Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

«Классификация методов построения индексов в базах данных»

Студент: ИУ7-73Б _____ М. Д. Маслова
(группа) (подпись, дата) (И. О. Фамилия)

Преподаватель: _____ А. А. Оленев
(подпись, дата) (И. О. Фамилия)

2022 г.

РЕФЕРАТ

Расчетно-пояснительная записка 18 с., 9 рис., 0 табл., 13 источн., 0 прил.

ИНДЕКСЫ, B-ДЕРЕВЬЯ, ХЕШ-ИНДЕКСЫ, БИТОВЫЕ ИНДЕКСЫ, ОБУЧЕННЫЕ ИНДЕКСЫ, БАЗЫ ДАННЫХ, СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Объектом исследования является построение индексов в базах данных.

Цель работы — классификация методов построения индексов в базах данных.

В разделе 1 рассмотрено понятие индекса в базах данных и его основные свойства, а также описаны типы индексов.

В разделе 2 проведен обзор методов построения индексов на основе B-деревьев, хеш-таблиц и битовых карт, а также соответсвующих обученных индексов.

В разделе 3 приведены критерии оценки качества описанных методов и проведено сравнение по этим критериям.

что-то про результат

СОДЕРЖАНИЕ

РЕФЕРАТ **3**

ВВЕДЕНИЕ **5**

1 Анализ предметной области **6**

1.1 Основные определения 6

1.2 Типы индексов 7

2 Описание существующих методов построения индексов **9**

2.1 Индексы на основе деревьев поиска 9

2.1.1 B-деревья 9

2.1.2 B^+-деревья 9

2.1.3 Обученные индексы 10

2.2 Индексы на основе хеш-таблиц 11

2.2.1 Хеш-индексы 11

2.2.2 Обученные хеш-индексы 12

2.3 Индексы на основе битовых карт 13

2.3.1 Фильтр Блума 13

2.3.2 Обученные индексы 13

3 Классификация существующих методов **15**

ЗАКЛЮЧЕНИЕ **16**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **18**

ВВЕДЕНИЕ

На протяжении последнего десятилетия происходит автоматизация все большего числа сфер человеческой деятельности [1]. Это приводит к тому, что с каждым годом производится все больше данных. Так, по исследованию компании IDC (International Data Corporation), занимающейся изучением мирового рынка информационных технологий и тенденций развития технологий, объем данных к 2025 году составит около 175 зеттабайт, в то время как на год исследования их объем составлял 33 зеттабайта [2].

Для хранения накопленных данных используются базы данных (БД), доступ к ним обеспечивается системами управления базами данных (СУБД), обрабатывающими запросы на поиск, вставку, удаление или обновление. При больших объемах информации необходимы методы для уменьшения времени обработки запросов, одним из которых является построение индексов [3].

Базовые методы построения индексов используют такие структуры, как деревья поиска, хеш-таблицы и битовые карты [4]. На основе данных методов проводятся исследования по разработке новых для уменьшения времени поиска и затрат на перестроение индекса при изменении данных, а также сокращения дополнительно используемой памяти. Одно из таких исследований [5] было проведено в 2018 году, авторы которого описраясь на идею, что обычные индексы не учитывают распределение данных, предложили новый вид индексов, основанный на машином обучении, и назвали их обученные индексы (learned indexes). За последние пять лет было проведено множество исследований [6–9] по совершенствованию обученных индексов в плане поддержки операций и улучшению производительности, поэтому в данной работе в сравнение к базовым приводятся методы построения обученных индексов.

Целью данной работы является ***классификация методов построения индексов в базах данных*** .

Для достижения поставленной цели требуется решить следующие задачи:

- провести анализ предметной области: дать основные определение, описать свойства индексов и их типы;
- описать методы построения индексов в базах данных;
- предложить и обосновать критерии оценки качества описанных методов и сравнить методы по предложенным критериям оценки.

1 Анализ предметной области

1.1 Основные определения

Индекс — это некоторая структура, обеспечивающая быстрый поиск записей в базе данных [10]. Индекс определяет соответствие значения аттрибута или набора аттрибутов — *ключа поиска* — конкретной записи с местоположением этой записи [11]. Это соответствие организуется с помощью индексных записей. Каждая из них соответствует записи в *индексируемой таблице* — таблице, по которой строится индекс — и содержит два поля: идентификатор записи или указатель на нее, а также значение индексированного поля в этой записи [12].

