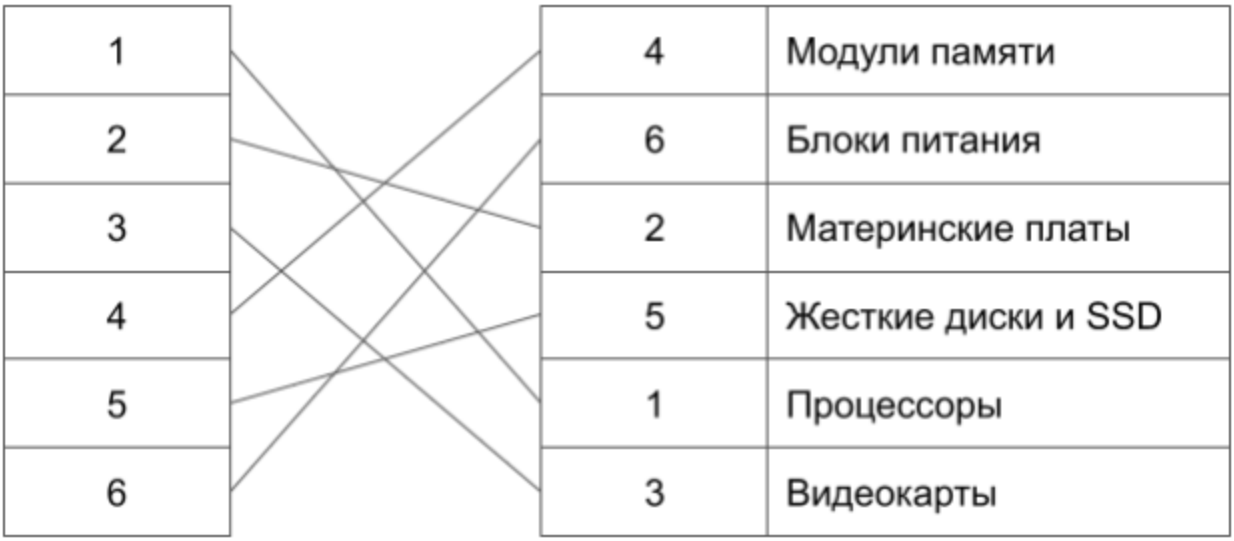
Индексы и ключи

Доброго времени суток, уважаемые студенты! Небольшая дополнительная лекция по индексам и ключам 😊

Небольшое вступление

Ключи нам с вами хорошо знакомы: это столбцы, при помощи которых мы добиваемся уникальности записей и связываем записи в разных таблицах. Ключи очень часто снабжаются индексами, поэтому в массовом сознании разработчиков баз данных они часто сливаются в одно понятие. Тем не менее, индексировать можно не только столбцы с ключами, но и любой столбец или комбинацию столбцов таблицы.
Обычно записи в таблице располагаются в хаотическом порядке. Чтобы найти нужную запись, необходимо сканировать всю таблицу, на что уходит много времени. Идея индексов состоит в том, чтобы создать копию столбца, которая постоянно будет поддерживаться в отсортированном состоянии.



Это позволяет очень быстро осуществлять поиск по такому столбцу, т. к. заранее известно, где
необходимо искать значение. Обратная сторона медали — добавление или удаление записи требует дополнительного времени на сортировку столбца. Кроме того, создание копии увеличивает объем памяти, необходимый для размещения таблицы на жестком диске и в оперативной памяти сервера. Именно поэтому СУБД не создает индексы для каждого столбца и их комбинаций, а отдает это на откуп разработчикам.

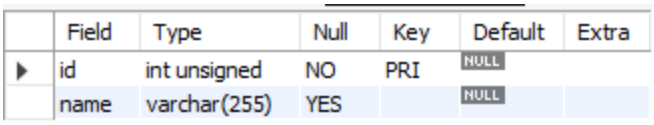
Виды индексов

Существует несколько видов индексов:
● обычный индекс — таких индексов в таблице может быть несколько;
● уникальный индекс — уникальных индексов также может быть несколько, но значения в нем не должны повторяться;
● первичный ключ — уникальный индекс, предназначенный для первичного ключа таблицы. В таблице может быть только один первичный ключ;
● полнотекстовый индекс — специальный вид индекса для столбцов типа TEXT, позволяющий
производить полнотекстовый поиск. Рассматривать его не будем, так как на практике задача
полнотекстового поиска осуществляется специализированными базами данных,

такими как
ElasticSearch.
Первичный ключ в таблице помечается специальным индексом PRIMARY KEY. Значение первичного ключа должно быть уникально и не повторяться в пределах таблицы. Кроме того, столбцы, помеченные атрибутом PRIMARY KEY, не могут принимать значение NULL. Для пометки поля таблицы в качестве первичного ключа достаточно поместить ключевое слово PRIMARY KEY в определение столбца.

