Лекция №7

Упорядочивание и поиск информации

Курс “Базы данных”

Лектор: Петруненко Д.А.. nokiad_1999@mail.ru

Хранение таблицы с добавлением в конец

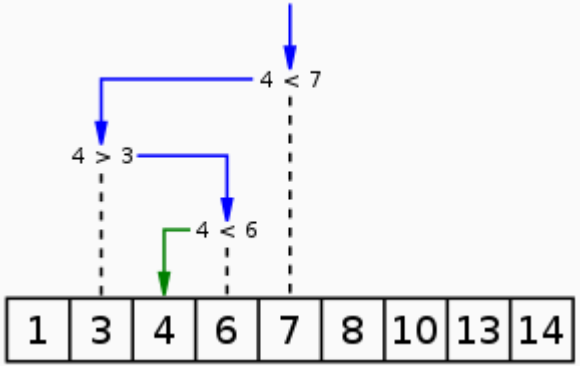
- Строки записываются в страницы по мере поступления в таблицу
- Добавление всегда происходит в конец - последнюю страницу таблицы

Поиск в таблице с добавлением в конец

- Поиск строки возможен путем сканирования таблицы
- Оценка N/2 в среднем, где N - общее количество страниц таблицы
- Нужны способы для более быстрого поиска

Поиск в отсортированном файле

- Строки таблицы упорядочены по значению ключа этого индекса
- Добавление информации в таблицу приводит к изменению физического порядка данных
- Поиск методом половинного деления: $\log_2 N$, - N количество блоков файла



Типы запросов

- точечный запрос
- набор из нескольких записей
- ранговый запрос
- выборка с использованием функций агрегирования

Индекс

Избыточная структура, предназначенная для ускорения поиска

Основное назначение:

- увеличение скорости доступа к данным;
- поддержка уникальности данных;
- автоматическое упорядочение записей при выборке.

