Севастопольский государственный университет
Кафедра «Информационные системы»

Управление данными

курс лекций

лектор:
ст. преподаватель кафедры ИС Абрамович А.Ю.

Лекция 13

Язык SQL.

Ограничения. Индексация

ОГРАНИЧЕНИЯ В ТАБЛИЦАХ

Ограничения – это правила, применяемые к столбцам данных в таблице. Они **используются для предотвращения ввода неверных данных в базу данных, что обеспечивает точность и достоверность данных.**

Ограничение (constraints) – это ограничение типа значений, которое накладывается на один или несколько столбцов таблицы. **Это позволяет поддерживать точность и целостность данных в таблице БД.**

В SQL существует несколько различных типов ограничений:

CHECK	**NOT NULL**	**DEFAULT**	**UNIQUE**	**PRIMARY KEY**	**FOREIGN KEY**

Ограничение CHECK

Ограничение-проверка — наиболее общий тип ограничений. В его определении можно указать, что значение данного столбца должно удовлетворять логическому выражению (проверке истинности).

Ограничение CHECK используется для ограничения значений, которые могут быть помещены в столбец.

```
CREATE TABLE employees (
    emp_id INTENTEGER,
    emp_name VARCHAR(55) NOT NULL,
    hire_date DATE NOT NULL,
    salary INTEGER NOT NULL CHECK (salary >= 3000 AND salary <= 10000),
    dept_id INTEGER,
);
```

Можно присвоить ограничению отдельное имя. Это улучшит сообщения об ошибках и позволит ссылаться на это ограничение, когда понадобится изменить его. Чтобы создать именованное ограничение, необходимо указать ключевое слово CONSTRAINT, а за ним идентификатор и собственно определение ограничения.

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric CONSTRAINT positive_price CHECK (price > 0)
);
```

Ограничение-проверка может также ссылаться **на несколько столбцов.**

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric CHECK (price > 0),
    discounted_price numeric CHECK (discounted_price > 0),
    CHECK (price > discounted_price)
);
```

Например, если хранится обычная цена и цена со скидкой, так можно гарантировать, что цена со скидкой будет всегда меньше обычной.

Про первые два ограничения можно сказать, **что это ограничения столбцов**, тогда как **третье является ограничением таблицы, так как оно написано отдельно от определений столбцов.**

Ограничения столбцов также можно записать в виде ограничений таблицы, тогда как обратное не всегда возможно, так как подразумевается, что ограничение столбца ссылается только на связанный столбец.

Ограничение-проверка **удовлетворяется, если выражение принимает значение true или NULL.** Так как результатом многих выражений с операндами NULL будет значение NULL, **такие ограничения не будут препятствовать записи NULL в связанные столбцы.** Чтобы гарантировать, что столбец не содержит значения NULL, можно использовать ограничение **NOT NULL.**

Ограничение NOT NULL

Ограничение NOT NULL гарантирует, что столбец обязательно будет иметь значение для каждой записи, то есть **значение будет не нулевым**. Таким образом программа не позволит хранить в столбцах пустые значения.

Если к столбцу применено ограничение NOT NULL, вставить новую строку в таблицу без добавления не-NULL-значения в этот столбец невозможно.

```
CREATE TABLE products (
    product_no integer NOT NULL,
    name text NOT NULL,
    price numeric
);
```

```
CREATE TABLE Customers(
Id INTEGER,
FirstName VARCHAR(20) NOT NULL,
LastName VARCHAR(20) NOT NULL,
Age INTEGER);
```

Ограничение NOT NULL всегда записывается как ограничение столбца и **функционально эквивалентно ограничению CHECK (имя_столбца IS NOT NULL), но явное ограничение** NOT NULL **работает более эффективно**. Хотя у такой записи есть недостаток — **назначить имя таким ограничениям нельзя.**

Ограничение DEFAULT

Ограничение DEFAULT определяет значение по умолчанию для столбцов.

