**Stored procedure that insert 10000 rows in customer table**

show tables;

delimiter $$

create procedure *testdata*()

begin

declare NUMROWS int default 10000;

declare count int default 0;

declare namevalue varchar(40) default 'mohamed';

declare email\_value varchar(40) default null;

declare cust\_password varchar(30) default '123';

declare cust\_username varchar(30);

declare cust\_adress varchar(30);

declare cust\_city varchar(30) default 'cairo';

declare cust\_country varchar(30) default 'egypt';

declare active\_value boolean default false;

while count < NUMROWS do

insert into customer( custName, custEmail, custPassword, custAdress, custCity, custCountry)

VALUES(*concat*('user',count),*concat*('user', count, '@ecommerce.com' ),*concat*('pass',count),*concat*('addres',count), *concat*('city',count),*concat*('country',count));

set count := count + 1;

end while;

end $$

delimiter ;

drop procedure *testdata*;

call *testdata*();

show tables;

select *count*(\*) from customer;

select \* from customer where custId between 1 and 50;

**Stored procedure that insert 10000 rows in product table**

delimiter $$

create procedure *insertDateintoProducts*()

begin

declare NUMROWS int default 10000;

declare count int default 0;

while count < NUMROWS do

insert into product(prodName, prodDescr, prodPrice, availability, prodImageUrl, prodComments, prodDiscount, prodShipCost)

VALUES(*concat*('product',count),*concat*('user', count, '@ecommerce.com' ),*floor*(10000\**rand*()),*floor*(126\**rand*()),*concat*('imgurl',count),*concat*('commentofproduct',count), *floor*(300\**rand*()),*floor*(10\**rand*()));

set count := count + 1;

end while;

end $$

delimiter ;

drop procedure *tesdata*;

call *insertDateintoProducts*();

select \* from product;

**Insert categories**

Didnt create a stored procrdure because i want to manually insert parentcategory for each category

If parent category is null means it is the parent

inseinsert into category(catName, catDiscr, catParentId, catProdId)

VALUES('furnature', 'furnature', null,1);

rt into category(catName, catDiscr, catParentId, catProdId)

VALUES('books', 'books description', null,5);

insert into category(catName, catDiscr, catParentId, catProdId)

VALUES('history', 'furnature', 2 ,6);

insert into category(catName, catDiscr, catParentId, catProdId)

VALUES('science', 'science description', 2 ,7);

**Insert data in product category table:**

wont work because select statement here ROW\_COUNT() returns the number of rows affected by the last INSERT, UPDATE, or DELETE statement, not the SELECT statement. To check if a row was returned by the SELECT statement, you can use a cursor or a variable to store the result and check if it is NULL. In this case, you can use a cursor to fetch the result into a variable and then check if the variable is NULL or not.

delimiter $$

create procedure *insertDateintoProductCategory*()

begin

declare NUMROWS int default 100;

declare count int default 0;

while count < NUMROWS do

INSERT INTO productcategory(prodId, catId, prodCatDefault)

VALUES (*FLOOR*(500 \* *RAND*()), *FLOOR*(6 \* *RAND*()),'1');

set count := count + 1;

end while;

end $$

delimiter ;

call *insertDateintoOrderItem*();

**delimiter $$**

**create procedure *insertDateintoProductCategory*()**

**begin**

**declare NUMROWS int default 100;**

**declare count int default 0;**

**declare categoryIdvar int ;**

**declare productIdVar int;**

**DECLARE result INT;**

**DECLARE cursor1 CURSOR FOR**

**SELECT 1 FROM productcategory WHERE prodId = productIdVar AND catId = categoryIdvar;**

**DECLARE CONTINUE HANDLER FOR NOT FOUND SET result = NULL;**

**while count < NUMROWS do**

**set productIdVar := *FLOOR*(1000 \* *RAND*())+1;**

**set categoryIdvar := *FLOOR*(6 \* *RAND*())+1;**

**open cursor1;**

**fetch cursor1 into result;**

**if result IS NULL THEN**

**INSERT INTO productcategory(prodId, catId, prodCatDefault)**

**VALUES (productIdVar, categoryIdvar,'1');**

**END IF;**

**close cursor1;**

**set count := count + 1;**

**end while;**

**end $$**

**delimiter ;**

**call *insertDateintoProductCategory*();**

**DROP procedure *insertDateintoProductCategory*;**

**select *count*(\*) from productcategory;**

**Insert in orders table:**

delimiter $$

create procedure *insertDateintoOrders*()

begin

declare NUMROWS int default 100;

declare count int default 0;

declare active\_value boolean default false;

while count < NUMROWS do

INSERT INTO orders (ordDiscount, ordTax, ordTotal, ordShipping, ordDate, ordShipDate, ordCustId)

VALUES (*FLOOR*(300 \* *RAND*()), *FLOOR*(300 \* *RAND*()), *FLOOR*(3000 \* *RAND*()), *FLOOR*(100 \* *RAND*())

, *DATE\_FORMAT*(*DATE*(*NOW*()) - INTERVAL *FLOOR*(10000 \* *RAND*()) DAY, '%Y-%m-%d 03:14:07')