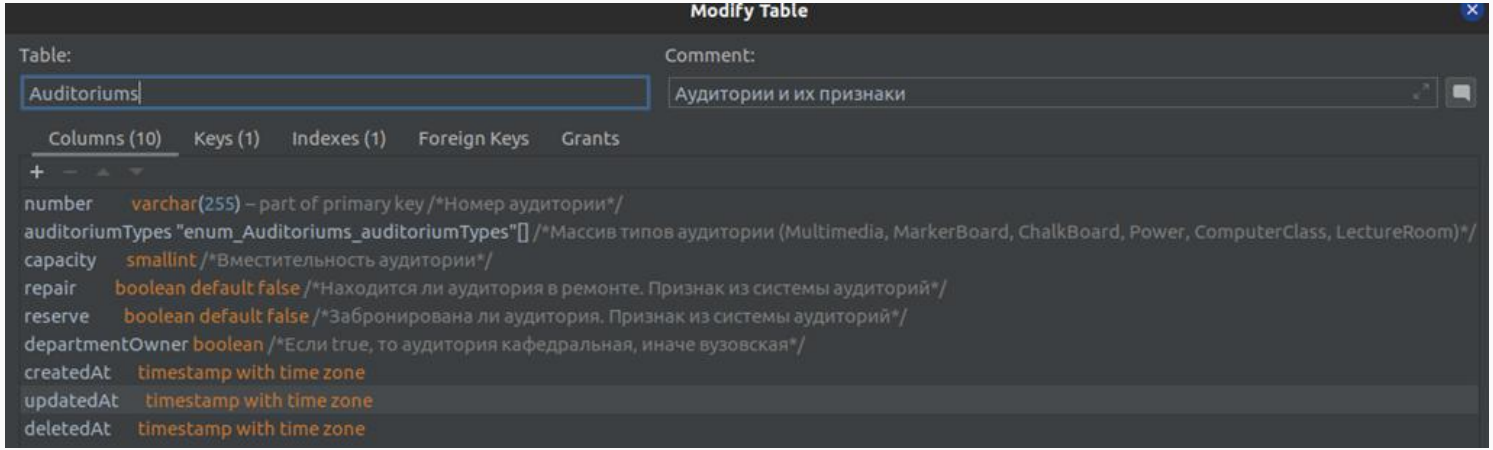
Поиск с помощью индекса

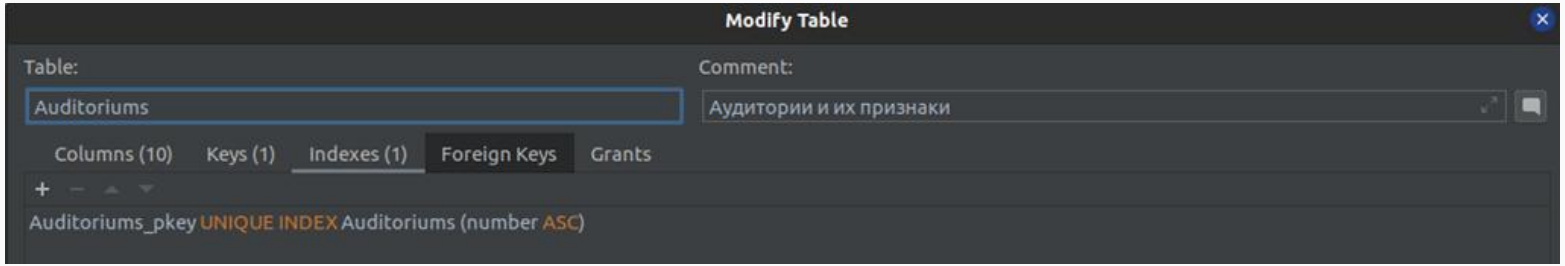
- Точное значение атрибута.
- Интервал значений атрибута.
- Значение нескольких атрибутов

Способы определения индекса

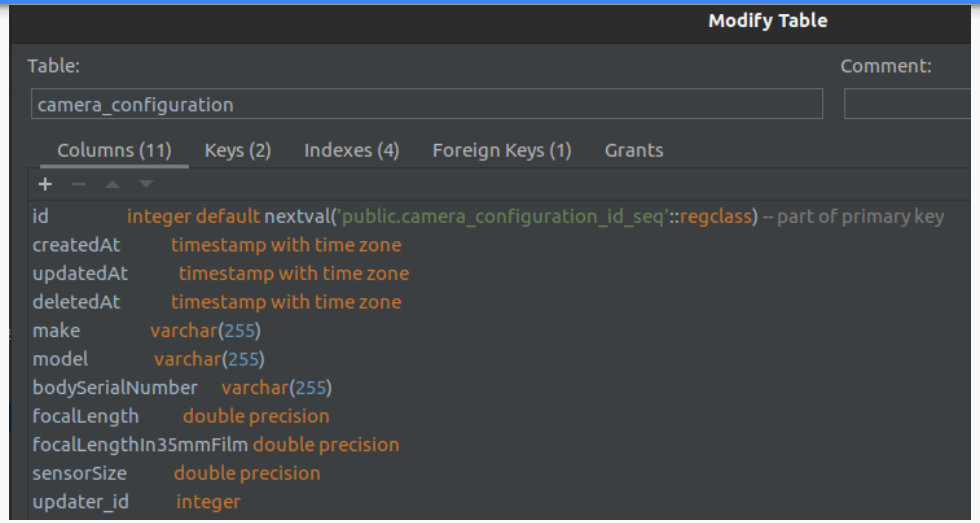
- Автоматическое создание индекса при создании первичного ключа **PRIMARY KEY**;
- Автоматическое создание индекса при определении ограничения целостности **UNIQUE**;
- Создание индекса с помощью команды **CREATE INDEX**