Давайте пометим поле **id** таблицы **catalogs** атрибутом **PRIMARY KEY**:

```
DROP TABLE IF EXISTS catalogs;
CREATE TABLE catalogs (
  id INT UNSIGNED NOT NULL PRIMARY KEY,
  name VARCHAR(255) COMMENT 'Название раздела'
) COMMENT = 'Разделы интернет-магазина';
DESCRIBE catalogs;
```



	Field	Type	Null	Key	Default	Extra
▶	id	int unsigned	NO	PRI	NULL	
	name	varchar(255)	YES		NULL	

Есть альтернативный способ объявления первичного ключа — отдельной записью **PRIMARY KEY** с указанием названия столбца в круглых скобках:

```
DROP TABLE IF EXISTS catalogs;
CREATE TABLE catalogs (
  id INT UNSIGNED NOT NULL,
  name VARCHAR(255) COMMENT 'Название раздела',
  PRIMARY KEY(id)
) COMMENT = 'Разделы интернет-магазина';
```

Ключевое слово **PRIMARY KEY** может встречаться в таблице только один раз, так как в таблице разрешен только один первичный ключ. Индекс необязательно должен быть объявлен по одному столбцу, вполне допустимо объявление индекса сразу по двум или более столбцам.

```
DROP TABLE IF EXISTS catalogs;
CREATE TABLE catalogs (
```

```
id INT UNSIGNED NOT NULL,
  name VARCHAR(255) COMMENT 'Название раздела',
  PRIMARY KEY(id, name(10))
) COMMENT = 'Разделы интернет магазина';
DESCRIBE catalogs;
```

	Field	Type	Null	Key	Default	Extra
▶	id	int unsigned	NO	PRI	NULL	
	name	varchar(255)	NO	PRI	NULL	

Здесь первичный ключ создается по столбцу **id** и по первым 10 символам столбца name. Можно индексировать по всем 255 символам текстового столбца. Однако размер индекса будет больше — как правило, для индекса достаточно первых символов строки. В качестве первичного ключа часто выступает целочисленный столбец. Такой выбор связан с тем, что целочисленные типы данных обрабатываются быстрее всех и занимают небольшой объем. Другая причина выбора такого типа — только данный тип столбца может быть снабжен атрибутом **AUTO_INCREMENT**, который обеспечивает автоматическое создание уникального индекса. Передача столбцу, снабженному этим атрибутом, значения **NULL** или 0, приводит к автоматическому присвоению ему максимального значения столбца, плюс 1.

```
DROP TABLE IF EXISTS catalogs;
CREATE TABLE catalogs (
  id INT UNSIGNED NOT NULL PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255) COMMENT 'Название раздела'
) COMMENT = 'Разделы интернет магазина';

INSERT INTO catalogs (name) VALUES ('Процессоры');
INSERT INTO catalogs VALUES (0, 'Мат.платы');
INSERT INTO catalogs VALUES (NULL, 'Видекарты');
SELECT * FROM catalogs;
```

	id	name
▶	1	Процессоры
	2	Мат.платы
	3	Видекарты
*	NULL	NULL

Это удобный механизм, который позволяет не заботиться о генерации уникального значения
средствами прикладной программы, работающей с СУБД MySQL. MySQL предлагает псевдотип SERIAL, который является обозначением для типа BIGINT, снабженного
дополнительными атрибутами UNSIGNED, NOT NULL, AUTO_INCREMENT и уникальным индексом.

SERIAL == BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE

```
DROP TABLE IF EXISTS catalogs;
CREATE TABLE catalogs (
  id SERIAL PRIMARY KEY,
  name VARCHAR(255) COMMENT 'Название раздела'
) COMMENT = 'Разделы интернет-магазина';
```

Так гораздо компактнее, а главное — более совместимо с другими базами данных, которые зачастую тоже поддерживают псевдотип **SERIAL**. В отличие от первичного ключа, таблица может содержать несколько обычных и уникальных индексов. Чтобы было удобно их различать, индексы могут иметь собственные имена. Часто имена индексов совпадают с именами столбцов, которые они индексируют, но для индекса можно назначить и совершенно другое имя. Объявление индекса производится при помощи ключевого слова **INDEX** или **KEY**.