Но DEFAULT **не ограничивает тип вводимых данных**, поэтому технически не может быть отнесен к ограничениям. **Тем не менее эта инструкция позволяет реализовать довольно важную функцию: подстановку значений по умолчанию, когда пользователь их не вводит.**

```
CREATE TABLE Customers2 (
CustomerName1 VARCHAR(46) NOT NULL,
CustomerName2 VARCHAR(46) NOT NULL,
CustomerAge INTEGER DEFAULT 18);
```

Если столбец таблицы определен как NOT NULL, но ему было присвоено значение по умолчанию, то в операторе INSERT не нужно явно присваивать значение этому столбцу, чтобы вставить новую строку в таблицу.

Ограничение UNIQUE

Unique значит «уникальный», и это название полностью отражает суть ограничения.

Ограничение UNIQUE гарантирует, что **никакие два значения в определяемом столбце не будут одинаковыми.**

ОГРАНИЧЕНИЯ СТОЛБЦА	ОГРАНИЧЕНИЯ ТАБЛИЦЫ

`CREATE TABLE products (`
`  product_no integer UNIQUE,`
`  name text,`
`  price numeric`
`);`
	`CREATE TABLE products (`
	`  product_no integer,`
	`  name text,`
	`  price numeric,UNIQUE (product_no)`
	`);`

При добавлении ограничения уникальности будет **автоматически создан уникальный индекс-B-дерево для столбца или группы столбцов**, перечисленных в ограничении. Условие уникальности, распространяющееся только на некоторые строки, нельзя записать в виде ограничения уникальности. **Ограничение уникальности нарушается, если в таблице оказывается несколько строк, у которых совпадают значения всех столбцов, включённых в ограничение.**

Два значения NULL при сравнении **никогда не считаются равными**. Это означает, что даже при наличии ограничения уникальности в таблице можно сохранить строки с дублирующимися значениями, если они содержат NULL в одном или нескольких столбцах ограничения.

Ограничение PRIMARY KEY

Ограничение первичного ключа означает, что образующий его столбец или группа столбцов может быть уникальным идентификатором строк в таблице. Для этого требуется, чтобы значения были одновременно уникальными и отличными от NULL.

ОГРАНИЧЕНИЯ СТОЛБЦА	ОГРАНИЧЕНИЯ ТАБЛИЦЫ

`CREATE TABLE products (`
`  product_no integer PRIMARY KEY,`
`  name text,`
`  price numeric`
`);`
	`CREATE TABLE products (`
	`  product_no integer,`
	`  name text, price numeric,`
	`  PRIMARY KEY (product_no)`
	`);`

Первичный ключ обычно состоит из одного столбца в таблице, **однако несколько столбцов могут составлять первичный ключ.**

Таблица может иметь максимум один первичный ключ. *(**Ограничений уникальности и ограничений NOT NULL**, которые функционально почти равнозначны первичным ключам, **может быть сколько угодно**, но назначить ограничением первичного ключа можно только одно.)*

Ограничение FOREIGN KEY

Ограничение внешнего ключа указывает, что значения столбца (или группы столбцов) должны соответствовать значениям в некоторой строке другой таблицы. Это называется ссылочной целостностью двух связанных таблиц.

ОГРАНИЧЕНИЯ СТОЛБЦА

```
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer REFERENCES products (product_no),
    quantity integer
);
```

ОГРАНИЧЕНИЯ ТАБЛИЦЫ

```
CREATE TABLE t1 (
  a integer PRIMARY KEY,
  b integer,
  c integer,
  FOREIGN KEY (b, c) REFERENCES other_table (c1, c2)
);
```

Число и типы столбцов в ограничении должны соответствовать числу и типам целевых столбцов.

Иногда имеет смысл задать в ограничении **внешнего ключа в качестве «другой таблицы» ту же таблицу**; такой внешний ключ называется ***ссылающимся на себя.***

```
CREATE TABLE tree (
    node_id integer PRIMARY KEY,
    parent_id integer REFERENCES tree,
    name text,
    ...);
```

Для узла верхнего уровня parent_id будет равен NULL, пока записи с отличным от NULL parent_id будут ссылаться только на существующие строки таблицы.