Индексы могут использоваться для поиска по конкретному значению или диапазону значений, а также для проверки существования элемента в таблице, однако обеспечение уменьшения времени доступа к записям в общем случае достигается за счет [11]:

- упорядочивания индексных записей по ключу поиска, что уменьшает количество записей, которые необходимо просмотреть;
- а также меньшего размера индекса по сравнению с индексируемой таблицей, сокращающего время чтения одного элемента.

В то же время индекс является структурой, которая строится в дополнение к существующим данным, то есть он занимает дополнительный объем памяти и должен соответствовать текущим данным. Последнее значит, что индекс необходимо изменять при вставке или удалении элементов, на что затрачивается время, поэтому индекс, ускоряя работу СУБД при доступе к данным, замедляет операции изменения таблицы, что необходимо учитывать[13].

Таким образом, индекс может описываться: [11]:

- *типом доступа* — поиск записей по аттрибуту с конкретным значением, или со значением из указанного диапазона;
- *временем доступа* — время поиска записи или записей;
- *временем вставки*, включающее время поиска правильного места вставки, а также время для обновления индекса;
- *временем удаления*, аналогично вставке, включающее время на поиск удаляемого элемента и время для обновления индекса;
- *дополнительной памятью*, занимаемая индексной стркутурой.

1.2 Типы индексов

Индексы могут быть:

- кластеризованные и некластеризованные;
- плотные и разреженные;
- одноуровневые и многоуровневые;
- а также иметь в своей основе различные структуры, что описывается в следующем разделе, так как исследуется в данной работе.

В *кластеризованных* индексах логический порядок ключей определяет физическое расположение записей, а так как строки в таблице могут быть упорядочены только в одном порядке, то кластеризованный индекс может быть только один на таблицу. Логический порядок *некластеризованных* индексов не влияет на физический, и индекс содержит указатели на записи таблицы [13].

Плотные индексы (рисунок 1.1) содержат ключ поиска и указатель на первую запись с заданным ключом поиска. При этом в кластеризованных индексах другие записи с заданным ключом будут лежать сразу после первой записи, так как записи в таких файлах отсортированны по тому же ключу. Плотные некластеризованные индексы должны содержать список указателей на каждую запись с заданным ключом поиска [11].

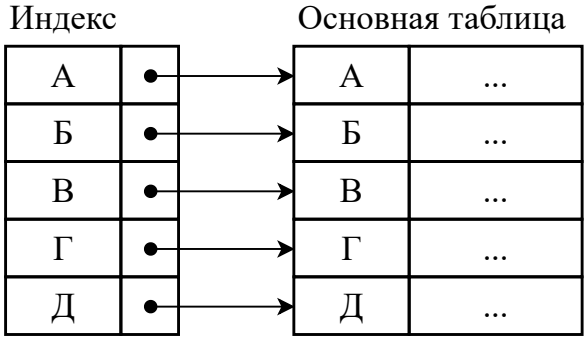

Рисунок 1.1 – Плотный индекс

В *разреженных* индексах (рисунок 1.2) записи содержат только некоторые значения ключа поиска, а для доступа к элементу отношения ищется запись индекса с наибольшим меньшим или равным значением ключа поиска, происходит переход по указателю на первую запись по найденному ключу и далее по указателям в файле происходит поиск заданной записи. Таким образом, разреженные индексы могут быть построены только на отсортированных последовательностях записей, иначе хранения только некоторых ключей поиска

будет недостаточно, так как будет неизвестно, после записи, с каким ключом будет лежать необходимый элемент отношения [11].

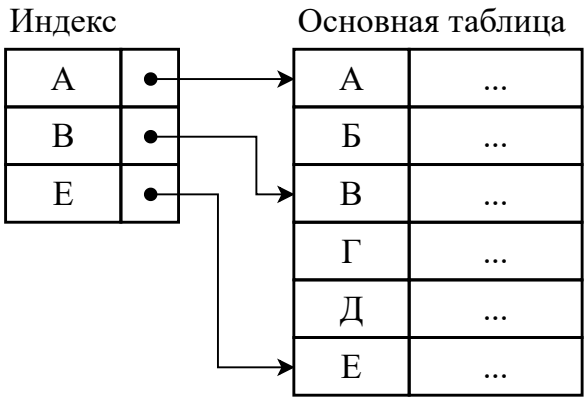

Рисунок 1.2 – Разреженный индекс

Поиск с помощью неразреженных индексов быстрее, так как указатель в записи индекса сразу приводит к необходимым записям. Однако разреженные индексы требуют меньше дополнительной памяти и сокращают время поддержания структуры индекса в актуальном состоянии при вставке или удалении [11].