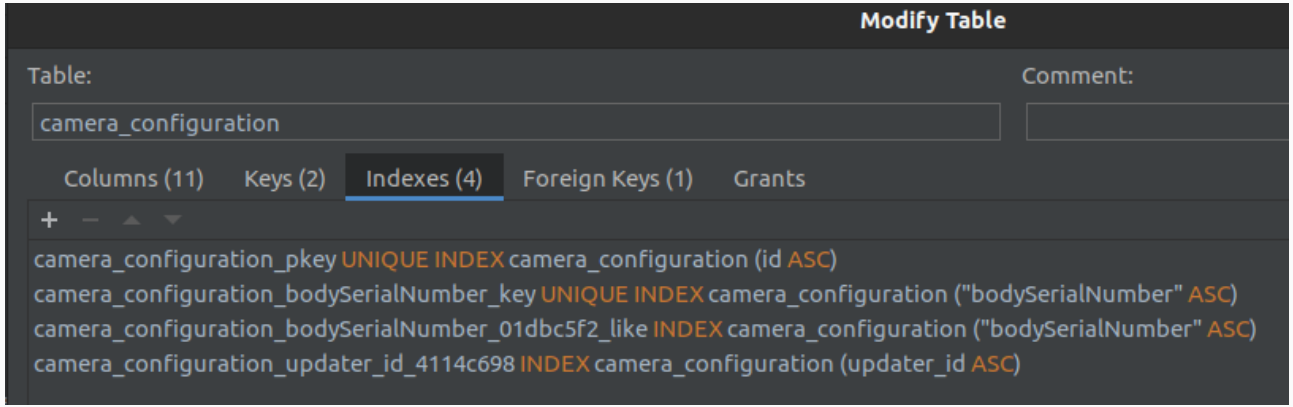
Datagrip


Modify Table
Table:
Auditoriums
Comment:
Аудитории и их признаки
Columns (10)
Keys (1)
Indexes (1)
Foreign Keys
Grants
number varchar(255) – part of primary key /*Номер аудитории*/
auditoriumTypes "enum_Auditoriums_auditoriumTypes"[] /*Массив типов аудитории (Multimedia, MarkerBoard, ChalkBoard, Power, ComputerClass, LectureRoom)*/
capacity smallint /*Вместительность аудитории*/
repair boolean default false /*Находится ли аудитория в ремонте. Признак из системы аудиторий*/
reserve boolean default false /*Забронирована ли аудитория. Признак из системы аудиторий*/
departmentOwner boolean /*Если true, то аудитория кафедральная, иначе вузовская*/
createdAt timestamp with time zone
updatedAt timestamp with time zone
deletedAt timestamp with time zone


Modify Table
Table:
Auditoriums
Comment:
Аудитории и их признаки
Columns (10)
Keys (1)
Indexes (1)
Foreign Keys
Grants
Auditoriums_pkey UNIQUE INDEX Auditoriums (number ASC)

Datagrip


Modify Table
Table:
Comment:
camera_configuration
Columns (11)
Keys (2)
Indexes (4)
Foreign Keys (1)
Grants
id integer default nextval('public.camera_configuration_id_seq'::regclass) -- part of primary key
createdAt timestamp with time zone
updatedAt timestamp with time zone
deletedAt timestamp with time zone
make varchar(255)
model varchar(255)
bodySerialNumber varchar(255)
focalLength double precision
focalLengthIn35mmFilm double precision
sensorSize double precision
updater_id integer


Modify Table
Table:
Comment:
camera_configuration
Columns (11)
Keys (2)
Indexes (4)
Foreign Keys (1)
Grants
camera_configuration_pkey UNIQUE INDEX camera_configuration (id ASC)
camera_configuration_bodySerialNumber_key UNIQUE INDEX camera_configuration ("bodySerialNumber" ASC)
camera_configuration_bodySerialNumber_01dbc5f2_like INDEX camera_configuration ("bodySerialNumber" ASC)
camera_configuration_updater_id_4114c698 INDEX camera_configuration (updater_id ASC)

Создание индекса

```
CREATE INDEX index_name ON table_name [USING method]
(
    column_name [ASC | DESC] [NULLS {FIRST | LAST }],
    ...
);
```

https://www.postgresqltutorial.com/postgresql-administration/postgresql-create-schema/

Характеристики индекса:

- Методы индексирования - btree, hash, gist, spgist, gin. В PostgreSQL по стандарту используется - btree
- индексируемая длина столбца, порядок (возрастание/убывание) - ASC и DESC. По умолчанию – ASC. Если столбец содержит NULL, вы можете указать параметр NULLS FIRST или NULLS LAST для сортировки. Значение NULLS FIRST используется по умолчанию, когда указан DESC, а значение NULLS LAST - по умолчанию, когда DESC не указан.
- с одним или несколькими столбцами;

