# Postgresql 索引

## 概述

本文基于实际场景，从概念、逻辑与物理模型上，跟踪分析Postgresql索引原理。

## 场景

已有一个表：

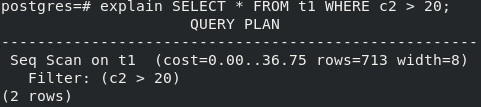
CREATE TABLE t1 (c1 INT, c2 INT);

INSERT INTO t1 VALUES (1,11),(2,17),(3,5),(4,19),(5,3),(6,13),(7,29),(8,23),(9,7),(10,31);

CREATE INDEX i1 ON t1 USING BTREE(c2 ASC);

对表进行范围查询：

SELECT \* FROM t1 WHERE c2 > 20;



实际执行效果：

执行器会顺序扫描全表，依次获取表中每一条数据，并使用过滤条件(c2 > 20)对每条数据进行过滤。

伪代码流程如下：

for loop 10:

row = read\_from(t1)

compare(row, 20)

ExecutePlan ExecScan ExecQual

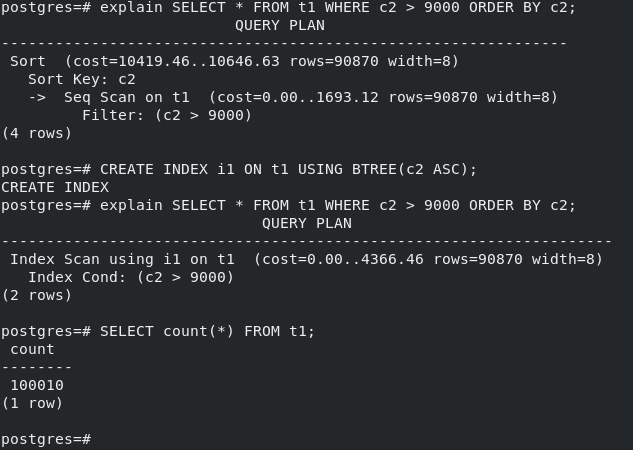
当数据量较大时，顺序扫描开销较大，只能使用索引扫描

with open("./tmp.sql", 'a+') as fd:

for i in range(100000):

c2 = randint(1, 100000)

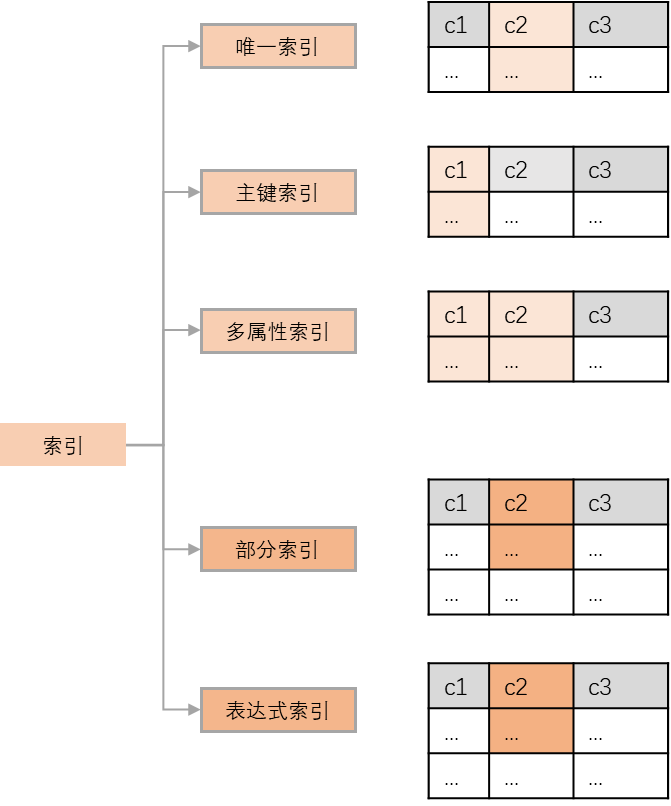
fd.writelines("INSERT INTO t1 VALUES (%d, %d);\n" % (i, c2))