Одноуровневые индексы ссылаются на данные таблице, индексы же верхнего уровня *многоуровневой* струкуры ссылают на индексы нижестоящего уровня [11] (рисунок 1.3).

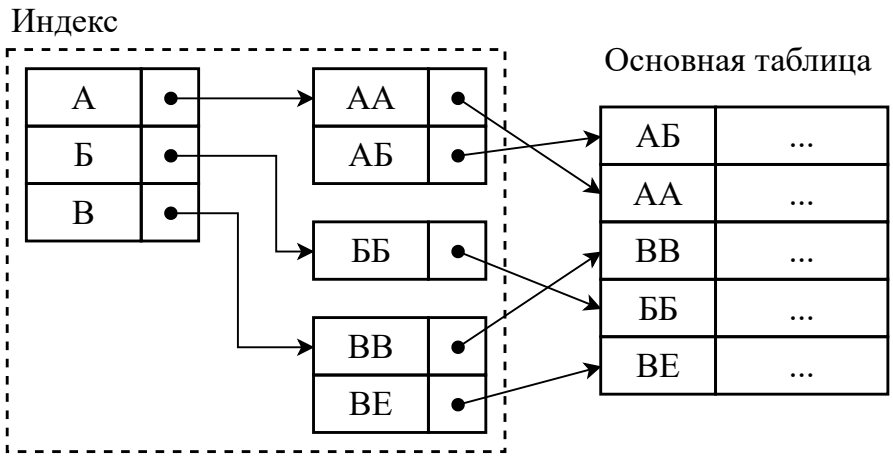

Рисунок 1.3 – Многоуровневый индекс

2 Описание существующих методов построения индексов

Как было сказано выше индексы обеспечивают быстрый поиск записей, поэтому в их основе лежат структуры, предназначенные для решения этой задачи. По данным структурам индексы подразделяются на

- индексы на основе деревьев поиска,
- индексы на основе хеш-таблиц,
- индексы на основе битовых карт.

2.1 Индексы на основе деревьев поиска

Дерево поиска — иерархическая структура, используемая для поска записей, в которой каждый переход на более низкий уровень иерархии уменьшает интервал поиска. При использовании деревьев поиска для построения индексов необходимо учитывать, что требуется обеспечить как ускорение поиска данных, так и уменьшение затрат на обновление индекса при вставках и удалениях. По этим причинам при решении задачи поиска в базах данных используют сбалансированные сильноветвящиеся деревья [**arki**].

В данном случае сбалансированными деревьями назвают такие деревья, что длины любых двух путей от корня до листьев одинаковы [**general**]. Сильноветвящимися же являются деревья, каждый узел которых ссылается на большое число потомков. Эти условия обеспечивают минимальную высоту дерева для быстрого поиска и свободное пространство в узла для внесения изменений в базу данных без необходимости изменения индекса при каждой операции.

Наиболее используемыми деревьями поиска, имеющими описанные свойства, являются B-деревья и их разновидность — B^+-деревья [**arki**].

2.1.1 B-деревья

Определение

Картика

Поиск

Вставка удаление

2.1.2 B^+-деревья

Отличие от B-деревьев

Картинка

2.1.3 Обученные индексы

B-tree индексы можно рассматривать как модель сопоставления ключа позиции искомой записи в отсортированном массиве (рисунок 2.1).

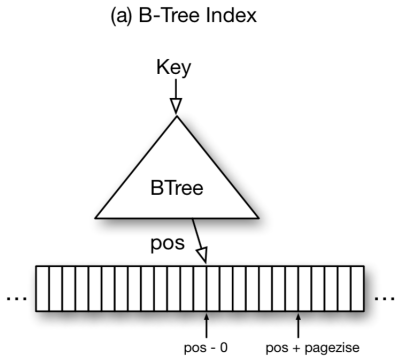

Рисунок 2.1 – B-деревья

Такие индексы как бы предсказывают положение записи с минимаксной ошибкой ($min\_err = 0$, $max\_err = page\_size$). Поэтому можем заминить B-деревья на линейную модель также с минимаксной ошибкой (рисунок 2.2).

