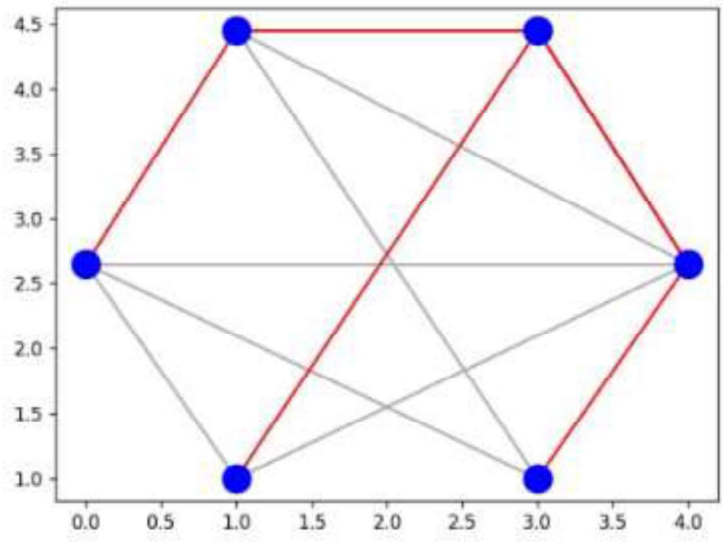

Рисунок 3 – Граф с лучшим маршрутом

В данной статье был рассмотрен типовой алгоритм решения задачи маршрутизации на графе с применением ГА. В дальнейшем развитие программы происходит путем добавления новых вершин и маршрутов, а также наложением новых ограничений на маршрут движения.

Список использованных источников

1. Аллилуева Н.В., Дараган А.Д., Ефремов А.А., Руденко Э.М. Математические аспекты применения генетическо- го алгоритма к решению задачи оптимизации на графах // Труды Московского института теплотехники. 2017. Т. 17. Ч. 1. С. 108-117.

2. Оптимизационные алгоритмы теории графов в программировании: методические материалы к изучению курсов «Дискретная математика», «Методы оптимизации», «Программирование на языке высокого уровня» и выполнению самостоятельных работ для студентов специальности 230101«Вычислительные машины, комплексы системы и сети» / НГТУ; Сост: Л.С. Ломакина, А.С. Суркова, П.И.Уваров, Н. Новгород, 2007. – 32 с

3. Михайлин Д.А., Аллилуева Н.В., Руденко Э.М. Сравнительный анализ эффективности генетических алгоритмов маршрутизации полета с учетом их различной вычислительной трудоемкости и многокритериальности решаемых задач.// Труды Московского авиационного института. Электронный журнал. 2018. Выпуск № 98. Режим доступа: http://trudymai.ru.

АНАЛИЗ ВЛИЯНИЯ ИНДЕКСОВ НА СКОРОСТЬ ВЫПОЛНЕНИЯ SQL-ЗАПРОСОВ

Автор: Макаров А.О., студент третьего курса филиала «Протвино» государственного университета «Дубна».

Научный руководитель: Нурматова Е.В., заведующий кафедрой информационных технологий филиала «Протвино» государственного университета «Дубна».

Аннотация.

Данная работа посвящена исследованию зависимости скорости выполнения SQL-запросов от наличия тех или иных индексов в базе данных.

Abstract

This article is devoted to the research of how SQL-query execution rate depends on presence of particular indexes in a database.

Ключевые слова: базы данных, SQL, запросы, индексы.
Keywords: databases, SQL, queries, indexes.

Цель работы: исследовать то, как на скорость выполнения SQL-запросов влияет наличие определённых индексов на таблицах, к которым идёт обращение.

Для достижения этой цели были поставлены следующие задачи:

1. Дать определение индексов различных типов;
2. Проанализировать влияние индексов на скорость выполнения запросов;
3. Сделать выводы о том, в каких случаях применять те или иные индексы в зависимости от задачи.

Гипотеза: скорость выполнения запросов зависит от того, есть ли индексы на таблицах, к которым идут данные запросы, и как хорошо эти индексы спроектированы.

Предметом исследования данной статьи является зависимость скорости выполнения запросов к базе данных от того, есть ли в ней определённые индексы.

Объект исследования при этом – скорость выполнения SQL-запросов.

Автор данной статьи является одним из разработчиков сервисов по автоматизации объектов электроэнергетики. Эти сервисы непрерывно работают с базой данных, запрашивая из неё информацию, для повышения скорости обработки этих запросов было предложено создание индексов в базе данных с предварительным исследованием того, насколько это предложение будет эффективно. Данная тема актуальна также и по той причине, что вопросы оптимизации выполнения SQL-запросов рассматриваются как при разработке высоконагруженных информационных систем, работающих с большими объёмами информации, так и на собеседованиях в IT-компании, которые в своей деятельности используют базы данных.