Для уникальных индексов вводится дополнительное ключевое слово **UNIQUE**. Давайте в таблице **products** снабдим индексом поле **catalog_id**.

```
DROP TABLE IF EXISTS products;
CREATE TABLE products (
  id SERIAL PRIMARY KEY,
  name VARCHAR(255) COMMENT 'Название',
  desription TEXT COMMENT 'Описание',
  price DECIMAL (11,2) COMMENT 'Цена',
  catalog_id INT UNSIGNED,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  KEY index_of_catalog_id(catalog_id)
) COMMENT = 'Товарные позиции';
DESCRIBE products;
```

Field	Type	Null	Key	Default	Extra
id	bigint unsigned	NO	PRI	NULL	auto_increment
name	varchar(255)	YES		NULL	
desription	text	YES		NULL	
price	decimal(11,2)	YES		NULL	
catalog_id	int unsigned	YES	MUL	NULL	
created_at	datetime	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED
updated_at	datetime	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED on update CURRENT_TI...

Создать индекс в уже существующей таблице можно при помощи оператора **CREATE INDEX**.

```
DROP TABLE IF EXISTS products;
CREATE TABLE products (
  id SERIAL PRIMARY KEY,
  name VARCHAR(255) COMMENT 'Название',
  desription TEXT COMMENT 'Описание',
  price DECIMAL (11,2) COMMENT 'Цена',
  catalog_id INT UNSIGNED,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
  ) COMMENT = 'Товарные позиции';
CREATE INDEX index_of_catalog_id ON products (catalog_id);
```

Удалить индекс из таблицы можно при помощи оператора **DROP INDEX**:

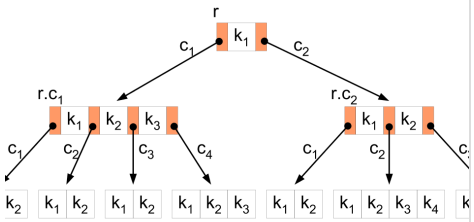
```
DROP INDEX index_of_catalog_id ON products;
```

Устройство индекса

B-дерево

B-дерево — структура данных, дерево поиска. С точки зрения внешнего логического представления, сбалансированное, сильно ветвистое дерево во внешней памяти.

W https://ru.wikipedia.org/wiki/B-дерево

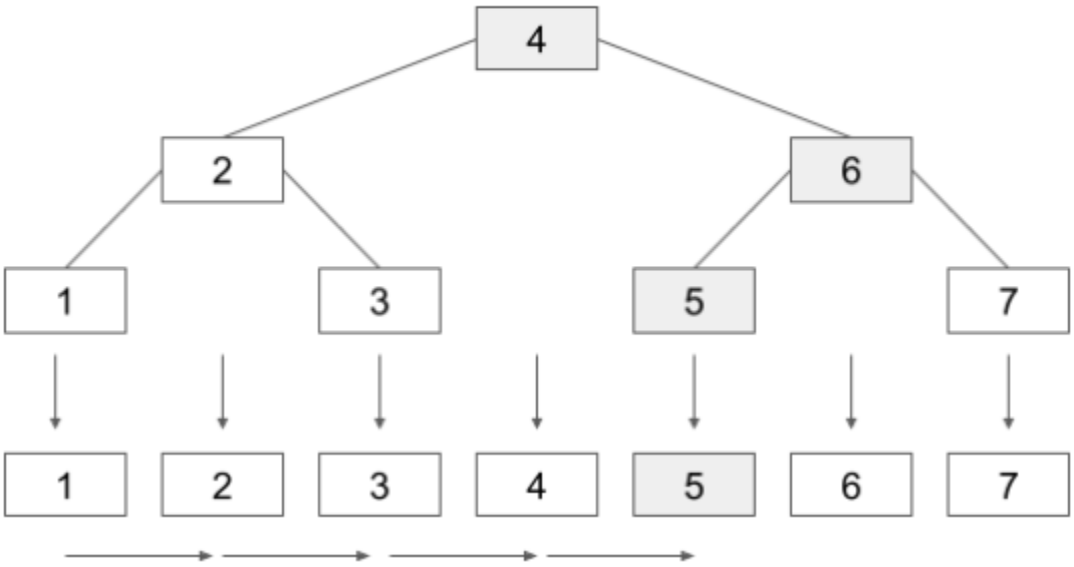


MySQL поддерживает два типа индекса:

- BTREE — бинарное дерево;

● HASH — хэш-таблица.