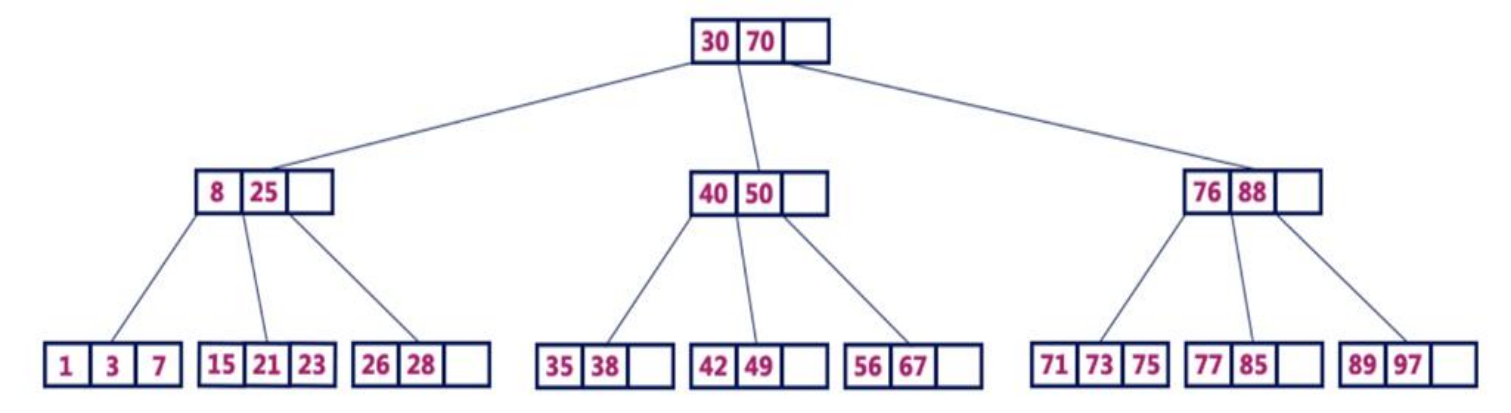
Виды индексов

- **B-деревья;**
- **Hash-индексы;**
- **индексы на основе битовых карт;**
- **пространственные индексы;**
- **многомерные индексы;**
- **полнотекстовые**

https://habr.com/ru/post/102785/

B-дерево

- Имеет внутренние (индексные) и листовые вершины.
- Листовые вершины находятся на самом нижнем уровне дерева.



Свойства B-дерева

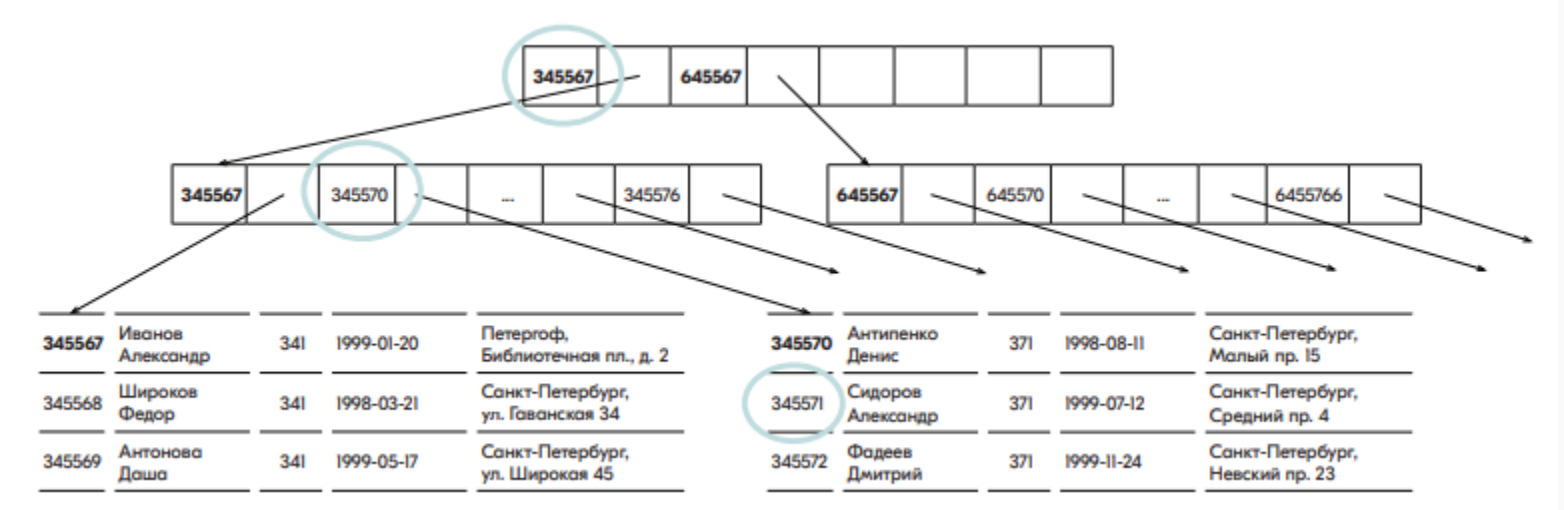
- B-дерево – сбалансированная структура, т.е. от корня до любой листовой страницы одинаковое количество узлов.
- Высота B-дерева – $\log_m N$, где N - количество блоков данных, а m - количество пар в индексном блоке.

