1. Министерство науки и высшего образования Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. **Институт кибербезопасности и защиты информации**

**ЛАБОРАТОРНАЯ РАБОТА № 4**

1. «Индексы и производительность»
2. по дисциплине «Системы управления базами данных»
3. Выполнил
4. студент гр. 4851003/70801 Гасанов Э.А.

<*подпись*>

1. Преподаватель
2. ассистент Полтавцева М.А.

<*подпись*>

1. Санкт-Петербург
2. 2021

**Цель**

Оценить влияние индексов на производительность процессов выборки и модификации данных.

**Задачи**

1. Провести эксперимент по определению производительности выборки с использованием индексов. Построить график зависимости времени выборки одной случайной записи от размера таблицы (для таблиц с индексами и без).
2. Провести эксперимент по определению производительности вставки с использованием индексов. Построить график зависимости времени выборки одной случайной записи от размера таблицы (для каждой их таблиц).
3. Провести эксперимент по определению производительности обновления с использованием индексов. Построить график зависимости времени выборки одной случайной записи от размера таблицы (для каждой их таблиц).
4. Задать индексы для отношений, созданных в лабораторной работе №1. Обоснуйте индексы, исходя из запросов, созданных в лабораторной работе №2.
5. Используемая СУБД — MS SQL.

**Ход работы**

Для выполнения данной лабораторной работы была создана новая пустая база данных. В ней было создано несколько таблиц с одинаковой шапкой, как в методическом пособии. Скрипты по созданию таблиц и индексов приведены в приложении 1. Скрипты по заполнению таблиц случайными значениями с указанием, откуда взяты случайные данные, приведены в приложении 2.

В качестве функции для замера времени выполнения использовалась встроенная функция SYSDATETIME(), рекомендованная в методическом пособии. В MS SQL существует целый ряд похожих функций, все они приведены в таблице 1.

|  |  |
| --- | --- |
| Функция | Особенности |
| SYSDATETIME | Имеют точность выше, чем GETDATE, GETUTCDATE |
| SYSUTCDATETIME |
| SYSDATETIMEOFFSET |
| CURRENT\_TIMESTAMP | Эквивалентно GETDATE |
| GETDATE |  |
| GETUTCDATE |  |

Таблица 1. Перечень функций для получения текущего времени

Функция SYSDATETIME была выбрана как самая точная.

Был проведён эксперимент по определению производительности выборки.

Для замера времени использовался следующий скрипт:

Declare @top int

Set @top = 100000

Declare @cnt int

Set @cnt = 0

Declare @\_F1 int

Set @\_F1 = 0

Declare @\_F2 varchar(80)

Set @\_F2 = ''

Declare @Time1 datetime2(7)

Declare @Time2 datetime2(7)

Set @cnt = 0

While @cnt < @top

Begin

select @\_F1 = F1 from arr where idx = @cnt

set @Time1 = SYSDATETIME()

--Insert Into T1 values (@\_F1,@\_F2)

SELECT @\_F2= min(F2) from T1 where F1 = @\_F1

set @Time2 = SYSDATETIME()

set @timesum= @timesum+DATEDIFF(microsecond,@Time1,@Time2)

Set @cnt = @cnt + 1

End

Рассмотрим графики времени выборки (select):