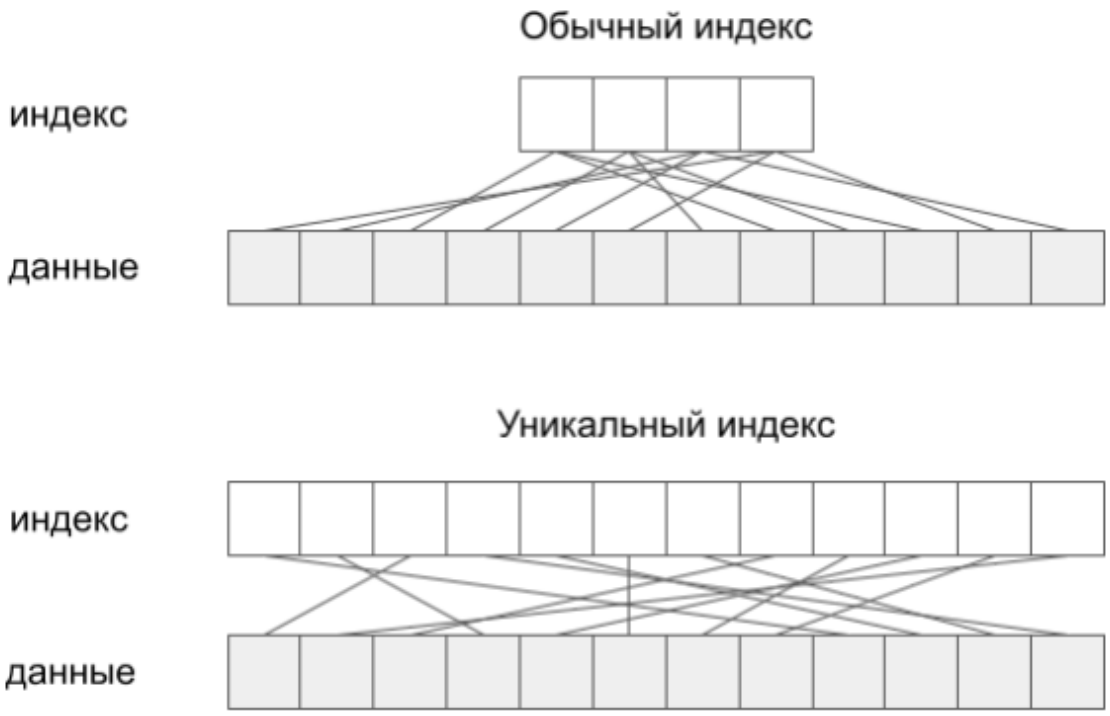
Общая идея бинарного дерева в том, что значения хранятся по порядку, и все листовые страницы находятся на одинаковом расстоянии от корня. **BTREE**-индекс ускоряет доступ к данным, поскольку подсистеме хранения не нужно сканировать всю таблицу для поиска нужной информации. В **BTREE**-индексах индексированные столбцы хранятся в упорядоченном виде, и они полезны для поиска по диапазону данных.



Поскольку узлы дерева отсортированы, их можно использовать как для поиска значений, так и в запросах с сортировкой при помощи **ORDER BY.**

```
CREATE INDEX index_of_catalog_id USING BTREE ON products (catalog_id);
CREATE INDEX index_of_catalog_id USING HASH ON products (catalog_id);
```

Хэш-индекс строится на основе хэш-таблицы и полезен только для точного поиска с указанием всех столбцов индекса. Для каждой строки подсистема хранения вычисляет хэш-код индексированных столбцов. В индексе хранятся хэш-коды и указатели на соответствующие строки.