Согласованность данных между родительскими и дочерними таблицами:

ON UPDATE

ON DELETE

NO ACTION

CASCADE

RESTRICT

SET DEFAULT

SET NULL

product
product_id
product_name
product_subcategory
brand
category
price

product_id	product_name	product_subcategory	brand	category	price
1	A	headphone	Sony	electronics	$280
2	B	sneaker	Nike	shoes	$70
3	C	shirt	Levi's	clothing	$50
4	D	baseball bat	Louisville Slugger	sports	$100

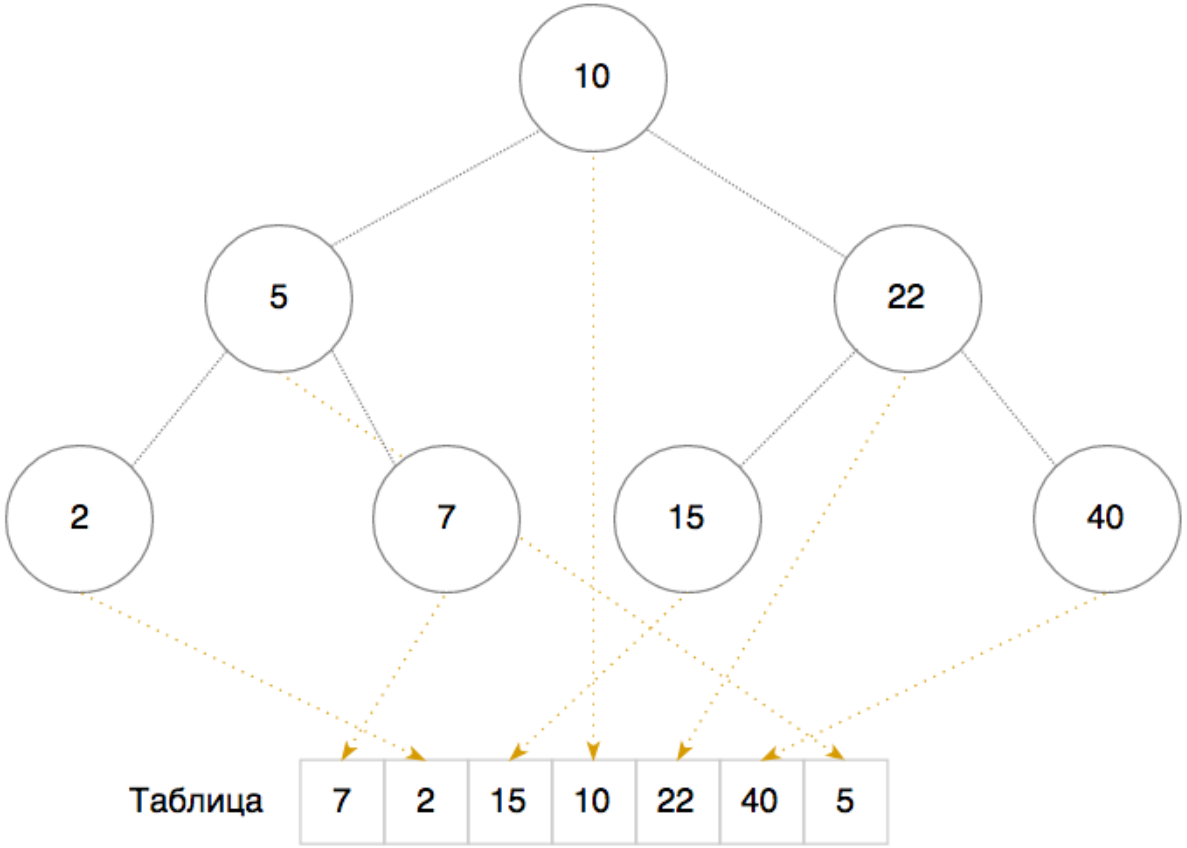
```
SELECT COUNT(*)
FROM product
WHERE category = 'electronics';
```

Для выполнения запроса база данных (БД) должна просканировать все *N миллионов строк*, чтобы проверить каждую запись на соответствие. Предположительно, *время* этой операции составляет *4 секунды*.

Можно ли быстрее? ***Конечно.***

ИНДЕКСЫ В СТАНДАРТЕ ЯЗЫКА

Индекс представляет из себя отдельную таблицу с отсортированными значениями и ссылками на запись в основной таблице. Сам индекс можно представить как дерево.