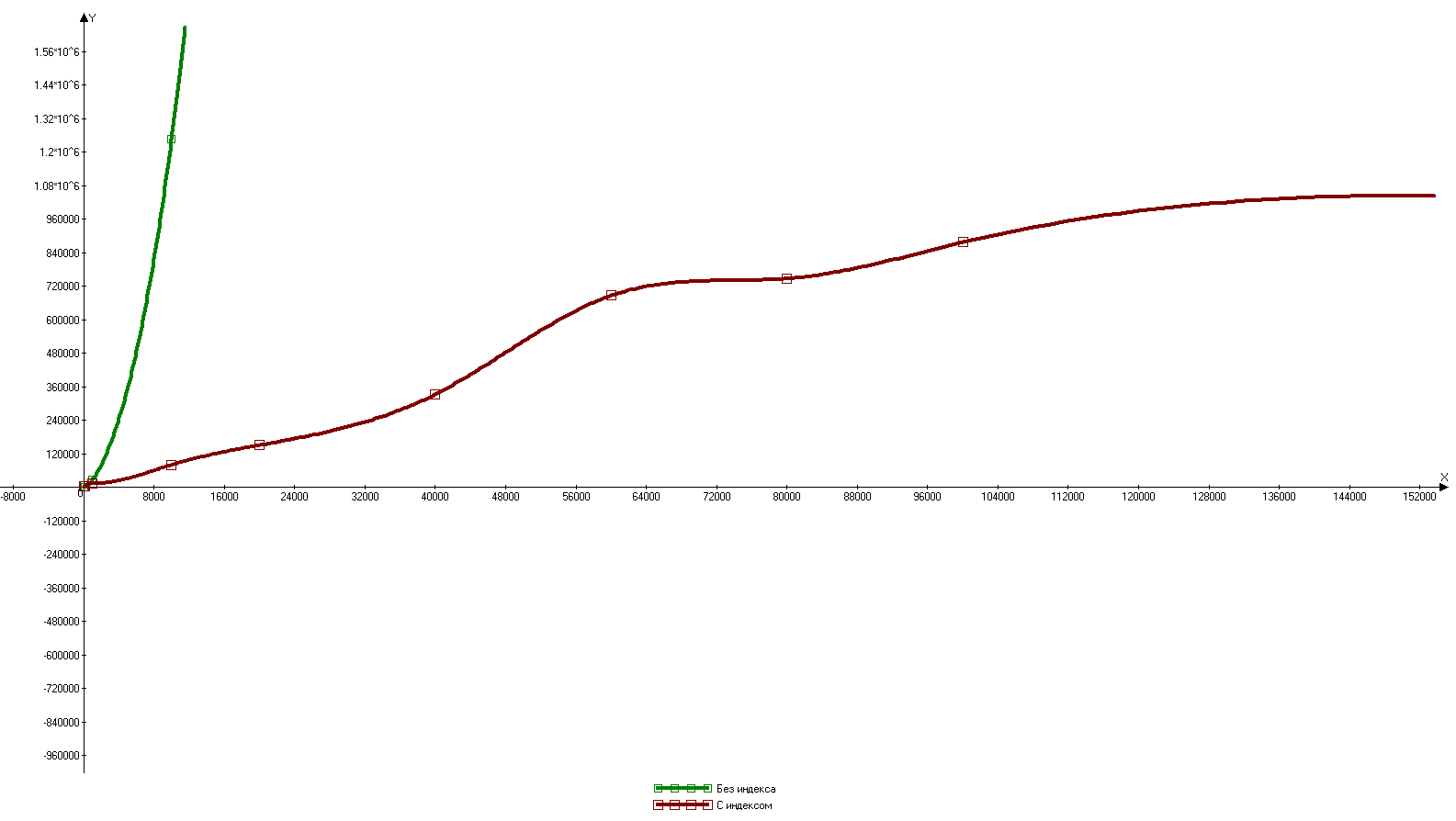


Рисунок 1 – График зависимости времени выборки от размера таблицы.