Селективность индекса — это отношение количества проиндексированных значений к общему количеству строк в таблице. Индекс с высокой селективностью хорош тем, что позволяет MySQL при поиске соответствий отфильтровывать больше строк. Уникальный индекс имеет селективность, равную единице. Если селективность индекса низкая, после локализации участка или набора элементов, где находится искомое значение, его потребуется дополнительно просканировать для точного поиска значения. В случае уникального индекса мы можем сразу получить искомый результат, без сканирования даже небольшого участка таблицы. Давайте поправим таблицы учебной базы данных:

```
DROP TABLE IF EXISTS products;
CREATE TABLE products (
  id SERIAL PRIMARY KEY,
  name VARCHAR(255) COMMENT 'Название',
  desription TEXT COMMENT 'Описание',
  price DECIMAL (11,2) COMMENT 'Цена',
  catalog_id INT UNSIGNED,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  KEY index_of_catalog_id(catalog_id)
) COMMENT = 'Товарные позиции';
```

```
DROP TABLE IF EXISTS orders;
CREATE TABLE orders (
  id SERIAL PRIMARY KEY,
  user_id INT UNSIGNED,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  KEY index_of_user_id(user_id)
) COMMENT = 'Заказы';
```

В таблице **orders_products** у нас два кандидата на индексацию: **order_id** и **product_id**. Порядок
следования столбцов в индексе имеет значение

year	last_name	first_name
1990	Абакумов	Сергей
1990	Борисов	Игорь
1990	Сергеев	Вячеслав
1991	Антонов	Александр
1991	Ковалев	Сергей
1991	Трофимов	Антон

SELECT * FROM tbl
WHERE year = 1990

SELECT * FROM tbl
WHERE
  year = 1990 AND
  last_name = Борисов

SELECT * FROM tbl
WHERE first_name = ‘Сергей’

В каждом запросе может использоваться ровно один индекс, если в запросе участвует два поля — **order_id** и **product_id** и индекс по двум столбцам сработает. Если у нас два отдельных запроса с участием **order_id** и **product_id**, то индекс **order_id** и **product_id** может использоваться в запросе с **order_id**, но его не получится использовать в запросе с **product_id**. В **discount**, с высокой долей вероятности, **user_id** и **product_id** будут использоваться раздельно. Давайте снабдим их индексами.

```
DROP TABLE IF EXISTS discounts;
CREATE TABLE discounts (
  id SERIAL PRIMARY KEY,
  user_id INT UNSIGNED,
  product_id INT UNSIGNED,
  discount FLOAT UNSIGNED COMMENT 'Величина скидки от 0.0 до 1.0',
```

```
  finished_at DATETIME NULL,
  started_at DATETIME NULL,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  KEY index_of_user_id(user_id),
  KEY index_of_product_id(product_id)
) COMMENT = 'Скидки';
```

Используемые источники

1. https://dev.mysql.com/doc/refman/5.7/en/tutorial.html
2. https://dev.mysql.com/doc/refman/5.7/en/literals.html
3. Линн Бейли. Head First. Изучаем SQL. — СПб.: Питер, 2012. — 592 с.
4. Грофф, Джеймс Р., Вайнберг, Пол Н., Оппель, Эндрю Дж. SQL: полное руководство, 3-е изд. :
   Пер. с англ. — М.: ООО "И.Д. Вильямс", 2015. — 960 с.
5. Дейт К. Дж. SQL и реляционная теория. Как грамотно писать код на SQL. — Пер. с англ. —
   СПб.: Символ-Плюс, 2010. — 480 с.
6. Кузнецов М.В., Симдянов И.В. MySQL на примерах. — СПб.: БХВ-Петербург, 2007. — 592с.
7. Кузнецов М.В., Симдянов И.В. MySQL 5. — СПб.: БХВ-Петербург, 2006. — 1024с.
8. Дейт К. Дж. Введение в системы баз данных, 8-е издание.: Пер. с англ. — М.: Издательский
   дом "Вильямс", 2005. — 1328 с.
9. Карвин Б. Программирование баз данных SQL. Типичные ошибки и их устранение. — Рид
   Групп, 2011. — 336 с.

Книги:

- “Изучаем SQL”, книга Бейли Л.
- Алан Бьюли "Изучаем SQL" (2007)
- Энтони Молинаро "SQL. Сборник рецептов" (2009)

1. Виктор Гольцман “MySQL 5.0. Библиотека программиста”
2. “Изучаем SQL”, книга Бейли Л.
3. https://www.webmasterwiki.ru/MySQL
4. Поль Дюбуа “MySQL. Сборник рецептов”