Индексы SQL представляют собой специальные таблицы, которые поисковая система базы данных может использовать для ускорения поиска данных. Проще говоря, **индекс является указателем на данные в таблице.**

Индекс можно создать для любого столбца или представления (view), **за исключением столбцов с типами данных для хранения больших объектов**: text, image или varchar(max).

Индекс позволяет **увеличить производительность запросов SELECT и WHERE**, но **замедляет ввод данных с помощью операторов UPDATE и INSERT**. Индексы можно создать или удалить не затрагивая данные.

Создание индекса включает инструкцию **CREATE INDEX**, которая позволяет **указать индекс, таблицу и столбцы или столбцы для индексации**, а также **задать порядок индексации**: по возрастанию или по убыванию.

```
CREATE INDEX index_name ON table_name;
```

```
sbrmvch=# select * from purchases;
 id |        name         |  cost   | user_id
----+---------------------+---------+---------
  1 | M1 MacBook Air      | 1300.99 |       1
  2 | Iphone 14           | 1200.00 |       2
  3 | Iphon 10            |  700.00 |       3
  4 | Iphone 13           |  800.00 |       1
  5 | Intel Core i5       |  500.00 |       4
  6 | M1 MacBook Pro      | 1500.00 |       5
  7 | IMAC                | 2500.00 |       7
  8 | ASUS VIVOBOOK       |  899.99 |       6
  9 | Lenovo              | 1232.99 |       1
 10 | Galaxy S21          |  999.99 |       2
 11 | XIAMI REDMIBOOK 14  |  742.99 |       4
 12 | M1 MacBook Air      | 1299.99 |       8
 13 | ACER                |  799.99 |       7
(13 rows)
```

```
sbrmvch=# create index users_cost on purchases(cost);
CREATE INDEX
```

Теперь необходимо найти покупку, стоимостью 800.00. **Что происходит внутри базы данных?** Она знает, что на колонку ***cost*** создан индекс и **начнет поиск сначала по индексу, начиная с корня и спускаясь вниз по узлам до тех пор, пока не найдет искомое значение.** В итоге поиска мы получаем **указатель на строку со всеми данными из таблицы.**

```
sbrmvch=# select * from purchases;
 id |         name          |  cost   | user_id
----+-----------------------+---------+---------
  1 | M1 MacBook Air        | 1300.99 |       1
  2 | Iphone 14             | 1200.00 |       2
  3 | Iphon 10              |  700.00 |       3
  4 | Iphone 13             |  800.00 |       1
  5 | Intel Core i5         |  500.00 |       4
  7 | IMAC                  | 2500.00 |       7
  8 | ASUS VIVOBOOK         |  899.99 |       6
 10 | Galaxy S21            |  999.99 |       2
 12 | M1 MacBook Air        | 1299.99 |       8
  9 | Lenovo                |  800.00 |       1
 13 | ACER                  |  800.00 |       7
  6 | M1 MacBook Pro        |  800.00 |       5
 11 | XIAMI REDMIBOOK 14    |  800.00 |       4
```

Количество покупок заданной стоимости (без индекса)

```
sbrmvch=# select count(*) from purchases where cost=800.00;
 count
-------
     5
(1 row)

Time: 1,031 ms
```

Создание индекса

```
sbrmvch=# create index users_cost on purchases(cost);
CREATE INDEX
Time: 2,296 ms
```

Количество покупок заданной стоимости (с использованием индекса)

```
sbrmvch=# select count(*) from purchases where cost=800.00;
 count
-------
     5
(1 row)

Time: 0,929 ms
```

Для удаления индекса используется команда:

```
DROP INDEX имя_индекса;
```

```
sbrmvch=# drop index users_cost;
DROP INDEX
```

Когда не следует использовать индексы?