碰壁 1：当数据量较少时，优化器根本就不会走索引扫描，被迫构造多数据场景

## 索引

根据索引与表中各列的关系，可将索引分为唯一索引、主键索引、多属性索引。根据与各行的关系，可分为部分索引和表达式索引。



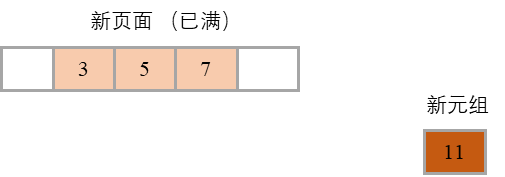
## 索引结构

### Btree

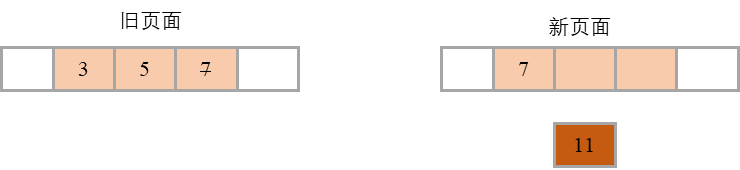
碰壁 2：由于本地使用了PostgreSQL 9.2.4，该版本不支持bt\_metap，bt\_page\_items，bt\_page\_stats函数，所以，无法查看Btee的详细内容

\_bt\_buildadd函数

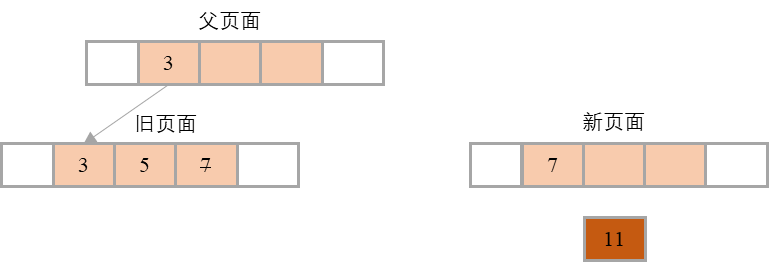
1. 已建立Index，Index仅有1个页面（即节点），而页面已满；此时，有一个元组待添加



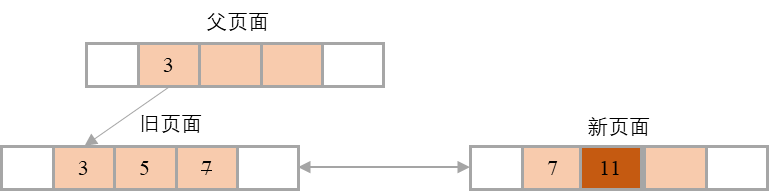
1. 由于当前页面已满，所以需新建页面；新建页面时，会将旧页面最大元组复制到新页面



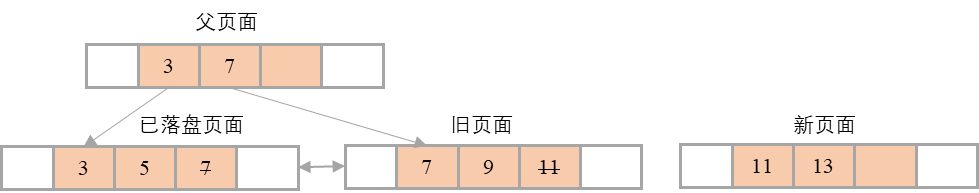
1. 此时，需为旧页面建立父页面；新建父页面时，会将旧页面的最小元组复制到父页面



1. 此时，可将新元组插入新页面中；并且，将旧页面与新页面相连接



1. 由于旧页面也不会再被用到，可以将其落盘；循环执行上述步骤，直至add所有元组



### Hash