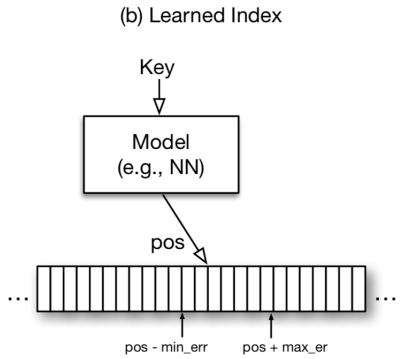

Рисунок 2.2 – Обученный индекс

Так для предсказания можно представлять Range Index Models как модели функции распределения (рисунок 2.3):

$$\text{position} = F(key) \cdot N, \tag{2.1}$$

где $F(key)$ — функция распределения, дающая оценку вероятности обнаружения ключа, меньшего или равного ключу поиска, то есть $P(X < key)$;

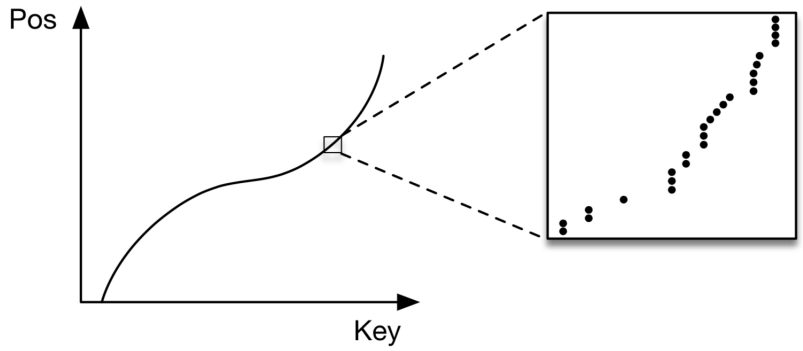

Рисунок 2.3 – Индекс как функция распределения

N — количество ключей.

Можно построить индексы на основе рекурсивной модели (рисунок 2.4), в которой строится иерархия моделей из n уровней. Каждая модель на вход получает ключ, на основе которого выбирает модель на следующем уровне. Модели последнего этапа предсказывают положение записи.

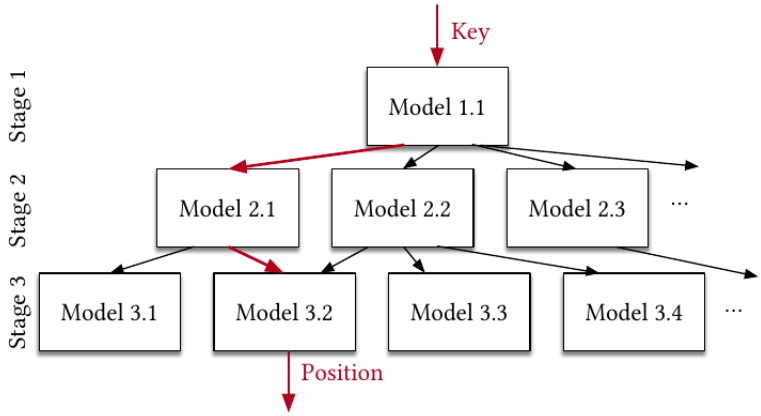

Рисунок 2.4 – Рекурсивная модель индекса

Можно использовать различные модели: например, на верхнем использовать нейронные сети, а на нижних простые линейные регрессионные модели или даже простые B-деревья.

2.2 Индексы на основе хеш-таблиц

2.2.1 Хеш-индексы

Лошадь

Хэш-индексы Одной из простейших реализаций индекса является хэш-карта. Хэш -карта - это набор сегментов, содержащих результаты хэш-функции

, примененной к ключу. Этот хэш указывает на местоположение, где можно найти записи. Хэш-карта пригодна только для поиска по одному ключу, поскольку сканирование диапазона было бы непомерно дорогим. Кроме того, хэш должен помещаться в память для обеспечения производительности. С учетом этих предостережений хэш-карты обеспечивают отличный доступ для конкретных случаев использования, для которых они работают. учше, когда индексируется небольшое количество значений.

