-777-72-71
Email: viccont65@gmail.com

Received: 17.06.2026.

Accepted: 11.09.2026.