Индексы в базах данных – это особые структуры данных, цель которых заключается в упрощении поиска записей в таблицах базы данных. Основу индексов составляют сбалансированные деревья. Поиск в таких деревьях имеет сложность $O(\log n)$, что меньше, чем сложность поиска записей по таблице, которая в худшем случае равна $O(n)$, где n – количество записей, среди которых ведётся поиск.

В СУБД Microsoft SQL Server существует несколько типов индексов, наиболее применяемыми из которых являются:

- Кластеризованный (Clustered), который хранит строки таблицы в отсортированном виде. Сортировка производится по значению ключа индекса. У каждой таблицы может быть только один кластеризованный индекс, так как данные могут быть отсортированы только в одном порядке. По возможности каждая таблица должна иметь кластеризованный индекс. Если у таблицы нет кластеризованного индекса, такая таблица называется «кучей». Кластеризованный индекс создается автоматически при создании ограничений
PRIMARY KEY (первичный ключ) и UNIQUE, если до этого кластеризованный индекс для таблицы еще не был определен. В случае создания кластеризованного индекса для таблицы, в которой есть некластеризованные индексы, то после создания все их необходимо перестроить.
- Некластеризованный (Nonclustered), который содержит значение ключа и указатель на строку данных, содержащую значение этого ключа. У таблицы может быть

несколько некластеризованных индексов. Создаваться некластеризованные индексы могут как на таблицах с кластеризованным индексом, так и без него. Именно этот тип индекса используется для повышения производительности часто используемых запросов, так как некластеризованные индексы обеспечивают быстрый поиск и доступ к данным по значениям ключа;

- Колоночный (Columnstore) – это индекс, основанный на технологии хранения данных в виде столбцов. Данный тип индекса эффективно использовать для больших хранилищ данных, поскольку он может увеличить производительность запросов к хранилищу до 10 раз и также до 10 раз уменьшить размер данных, так как данные в Columnstore индексе сжимаются.

Рассмотрим использование индексов на примере нескольких таблиц, хранящих большие объёмы данных (таблица 1):

Таблица	Атрибут	Количество строк
T1	Key_T1 int	~6,5 млн
	ForeignKey_T1 int	
T2	Key_T2 int	1 млн
	Data_T2 datetime	
T3	Key_Field int	1 млн
	Field_1 int	
	Field_2 int	
	Field_3 int	

Таблица 1 – таблицы базы данных

Изначально на таблицах не созданы никакие индексы, данные в них располагаются в произвольном порядке. Поля со словом Key в названии имеют уникальные значения.

К этим таблицам было сформировано пять запросов, четыре из которых – на выборку из таблиц и один – на вставку и удаление. Время выполнения каждого из них измерялось в два этапов:

1. До создания каких-либо индексов (запросы к «куче»);
2. После создания индексов, которые, по предположению, ускорят выполнение запроса.

Проанализируем выполнение этих запросов по отдельности.

Первый запрос на выборку выглядит следующим образом:

```
select ForeignKey_T1 from T1 where Key_T1 = floor(rand() * (10000000 – 1)).
```

Данный запрос выводит значение поля ForeignKey_T1 таблицы T1, для которого поле Key_T1 равно какому-то случайному числу. Время выполнения этого запроса до создания на таблице T1 кластеризованного индекса по полю Key_T1 равно 364 миллисекундам, тогда как после его создания это время уменьшилось до 4 миллисекунд, то есть в 91 раз. Из этого следует, что поиск в таблице по ключевому полю выполняется значительно быстрее при создании по этому полю кластеризрованного индекса.

Рассмотрим второй запрос на выборку:

```
select count(*) from T1 where ForeignKey_T1 between 1000 and 1000000
group by ForeignKey_T1
```

Запрос подсчитывает количество строк в группах по полю ForeignKey_T1 в таблице T1, причём это поле должно быть в даипазон от тысячи до одного миллиона. Полученные результаты времени выполнения, а также размер созданных индексов представлены на таблице 2.