only general

Мы можем упорядочивать записи, используя метод, называемый хэшированием, чтобы быстро находить записи , которые имеют заданное значение ключа поиска. Например, если файл записей сотрудников хэширован в поле имя, мы можем получить все записи о Джо. При таком подходе записи в файле группируются в сегменты, где сегмент состоит из основной страницы и, возможно, дополнительных страниц, связанных в цепочку. Корзина, к которой принадлежит запись, может быть определена путем применения специальной функции, называемой хэш-функцией, к ключу поиска. Задан номер корзины, структура индекса на основе хэша позволяет нам извлекать основную страницу для корзины в одном или двух дисковых вводах-выводах. При вставках запись вставляется в соответствующую корзину, при этом страницы "переполнения"выделяются по мере необходимости. Чтобы выполнить поиск записи с заданным значением ключа поиска, мы применяем хэш-функцию для определения корзины, к которой принадлежат такие записи, и просматриваем все страницы в этой корзине. Если у нас нет значения ключа поиска для записи, например, индекс основан на sal, и нам нужны записи с заданным значением возраста, мы должны просканировать все страницы в файле.

2.2.2 Обученные хеш-индексы

Хеш-индексы можно рассматривать как модель сопоставления ключа позиции искомой записи в неупорядоченном массиве.

Функция распределения вероятностей распределения ключей один из возможных способов обучения хеш-индексов. Функция распределения масштабируется на размер хеш-таблицы M и для поиска положения записи аналогично случаю с B-деревьями используется формула:

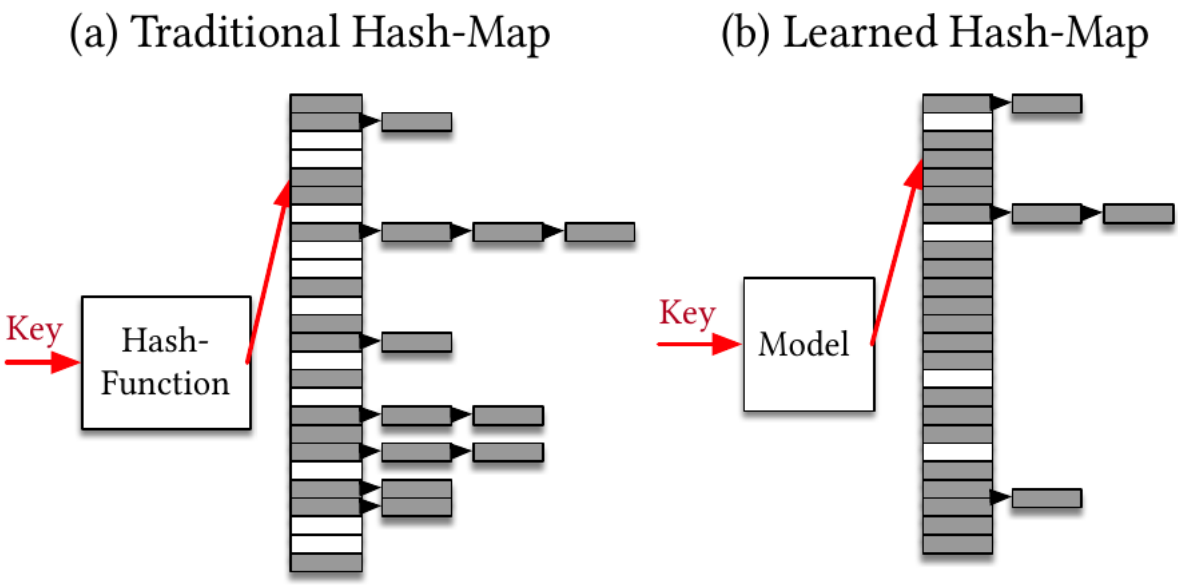

Рисунок 2.5 – Хеш-индексы

$$h(K) = F(K) \cdot M, \tag{2.2}$$

где K — ключ.

2.3 Индексы на основе битовых карт

2.3.1 Фильтр Блума

Камбала 146

Лошадь

Индекс растрового изображения хранит свои данные в виде битовых массивов (растровых изображений). Когда вы проходите по индексу, это делается путем выполнения побитовых логических операций над растровыми изображениями. В B-деревьях индекс лучше всего работает со значениями, которые не повторяются часто. Это также известно как высокая мощность. Индекс растрового изображения работает намного л учше, когда индексируется небольшое количество значений.

2.3.2 Обученные индексы

Данные индексы можно рассматривать как модель проверки существования записи в массиве данных.