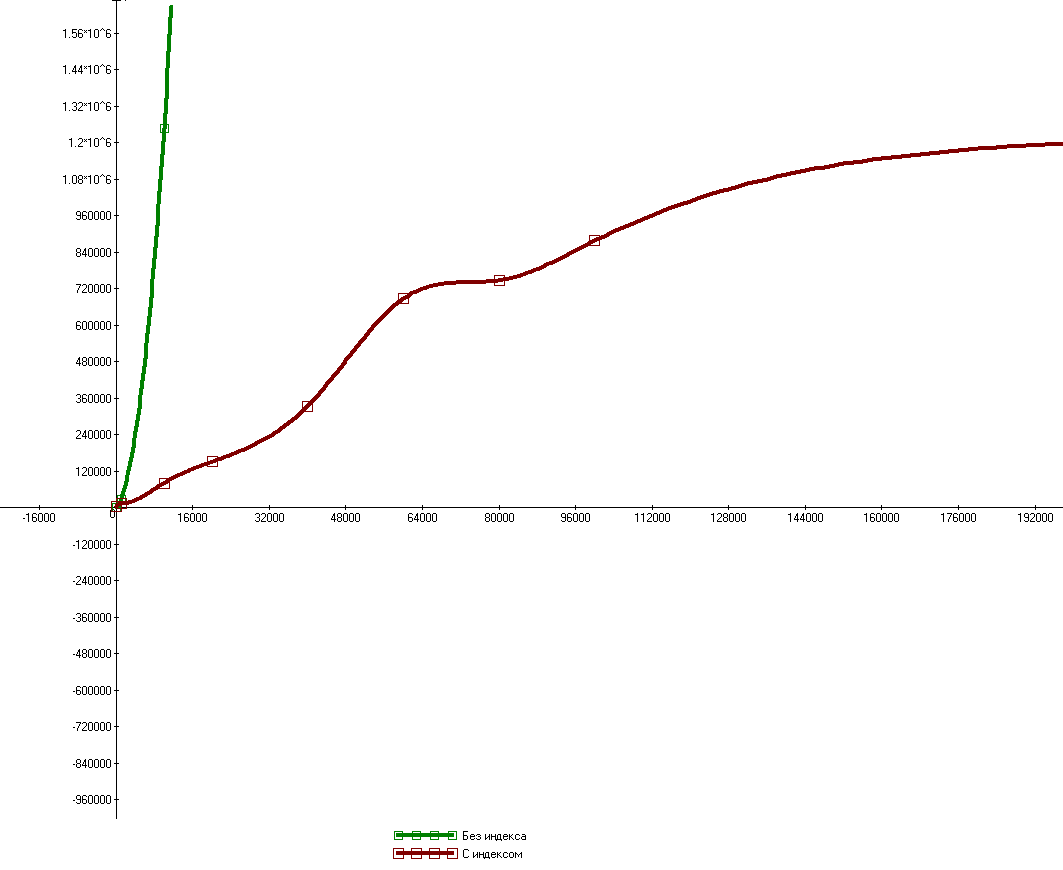


Рисунок 2 – Тот же график, но с увеличенным масштабом.

На рисунках по вертикали отмечено время в микросекундах, по горизонтали — размер таблицы в строках.

Красным показана зависимость для таблицы с индексом, зелёным — без индекса. Зависимость для таблицы **с индексом** визуально довольно близка к **логарифмической** по определению B-дерева. В то же время, для таблицы **без индекса** эта зависимость ближе к **экспоненциальной**.

Был проведён эксперимент по определению производительности вставки. Ниже приведён скрипт для замера. Очистка таблиц происходит удалением и созданием новой таблицы.

While @cnt < @top

Begin

select @\_F1 = F1, @\_F2 = F2 from arr where idx = @cnt

set @Time1 = SYSDATETIME()

Insert Into T1 values (@\_F1,@\_F2)

set @Time2 = SYSDATETIME()

set @timesum= @timesum+DATEDIFF(microsecond,@Time1,@Time2)

Set @cnt = @cnt + 1

End

Ниже приведены графики зависимости времени вставки от количества вставляемых записей