Хотя индексы предназначены для повышения производительности базы данных, бывают случаи, **когда их использовать не следует:**

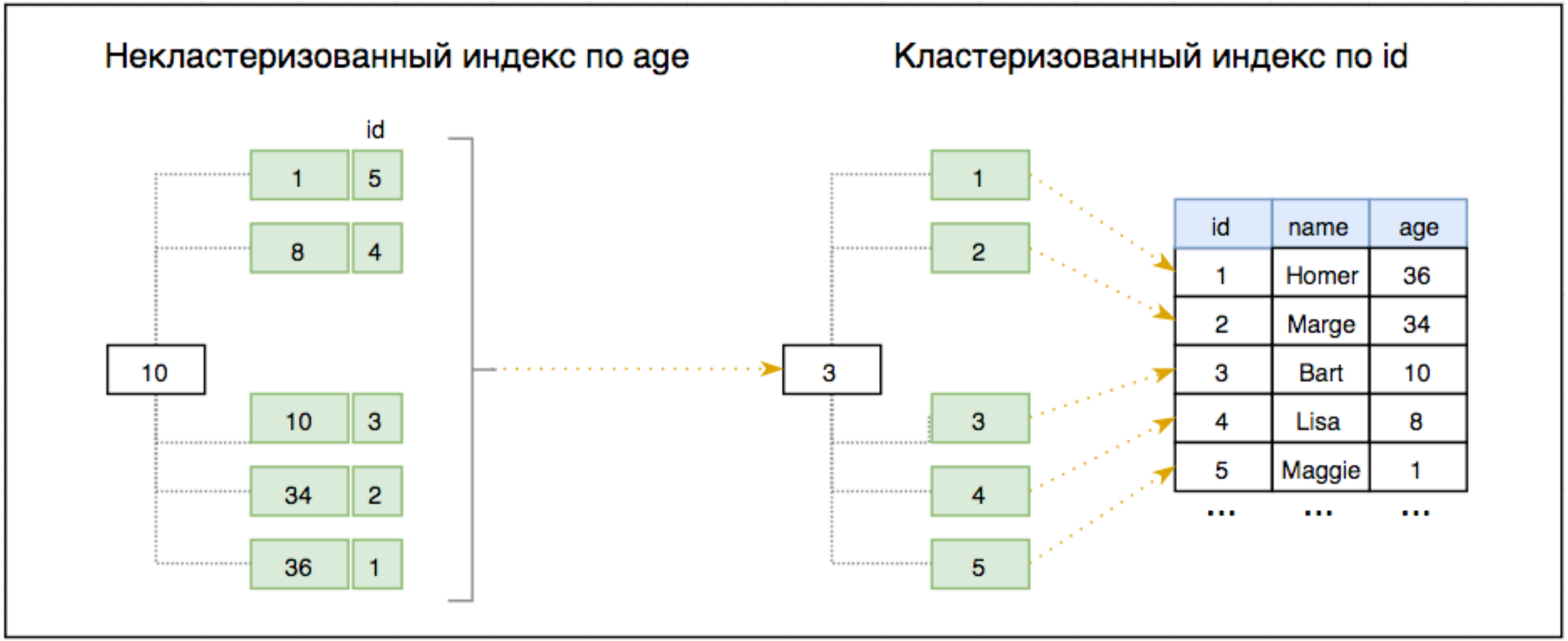
- индексы не должны использоваться **в небольших таблицах;**
- индексы не должны использоваться в таблицах, **в которых часто выполняются массовые операции UPDATE или INSERT;**
- индексы не должны использоваться для **столбцов, содержащих большое количество значений NULL;**
- столбцы, которые часто **обрабатываются**, также не должны индексироваться.

КЛАСТЕРИЗОВАННЫЙ И НЕКЛАСТЕРИЗОВАННЫЙ ИНДЕКС

Кластеризованный индекс уже **хранит данные в своих листьях**. Он находится в **отсортированном виде** и создается **только один на всю таблицу**. Обычно на колонку ***id***, которая является первичным ключом (***primary key***), по умолчанию **создается кластеризованный индекс**.

Некластеризованный индекс хранит в своих листьях **ссылки на записи кластеризованного индекса или на записи из кучи** *(куча (heap) это просто неотсортированные данные таблицы)*, если кластеризованного индекса нет.

Кластеризованный индекс это не отдельная таблица, а просто отсортированная таблица по выбранной колонке.