Фильтр Блума — алгоритм используемый для проверки существования записи.

Фильтр Блума использует массив бит размером m и k хеш-функций, каждая из которых сопоставляет ключ с одну из m позиций. Для добавления элемента в множество существующих значений ключ подается на вход каждой хеш-функции, возвращающих позицию бита, который должен быть установлен в единицу. Для проверки принадлежности ключа множеству, ключ также подается на вход k хеш-функций. Если какой-либо бит, соответствующий одной из возвращенных позиций, равен нулю, то ключ не входит во множество. Из этого следует, что данный алгоритм гарантирует отсутсвие ложноотрицательных результатов.

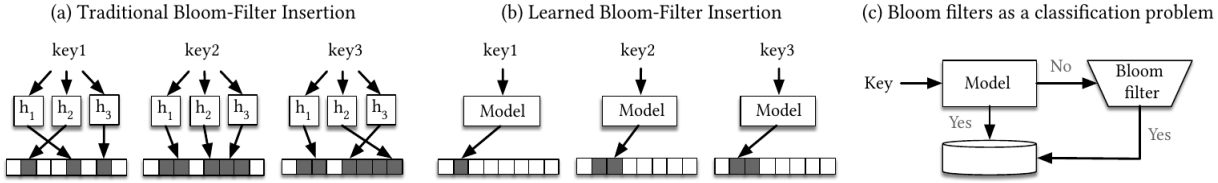

Рисунок 2.6 – Bitmap-индексы

В случае индексов существования необходимо обучить функцию таким образом, чтобы среди возвращенных значений для множества ключей были коллизии, аналогично для множества неключей, но при этом не было коллизий возвращенных значений для ключей и неключей.

В отличие от оригинального фильтра Блума, где $FNR = 0$, $FPR = const$, где const выбрано априори, при обучении достигается заданное значение FPR при $FNR = 0$ на реальных запросах.

3 Классификация существующих методов

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Исследование способов ускорения поисковых запросов в базах данных / Е. В. Коптенок [и др.] // Вестник образовательного консорциума среднерусский университет. Информационные технологии. — 2019. — N 1(13). — С. 24–27.
2. *Reinsel D.*, *Gantz J.*, *Rydning J.* The Digitization of the World From Edge to Core // IDC White Paper. — 2018.
3. *Носова Т. Н.*, *Калугина О. Б.* Использование алгоритма битовых шкал для увеличения эффективности поисковых запросов, обрабатывающих данные с низкой избирательностью // Электротехнические системы и комплексы. — 2018. — N 1(38). — С. 63–67.
4. DAMA-DMBOK : Свод знаний по управлению данными. — 2-е изд. — М. : Олимп-Бизнес, 2020. — 828 с.
5. The Case for Learned Index Structures / T. Kraska [et al.] // Proceedings of the 2018 International Conference on Management of Data. — SIGMOD'18, June 10–15, 2018, Houston, TX, USA, 2018. — P. 489–504.
6. ALEX: An Updatable Adaptive Learned Index / J. Ding [et al.] // Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. — 2020. — P. 969–984.
7. APEX: A High-Performance Learned Index on Persistent Memory (Extended Version) / B. Lu [et al.] // Proceedings of the VLDB Endowment. — 2022. — Vol. 15(3). — P. 597–610.
8. Updatable Learned Index with Precise Positions / J. Wu [et al.] // Proceedings of the VLDB Endowment. — 2021. — Vol. 14(8). — P. 1276–1288.
9. *Ferragina P.*, *Vinciguerra G.* The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds // PVLDB. — 2020. — Vol. 13(8). — P. 1162–1175.
10. *Григорьев Ю. А.*, *Плутенко А. Д.*, *Плужникова О. Ю.* Реляционные базы данных и системы NoSQL: учебное пособие. — Благовещенск : Амурский гос. ун-т, 2018. — 424 с.

11. *Silberschatz A.*, *Korth H. F.*, *Sudarshan S.* Database System Concepts. — New York : McGraw-Hill, 2020. — 1344 p.

12. *Эдвард Сьоре*. Проектирование и реализация систем управления базами данных. — М. : ДМК Пресс, 2021. — 466 с.

13. *Осипов Д. Л.* Технологии проектирования баз данных. — М. : ДМК Пресс, 2019. — 498 с.