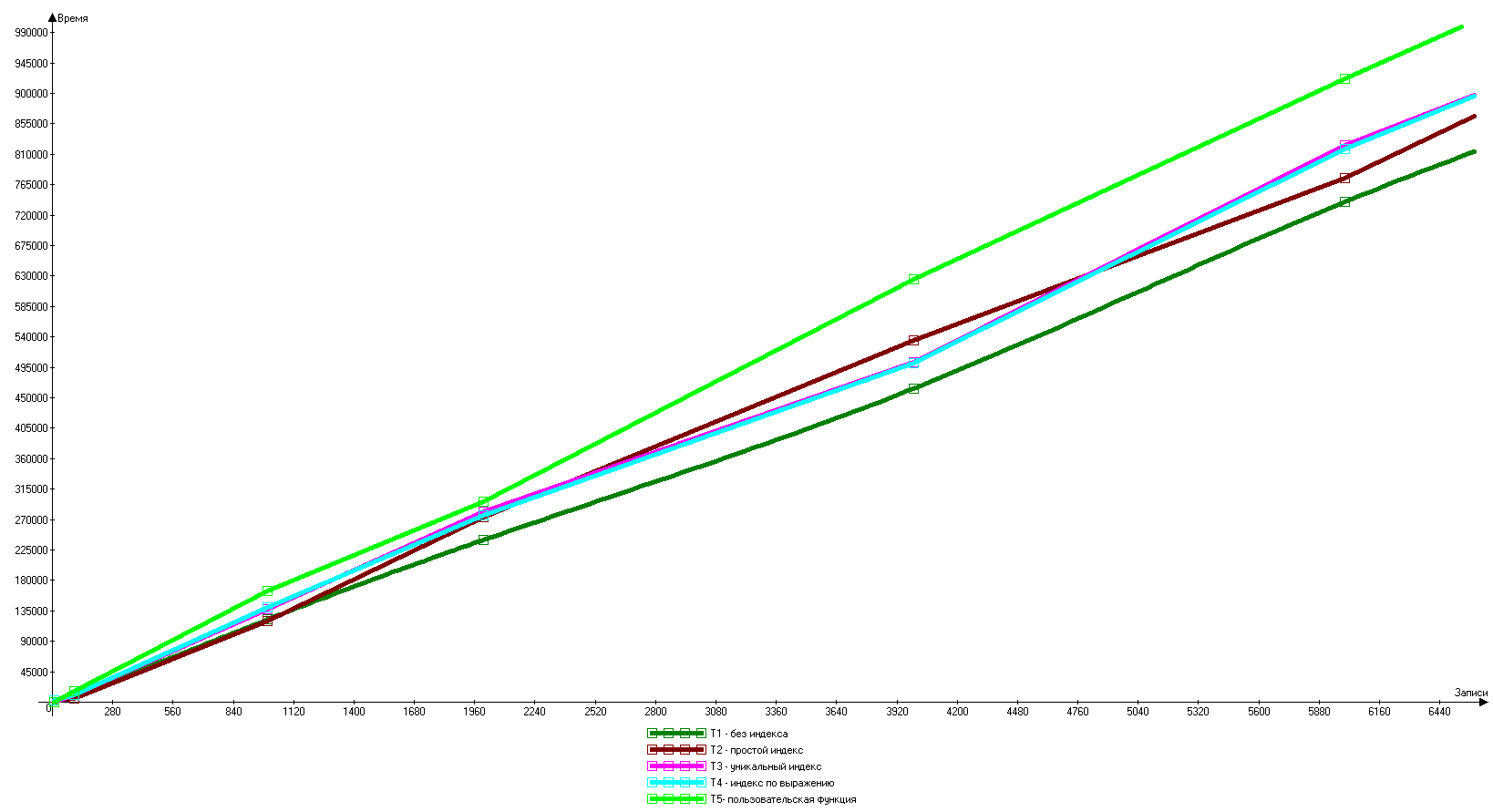


Рисунок 3 – График времени вставки от количества записей.

Все зависимости получились линейными, но с разными коэффициентами. Вставка в таблицу с индексом по функции занимает больше всего времени. Остальные таблицы показывают очень близки результаты, и только при больших размерах таблицы начинает проявляться видимая разница. Так, вставка в таблицу без индекса в итоге занимает меньше всего времени.

Был проведён эксперимент по определению производительности обновления данных. Ниже приведён скрипт для замера.

set @Time1 = SYSDATETIME()

update T1 set F1=F1+10

set @Time2 = SYSDATETIME()

print 'size ' + convert(varchar(20),@top)+ ' Time elapsed T1: '+convert(varchar(36),DATEDIFF(microsecond,@Time1,@Time2),14)