Некластеризованный индекс по age
Кластеризованный индекс по id
id
1
5
8
4
10
10
3
34
2
36
1
3
1
2
3
4
5
id
name
age
1
Homer
36
2
Marge
34
3
Bart
10
4
Lisa
8
5
Maggie
1
...
...
...

ПАРАМЕТРЫ ИНДЕКСА

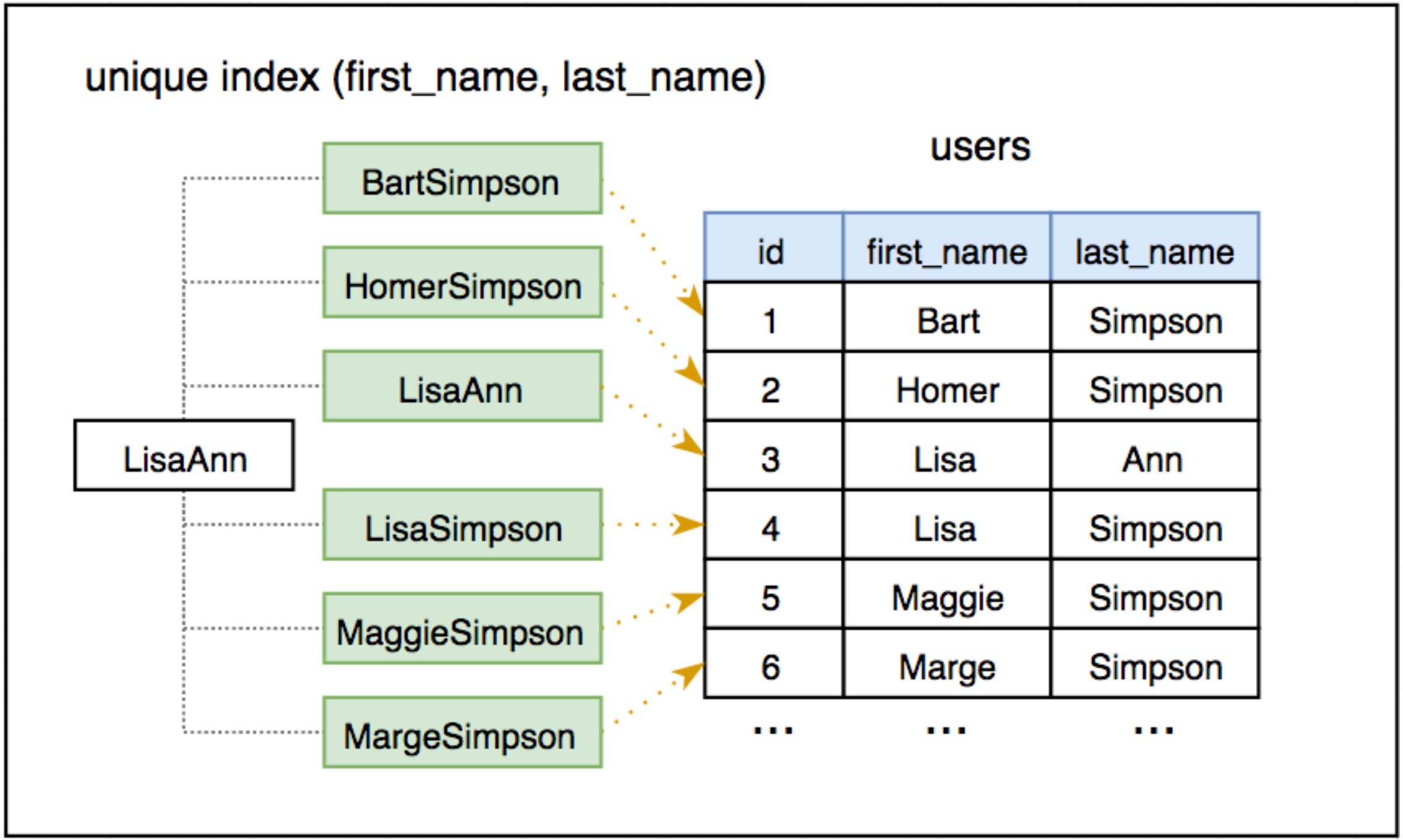
```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ [ IF NOT EXISTS ]name ]
ON table_name [ USING method ]
({ column | ( expression ) } [ ASC | DESC ]
[ NULLS { FIRST | LAST } ] [, ...]) [ WHERE predicate ]
```

