**创建索引**

**1.创建索引的语法**

CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ name ] ON table\_name [ USING method ]

( { column\_name | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] )

[ WITH ( storage\_parameter = value [, ... ] ) ]

[ TABLESPACE tablespace\_name ]

[ WHERE predicate ]

CONCURRENTLY：通常创建索引会锁表，阻塞插入，更新和删除操作；并发(CONCURRENTLY)创建索引则不会

USING method：指定不同索引类型

Opclass：指定操作符类 ，操作符类指明该索引用于该字段时要使用的操作符。

create index idx\_t1\_id on t1(id);--指定索引名称

create index on t1(id);--自动生成索引名称

create index on t1 using btree(id);--创建btree索引

create index on t1 using hash(name);--创建hash索引

create index on t1(id text\_pattern\_ops);--支持操作符类

CREATE INDEX on t1(age) WITH (FILLFACTOR=80);--指定存储参数

create index on t1(id) tablespace ts\_index;

B-tree索引以升序方式存储，并将空值放在最后。

可以在创建B-tree索引时通过ASC、DESC、NULLS FIRST和NULLS LAST选项来改变索引的排序，例如：

CREATE INDEX ON t1 (age DESC);--降序创建索引

CREATE INDEX ON t1 (age NULLS FIRST); --指定空值排在前面

explain  select \* from t1 where age is null;

**2.查看索引**
\d t1

\di

SELECT tablename,indexname,indexdef FROM pg\_indexes WHERE schemaname = 'public' ORDER BY tablename,indexname;

SELECT indexname,indexdef FROM pg\_indexes WHERE tablename = 't1';  --显示表的所有索引

**并发创建索引**

创建索引会锁表，并且会把表中的数据全部读一遍，所用时间由表的大小决定， 较大的表，花费的时间很久。 创建索引的过程中， 可以查询表，但增、 删、 改等操作需要等索引建完后才能进行，如果表比较大， 可能需要几十分钟，甚至几个小时。 因此PostgreSQL提供了并发建索引的方法，即创建索引不阻塞更新。使用CONCURRENTLY， PostgreSQL会执行表的两次扫描， 因此需要更长的时间来建索引。

CREATE INDEX CONCURRENTLY on t1(id);

使用concurrently关键字创建索引的各阶段如下：

1、开启事务1，拿到当前snapshot1。

2、扫描t1表前，等待所有修改过t1表（写入、删除、更新）的事务结束。

3、扫描t1表，并建立索引。

4、结束事务1。

5、开启事务2，拿到当前snapshot2。

6、再次扫描t1表前，等待所有修改过t1表（写入、删除、更新）的事务结束。

7、在snapshot2之后启动的事务对t1表执行的DML，会修改这个idx\_t1\_id的索引。

8、再次扫描t1表，更新索引。（从TUPLE中可以拿到版本号，在snapshot1到snapshot2之间变更的记录，将其合并到索引）

9、上一步更新索引结束后，等待事务2之前开启的持有snapshot的事务结束。

10、结束索引创建。索引可见。

注意点：

1.此选项只能指定一个索引的名称

2.普通CREATE INDEX命令可以在事务内执行，但是CREATE INDEX CONCURRENTLY不可以在事务内执行

3.分区表不支持CONCURRENTLY方式创建索引。

为了减少等待的时间，尽量避免创建索引过程中，两次SCAN之前对被创建索引表实施长事务，并且长事务中包含修改被创建索引的表。在第二次SCAN前，尽量避免开启长事务。

示例

创建测试表：CREATE TABLE t1(id serial primary key, info text);

插入测试数据：INSERT INTO t1 select n from generate\_series(1,5000000) n;

开两个psql窗口，一个窗口中建索引，另一个窗口中删除一条数据

CREATE INDEX on t1(info);

DELETE FROM t1 where id=1; --等待

创建索引时启用CONCURRENTLY选项

CREATE INDEX CONCURRENTLY on t1(info);

另一个窗口的删除操作会立即执行

DELETE FROM t1 where id=2;--立即删除

DELETE FROM t1 where id=3;

**重建索引**

重建指定索引

reindex index idx\_t1\_id;

并发重建索引

reindex index CONCURRENTLY idx\_t1\_t1 ;

重建表的所有索引

REINDEX TABLE t1;

重建模式下的所有索引：

REINDEX SCHEMA public;

重建数据库下的所有索引：

REINDEX DATABASE postgres;

重建数据库下的所有系统索引：

REINDEX SYSTEM postgres;

并发创建索引的时候， 如果在索引创建过程中被强行取消可能会留下一个无效的索引， 这个索引仍然会导致更新速度变慢。如果所创建的是一个唯一索引， 这个无效的索引还会导致插入重复值失败

CREATE INDEX CONCURRENTLY on t1(info);--ERROR

查看无效索引

\ d t1--可以查看INVALID状态的索引

select indexrelid::regclass index\_name,indrelid::regclass table\_name from pg\_index where not indisvalid;--返回这些索引的完整列表

插入重复数据， 无效唯一索引的约束仍然有效：

INSERT INTO t1 VALUES(10,'10');--ERROR:

手动删除此索引就可以了

DROP INDEX idx\_t1\_info;

**修改索引**

语法格式：

ALTER INDEX name RENAME TO new\_name

ALTER INDEX name SET TABLESPACE tablespace\_name

ALTER INDEX name SET ( storage\_parameter = value [, ...] )

ALTER INDEX name RESET ( storage\_parameter [, ... ] )

重命名索引：

ALTER INDEX idx\_t1\_name RENAME TO idx\_t1\_name2;

移动表空间：

ALTER INDEX idx\_t1\_name SET TABLESPACE ts\_index;

修改索引的填充因子：

ALTER INDEX idx\_t1\_name SET (fillfactor = 75);

重置填充因子为默认值：

ALTER INDEX idx\_t1\_name RESET (fillfactor);

**删除索引**

语法格式：

DROP INDEX [ IF EXISTS ] name [, ...] [ CASCADE |RESTRICT ]

如果索引存在则删除， 如果不存在也不报错：

DROP INDEX IF EXISTS idx\_t1\_name;

删除索引时， 默认使用选项 RESTRICT， 所以加不加关键字 RESTRICT 效果都是一样的， 如果有对象依赖该索引， 则会删除失败， 而使用CASCADE选项表示当有依赖这个索引的对象时， 一并把这些对象删除掉， 如外键约束。 示例如下：

CREATE TABLE class(

id int,

name varchar(40)

);

CREATE UNIQUE INDEX ON class(id);

CREATE TABLE student(

id int primary key,

name varchar,

age int,

class\_id int REFERENCES class(id)

);

如果表 student 上的外键引用了表 class 上的唯一索引， 想删除此索引， 将会操作失败：

DROP INDEX idx\_t1\_name;--ERROR

此时加上 CASCADE 删除成功：

DROP INDEX idx\_t1\_name CASCADE;--wainning

**索引排序**

一个索引在每一个索引列上只能支持一种排序规则。如果需要多种排序规则，需要多个索引。

CREATE TABLE t1 ( id integer, name varchar COLLATE "zh\_CN" );

CREATE INDEX ON t1 (name);

该索引自动使用列上的排序规则：

SELECT \* FROM t1 WHERE name > 'xxx';

排序规则不同，无法使用索引：

SELECT \* FROM t1 WHERE name > 'xxx' COLLATE "C";

可以在创建索引时指定排序规则：

CREATE INDEX ON t1 (name COLLATE "C");;