«Куча»	Некластеризованный индекс по полю ForeignKey_T1	Некластеризованный колоночный индекс по полю ForeignKey_T1
1900 мс	613 мс	540 мс
–	114 МБ	28 МБ

Таблица 2 – Результаты выполнения второго запроса

Как видно из этой таблицы, скорость выполнения запроса возросла в более чем три раза при создании некластеризованного индекса. Создание колоночного индекса вместо обычного некластеризованного помимо уменьшения времени выполнения запроса даёт выигрыш и по месту, которое отводится под этот тип индекса: по сравнению с обычным некластеризованным колоночный индекс занимает в 4 раза меньше дискового пространства.

Далее рассмотрим запрос, в котором соединяются таблицы T1 и T2, и проанализируем влияние индексов на его выполнение.

Запрос выглядит следующим образом:

```
select Key_T1, ForeignKey_T1, T2.Data_T2 from T1 join T2
on T1.ForeignKey_T1 = T2.Key_T2
where Key_T1 = floor(rand() * 4000000 + 6000000)
```

В данном запросе выводятся поля Key_T1, ForeignKey_T1 и Data_T2 из результата соединения таблиц по полям ForeignKey_T1 и Key_T2, причём поле Key_T1 должно быть равно какому-то определённому значению (не столь существенно, какому). Время выполнения этого запроса в зависимости от созданных индексов представлено на таблице 3. В третьей строке таблицы записаны размеры некластеризованных индексов. Размеры кластеризованных и колоночных индексов и «кучи» равны между собой и, следовательно, не указываются.

«Куча»	Кластеризованный индекс на таблице T2 по полю Key_T2, некластеризованынй индекс на таблице T1 по полям ForeignKey_T1, Key_T1	Кластеризованный индекс на таблице T2 по полю Key_T2, колоночный индекс на таблице T1 по полям ForeignKey_T1, Key_T1	Кластеризованный индекс на таблице T2 по полю Key_T2, кластеризованный индекс на таблице T1 по полю Key_T1
447 мс	337 мс	97 мс	3 мс
–	88 МБ	46 МБ	–

Таблица 3 – Результаты выполнения третьего запроса

Как и для предыдущего запроса, колоночный индекс имеет преимущество над некластеризованным как в плане занимаемого места (46 МБ против 88), так и в скорости

выполнения запроса (97 мс против 337), однако создание кластеризованных индексов на обоих соединяемых таблицах показывает ещё более лучший результат времени выполнения – всего 3 миллисекунды, при этом объём занимаемого дискового пространства не увеличился.

Таким образом, на основе полученных результатов можно сформулировать следующие правила применения индексов:

1. Индексы следует создавать по часто используемым в запросах столбцах;
2. Кластеризованные индексы следует создавать всегда, так как они в целом ускоряют работу любых запросов, в том числе и запросов на соединение таблиц;
3. Предпочтительнее использовать колоночные индексы вместо обычных некластеризованных, поскольку первые дают больший выигрыш по сравнению со вторыми как по времени выполнения запросов, так и по занимаемому пространству на диске.

Выводы: было дано определение индексов различных типов, проанализировано влияние индексов на скорость выполнения запросов и сформулированы правила того, в каких случаях применять те или иные индексы от задачи.

Список использованных источников

1. Базы данных : учебник / Е. В. Фешина, В. В. Ткаченко. – Краснодар: КубГАУ, 2020. – 172 с.

2. Волк, В. К. Базы данных. Проектирование, программирование, управление и администрирование : учебник для вузов / В. К. Волк. — 3-е изд., стер. — Санкт-Петербург : Лань, 2022. — 244 с.

3. Уорд, Б. Инновации SQL Server 2019 / Б. Уорд ; перевод с английского Н. Б. Желновой. — Москва : ДМК Пресс, 2020. — 408 с.

4. Эффективное использование СУБД MS SQL Server: учебное пособие. – М.: Финансовый университет, 2017. – 134 с.

5. SQL для простых смертных / Мартин Грабер. - Издательство «ЛОРИ», 2014. - 383 с.

АВТОМАТИЗИРОВАННАЯ СИСТЕМА КОНТРОЛЯ РАБОТЫ КОМПЬЮТЕРНОГО КЛАССА

Автор: Рамодин Н., обучающийся 11 класса МБОУ СОШ № 10 г.о. Серпухов Московской области.

Научный руководитель: Гущина Л. Н., учитель информатики.

Аннотация

Разработка внешнего приложения средствами объектно-ориентированного языка программирования Visual Basic for Application для автоматизации ведения отчетности по использованию компьютерной техники.

Annotetion

Development of an external application using the object-oriented programming language Visual Basic for Application to automate reporting on the use of computer technology.

Ключевые слова: база данных, электронное приложение, внешнее приложение, объектно-ориентированное программирование, предметная область.

Keywords: database, electronic application, external application, object-oriented programming, domain.