https://habr.com/ru/companies/otus/articles/459216/

Поиск в B-дереве

- Начинается с корня дерева.
- На каждом уровне в блоке находится значение, покрывающее искомый ключ. Переход на следующий уровень.
- Считывается блок данных на листовом уровне

Пример: поиск в B-дереве StudentId=345571



345567	Иванов Александр	341	1999-01-20	Петергоф, Библиотечная пл., д. 2
345568	Широков Федор	341	1998-03-21	Санкт-Петербург, ул. Гаванская 34
345569	Антонова Даша	341	1999-05-17	Санкт-Петербург, ул. Широкая 45

345570	Антипенко Денис	371	1998-08-11	Санкт-Петербург, Малый пр. 15
345571	Сидоров Александр	371	1999-07-12	Санкт-Петербург, Средний пр. 4
345572	Фадеев Дмитрий	371	1999-11-24	Санкт-Петербург, Невский пр. 23

Вставка в B-дерево

- Производим поиск по значению вставляемого ключа.
- Если в блоке есть место, то добавляем туда новую запись. Иначе создаем новый блок, а записи старого распределяем поровну в два блока.
- Так же поступаем со всеми уровнями

Кластерный индекс

- Строится на основе B-дерева.
- Должно быть не больше 1.
- Обеспечивает самый быстрый поиск по заданному ключу.
- PostgreSQL по умолчанию не назначает индексы в качестве индексов кластеризации в отличии от MySql.

Кластерный индекс создание

```
CREATE TABLE test.cluster_table

(id      INTEGER, name VARCHAR) WITH (FILLFACTOR = 90);

CREATE INDEX id_idx ON test.cluster_table (id);

CLUSTER [VERBOSE] test.cluster_table USING id_idx;
```

Рекомендации для ключа кластерного индекса

1. Уникальный.
2. Узкий.
3. Статичный

Фактор кластеризации

Коэффициент кластеризации является мерой того, насколько близко порядок данных в таблице соответствует порядку данных в индексе. Высокий коэффициент кластеризации означает, что данные в таблице хорошо упорядочены в соответствии с индексом, что может повысить производительность запросов.

Некластерные индексы

- Создаются на основе кластерного индекса, если он есть.
- Строятся автоматически при объявлении столбца (-ов) UNIQUE.
- Могут быть созданы командой CREATE INDEX.

Индекс по составному ключу

- Индекс может быть создан на основании нескольких полей.
- Порядок полей важен.

Пример: создание составного индекса

```
CREATE INDEX index_name ON table_name(a,b,c,...);
```

Индексы: недостатки

- Занимают дополнительное место на диске и в оперативной памяти.
- Чем больше/длиннее ключ, тем больше размер индекса.
- Замедляются операции вставки, обновления и удаления записей.

Рекомендации

- Для интенсивно обновляемых таблиц – ??? .
- Для таблиц с редкими обновлениями, но большими объемами данных - ???
- Индексы для маленьких таблиц - ???

Рекомендации

- Для интенсивно обновляемых таблиц – меньше индексов.
- Для таблиц с редкими обновлениями, но большими объемами данных – больше индексов.
- Индексы для маленьких таблиц лучше не строить.