Уникальный индекс

Такой индекс обеспечивает **уникальность значений в индексируемой колонке**.

```
CREATE UNIQUE INDEX users_uid_idx ON users (uid);
```

В этом случае **обеспечивается уникальность значений на все колонки, но не на каждую отдельно.** То есть, **если создается составной уникальный индекс на поля first_name и last_name**, то это означает, **что повторяющихся имен + фамилий не будет**, но **разрешается использовать одинаковые либо имена, либо фамилии по отдельности**. При создании первичного ключа уникальный индекс создается автоматически.


unique index (first_name, last_name)
users
BartSimpson
HomerSimpson
LisaAnn
LisaAnn
LisaSimpson
MaggieSimpson
MargeSimpson
id
first_name
last_name
1
Bart
Simpson
2
Homer
Simpson
3
Lisa
Ann
4
Lisa
Simpson
5
Maggie
Simpson
6
Marge
Simpson
...
...
...

Блокировка

При **обычном создании индекса** БД **блокирует вставку, изменение и удаление в таблице**. В некоторых случаях бывает, что индекс создается не быстро, а таблица обновляется очень часто и не хотелось бы блокировать ее изменение. **Для этого есть параметр – CONCURRENTLY**.

```
CREATE INDEX users_age__idx CONCURRENTLY ON users (age);
```

В таком случае создание индекса **не будет блокировать изменение таблицы**, но **время** на само **создание увеличится**. Некоторое время индекс использоваться не будет.

Сортировка

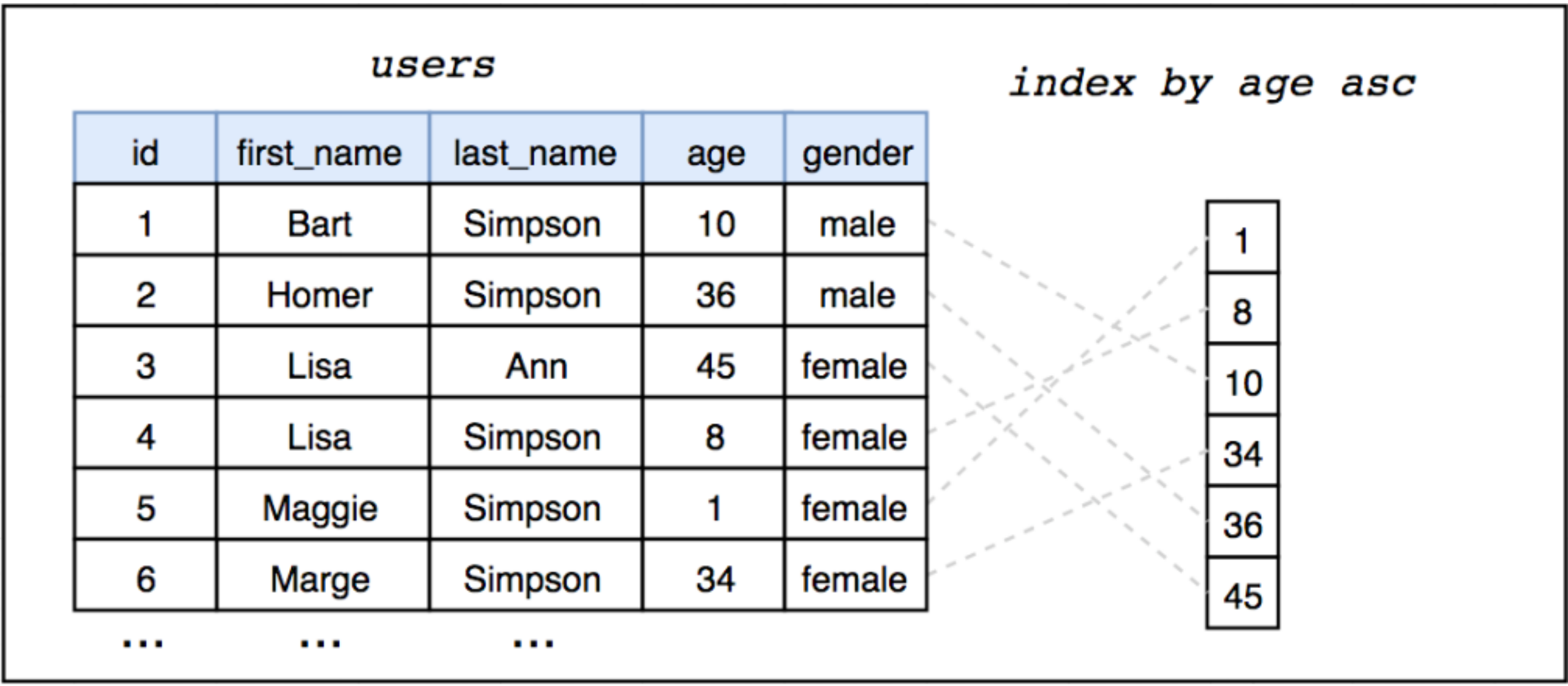
Индекс бывает полезен при сортировке выборки. При создании индекса можно указать ему порядок сортировки **ASC или DESC**.

```
CREATE INDEX users_age__idx ON users (age) ASC;
```