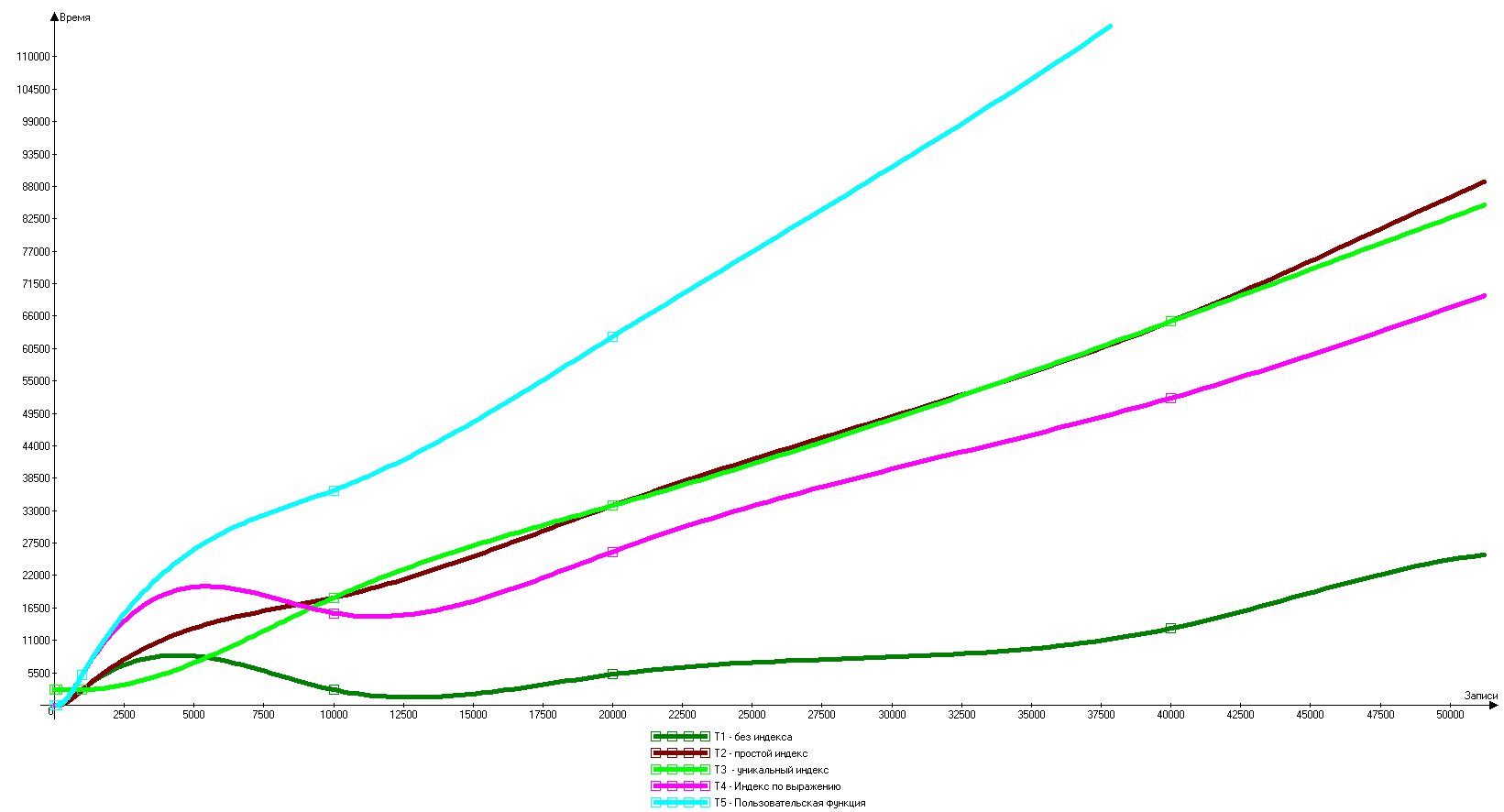


Рисунок 4 – График зависимости времени обновления от записей.

Полученные в результате эксперимента графики зависимостей ближе всего к линейным. Явно видно, что операция обновления быстрее всего выполняется в таблице без индекса, и медленнее всего в таблице с индексом по функции.

В лабораторной работе №1 в таблицах по заданию присутствуют первичные ключи. Первичный ключ представляет из себя уникальный индекс(кластер). Таким образом, для таблиц в первой лабораторной работе индексы уже существуют. Однако возможность оптимизации есть. Проанализировав реализованные запросы из лабораторной работы №2, видно, что самыми частыми полями для обращения являются:

1. Chelovek.FIO
2. Ocenka.ObuchayushchijsyaNumerPassporta

В то время как в таблице Ocenka уникальные индексы, следующие:

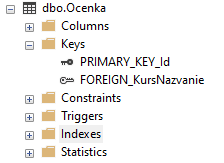


Рисунок 5 – Ключи в таблице Ocenka.

Аналогично в Chelovek

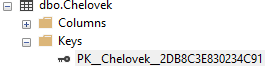


Рисунок 6 – Ключ в таблице Chelovek.

Тогда создадим индексы на колонки, к которым больше всего обращений в таблицах Chelovek и Ocenka командами:

CREATE INDEX ObuchNumPass on Ocenka (ObuchayushchijsyaNumerPassporta)

CREATE INDEX Chel\_idx ON Chelovek (FIO)

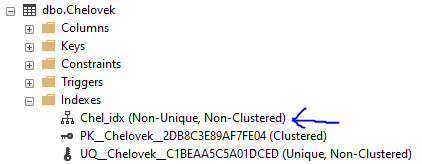


Рисунок 7 – Созданный индекс в таблице Chelovek.

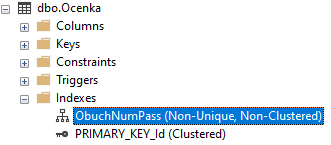


Рисунок 8 – Созданный индекс в таблице Ocenka.