Полнотекстовый поиск

- Для хранения подготовленных документов в PostgreSQL предназначен тип данных tsvector;
- Для представления обработанных запросов — тип tsquery;

Полнотекстовый поиск с использованием индексов

Для полнотекстового поиска PostgreSQL предлагает два индекса на выбор:

- GIN — быстро ищет, но не слишком быстро обновляется. Отлично работает, если сравнительно редко меняются данные, по которым происходит поиск;
- GiST — ищет медленнее GIN, зато очень быстро обновляется. Может лучше подходить для поиска по очень часто обновляемым данным;

Процесс полнотекстового индексирования

- Фильтрация, разбиение по словам.
- Удаление стоп-слов и нормализация.
- Заполнение полнотекстового индекса

Создание таблицы с полнотекстовым индексом

```
CREATE TABLE IF NOT EXISTS
  articles(id SERIAL PRIMARY KEY, title VARCHAR(128), content TEXT);

CREATE INDEX IF NOT EXISTS idx_fts_articles ON articles
  USING gin(to_tsvector(title, content));
```

Пример: поиск с помощью полнотекстового индекса

SELECT id, title FROM articles WHERE to_tsvector (title, content) @@ to_tsquery('bjarne <-> stroustrup');

```
  id   |                           title
-------+-----------------------------------------------------------
  2470 | Binary search algorithm
  2129 | Bell Labs
  2130 | Bjarne Stroustrup
  3665 | C (programming language)
...
(17 rows)

Time: 136.357 ms
```

Индексы на основе Hash

- Подбирается функция перемешивания - hash function
- Функция хэширования для ключа выдает номер участка
- В памяти хранится таблица адресов участков
- Доступ к данным возможен за одно обращение к диску

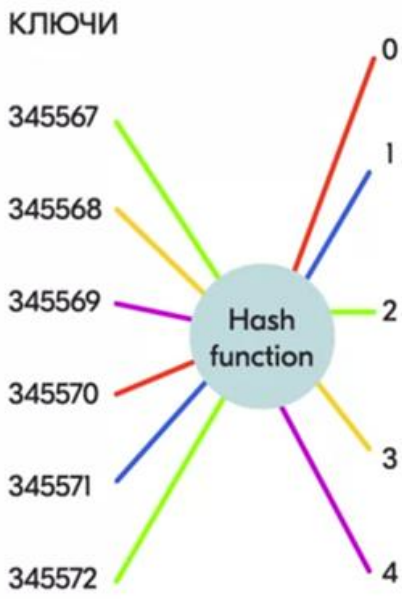
Создание

```
CREATE INDEX index_name ON table_name USING HASH
(indexed_column);
```

Пример

StudentId	StudentName	GroupNumber	BirthDate	Address
345567	Иванов Александр	341	1999-01-20	Петергоф, Библиотечная пл., дом 2
345568	Широков Федор	341	1998-03-21	Санкт-Петербург, ул. Гаванская 34
345569	Антонова Даша	341	1999-05-17	Санкт-Петербург, ул. Широкая 45
345570	Антипенко Денис	371	1998-08-11	Санкт-Петербург, Малый пр. 15
345571	Сидоров Александр	371	1999-07-12	Санкт-Петербург, Средний пр. 4
345572	Фадеев Дмитрий	371	1999-11-24	Санкт-Петербург, Невский пр. 23

Пример



345570	Антипенко Денис	371	1998-08-11	Санкт-Петербург, Малый пр. 15
345571	Сидоров Александр	371	1999-07-12	Санкт-Петербург, Средний пр. 4
345567	Иванов Александр	341	1999-01-20	Петергоф, Библиотечная пл., дом 2
345572	Фадеев Дмитрий	371	1999-11-24	Санкт-Петербург, Невский пр. 23
345568	Широков Федор	341	1998-03-21	Санкт-Петербург, ул. Гаванская 34
345569	Антонова Даша	341	1999-05-17	Санкт-Петербург, ул. Широкая 45

Недостатки hash-индексов

- Таблица адресов участков может не помещаться в оперативную память
- Неравномерность размещения записей, возникновение коллизий
- Хэш-индексы могут обрабатывать только простое сравнение на равенство (=)