users

id	first_name	last_name	age	gender
1	Bart	Simpson	10	male
2	Homer	Simpson	36	male
3	Lisa	Ann	45	female
4	Lisa	Simpson	8	female
5	Maggie	Simpson	1	female
6	Marge	Simpson	34	female
...	...	...		

index by age asc



Функциональный индекс

Индекс требуется не по самому полю, а **по результату выражения**.

```
sbrmvch=# SELECT * FROM users
WHERE (name||' '||profession) = 'Amigos QA';
 id |  name  | profession
----+--------+------------
  8 | Amigos | QA
(1 row)

Time: 2,604 ms
```

```
sbrmvch=# SELECT * FROM users
WHERE (name||' '||profession) = 'Amigos QA';
 id |  name  | profession
----+--------+------------
  8 | Amigos | QA
(1 row)

Time: 0,945 ms
```

```
CREATE INDEX users_idx
ON users ((name || ' ' || profession));
```

```
sbrmvch=# CREATE INDEX users_idx
ON users ((name ||' '|| profession));
CREATE INDEX
Time: 3,071 ms
```

Функциональный индекс не хранит выражение, а наоборот – **результат выражения**. Благодаря этому сильно **ускоряется выборка**, т.к. отпадает необходимость высчитывать выражение для каждой записи. Но есть обратная сторона – **сильно падает скорость создания и обновления записи**, т.к. рассчитывается новое значение.

Частичный индекс

Частичный индекс — это индекс, который строится по подмножеству строк таблицы, определяемому условным выражением (оно называется предикатом частичного индекса). Такой индекс содержит **записи только для строк, удовлетворяющих предикату**. Частичные индексы довольно специфичны, но в ряде ситуаций они могут быть очень полезны.

```
create index us on users(name) where profession='QA';
```

Частичные индексы могут быть полезны тем, что **позволяют исключить из индекса значения, которые обычно не представляют интереса.**

Покрывающий индекс

Такие индексы специально предназначены для включения столбцов, которые требуются в определённых часто выполняемых запросах. Так как в запросах обычно нужно получить не только столбцы, по которым выполняется поиск, СУБД позволяет создать индекс, в котором некоторые столбцы будут просто «дополнительной нагрузкой», но не войдут в поисковый ключ. Это реализуется предложением `INCLUDE`, в котором перечисляются дополнительные столбцы.

```
sbrmvch=# SELECT name  FROM users
WHERE profession = 'QA';
   name
--------
 Bob
 Kate
 Amigos
 Sima
 Kim
 Kate
(6 rows)
```

Индекс может удовлетворить подобные запросы при сканировании только индекса, так как значение profession можно получить из индекса, не обращаясь к данным в куче.

```
sbrmvch=# CREATE INDEX tab_p_n ON users(name) INCLUDE (profession);
CREATE INDEX
```