**Контрольные вопросы**

1. Что такое индекс?

Индексы — это специальные структуры в базах данных, которые позволяют ускорить поиск и сортировку по определенному полю или набору полей в таблице, а также используются для обеспечения уникальности данных. Проще всего индексы сравнить с указателями в книгах. Если нет указателя, то нам придется просмотреть всю книгу, чтобы найти нужное место, а с указателем то же действие можно выполнить намного быстрее.

1. Какие структуры используются для представления индексов в СУБД

Используется B-дерево,В+-дерево, B\*-дерево. Последнее – комбинация двух первых.

1. Что такое B+ дерево?

B+ -дерево – модификация сбалансированного B-дерева.

В B+-дереве настоящие ключи хранятся лишь в листьях дерева, а во внутренних узлах хранятся лишь ключи-маршрутизаторы, необходимые для поиска по дереву.

Листья в B+-дереве содержат 𝑡 ≤ 𝑛 ≤ 2𝑡 ключей, где 𝑡 – порядок дерева, ограничения для внутренних узлов такие же, как и в B-дереве.

Деление листьев происходит поровну на две части, крайний ключ из левой половины делимого узла копируется в родительскую вершину в качестве ключа-маршрутизатора аналогично перемещению медианы для обычного деления, деление внутренних узлов происходит так же, как и в B-дереве. Сложность O(log n).

1. Чем отличается расширенное хэширование при построении индексов?

Хешированиемили называется технология быстрого прямого доступа к хранимой записи на основе заданного значения некоторого поля, при этом совсем не обязательно, чтобы поле было ключевым. Расширенное-хеширование - В основной памяти поддерживается справочник, организованный на основе бинарного дерева поиска, ключ к которому является значение хэш функции, а в листовых вершинах хранятся номера блоков записей внешней памяти.

В этом случае любой поиск в дереве цифрового поиска является «успешным», т.е. ведет к некоторому блоку внешней памяти. Входит ли в этот блок искомая запись, обнаруживается уже после прочтения блока в основную память.

Главное отличие между индексацией и хешированием является то, что Индексирование оптимизирует производительность базы данных, уменьшая количество обращений к диску для обработки запросов, а хеширование вычисляет прямое расположение записи данных на диске без использования структуры индекса.

1. В каких случаях требуется создание индексов?

В случае, если необходимо ускорить доступ к данным. Но это не является гарантированным правилом и, например, при небольшом размере таблицы, простой перебор записей может быть более эффективен чем выборка по индексам.

1. В каких случаях СУБД сама создает индексы?

SQL Server автоматически создаёт индексы по первичным ключам и по внешним ключам.

**Вывод**

Были изучены индексы в СУБД. Были рассмотрены основные типы и виды индексов, способы их задания. Было изучено влияние разных индексов на производительность СУБД в задачах выборки, вставки и обновления данных.

**Список литературы**

1. Базы данных. Лабораторный практикум. Полтавцева М.А.: учебное пособие / М.А. Полтавцева. – СПб.: Изд-во Политехн. Ун-та, 2016 г.
2. Конспект лекций по СУБД.

**Приложение 1**

Скрипты по создание таблиц и индексов:

create OR ALTER function idx\_func (@x int)

returns int

WITH SCHEMABINDING

as

begin

DECLARE @ret int;

if @x%1 = 0

set @ret = (1+@x)

else

set @ret = (1-@x)

return @ret

end

create table T1

(

F1 int,

F2 varchar(80)

)

create table T2

(

F1 int,

F2 varchar(80)

)

create index T2\_F1\_i on T2 (F1)

create table T3

(

F1 int unique,

F2 varchar(80)

)

create table T4

(

F1 int,

F2 varchar(80),

F3 as (abs(F1)+1)

)

create index T4\_F1\_i on T4(F3)

create table T5

(

F1 int,

F2 varchar(80),

F3 as (dbo.idx\_func(F1))

)

create index T5\_F1\_i on T5(F3)

**Приложение 2**

Скрипты по заполнению таблиц случайными значениями:

Declare @top int

Set @top = 1000

Declare @cnt int

Set @cnt = 0

While @cnt < @top

Begin

set @\_F1 = ABS(CHECKSUM(NEWID()))

set @\_F2 = convert(varchar(36),newid())

Insert Into arr values (@cnt,@\_F1,@\_F2)

Set @cnt = @cnt + 1

End