,*CONCAT*(*DATE*(*NOW*()) - INTERVAL *FLOOR*(10000 \* *RAND*()) DAY, ' 03:14:07'), *FLOOR*(300 \* *RAND*()));

set count := count + 1;

end while;

end $$

delimiter ;

call *insertDateintoOrders*();

drop procedure *insertDateintoOrders*;

select \* from product;

select \* from orders;

delete from orders;

**Insert in orderitem table**

Note this is many to many table primary key is composite between the two foreign keys

Pk must be unique and that is why in our stored procedure we make a condition to ensure that

We don't insert same pk again

delimiter $$

create procedure *insertDateintoOrderItem*()

begin

declare NUMROWS int default 100;

declare count int default 0;

DECLARE ordItemOrdId INT;

DECLARE ordItemProdId INT;

while count < NUMROWS do

SET ordItemOrdId = *RAND*() \* 1000 + 1;

SET ordItemProdId = *RAND*() \* 100 + 1;

SELECT \* FROM orderItem WHERE ordItemOrdId = ordItemOrdId AND ordItemProdId = ordItemProdId;

IF (*ROW\_COUNT*() = 0) THEN

INSERT INTO orderitem(ordItemOrdId,ordItemProdId,ordItemQty, ordItemPrice, ordItemDiscount, ordItemTax)

VALUES (ordItemOrdId,ordItemProdId,*FLOOR*(5 \* *RAND*()), *FLOOR*(300 \* *RAND*()), *FLOOR*(3000 \* *RAND*()), *FLOOR*(100 \* *RAND*()) );

END IF;

set count := count + 1;

end while;

end $$

delimiter ;

call *insertDateintoOrderItem*();

Rather than denormalize the entire database, however, a more common technique is to pre-compute the result of the join for all customers, and then copy that result into a separate, denormalized table.So, we might create a table that looked something like this

insert into saleshisory(custName,productName , price, qty)

select c.custName,p.prodName,p.prodPrice, oi.ordItemQty from customer c inner join orders o on c.custId = o.ordCustId

inner join orderitem oi on o.ordId = oi.ordItemOrdId

inner join product p on oi.ordItemProdId = p.prodId

**Retrieving categories and sub categories**

-- retriving top level categories {parent = null}

select catName as name, catParentId , catId

from category c

where catParentId is null

union all

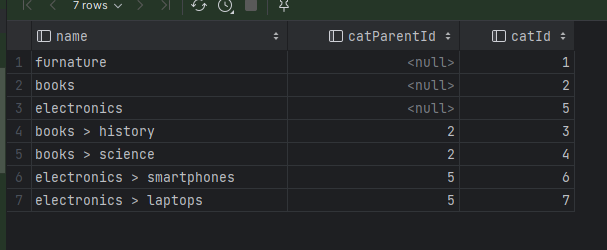
select *concat*( c.catName , ' > ', c1.catName) as name , c1.catParentId , c1.catId

from category c1 inner join category c on c1.catParentId=c.catId

where c.catParentId is null

order by catParentId ,catId;

This is the result of query



-- report of daily revenue

select \* from orders;

select *date*(ordDate) as orderdate, *sum*(ordTotal) as dailyRevenue from orders o

where *date*(ordDate) ='2018-11-17'

group by *date* (ordDate)

order by *date* (ordDate) desc ;

-- total revenue for a specific data

select *sum*(ordTotal) as totalrevenue from orders

where *date\_format*(ordDate,'%Y-%m-%d') = '2018-11-17' ;

Same query using postgres 👍

-- total revenue for a specific data

select *sum*(total\_cost) as totalrevenue from ""

where *to\_char*(orders.order\_data,'MMM') = 'JAN' ;

-- top selling products for a given month

SELECT prodName, *sum*(ordItemQty) as nofitems FROM product p

inner join orderitem oi on p.prodId = oi.ordItemProdId

inner join orders o on oi.ordItemOrdId = o.ordId

where *date\_format*(ordDate,'%Y-%m-%d') = '2018-11-17'

group by prodName

order by *sum*(ordItemQty) desc ;

-- list of customers who have placed orders more than 500 in a given month

select c.custName, *sum*(ordTotal) as total

from customer c

inner join orders o on c.custId = o.ordCustId

where *date\_format*(ordDate,'%Y-%m-%d') = '2000-10-31'

group by c.custName

HAVING total >400

order by total desc ;

Inserting fake date to test using postgres

create table temp(t int) ;

insert into temp(t) select *random*()\*100 from *generate\_series*(1,10000);

Where *generate\_series is a built in function*

**Schema for ecommerce app mysql**

create table customer(

custId serial primary key ,

custName varchar(40) not null,

custEmail varchar(40) not null ,

custPassword varchar(30) not null,

custUsername varchar(20),

custAdress varchar(100),

custCity varchar(20),

custCountry varchar(20)

);

create table orders(

ordId serial primary key ,

ordDiscount numeric(7,2) not null ,

ordTax numeric(7,2) not null,

ordTotal numeric(7,2)not null,

ordShipping numeric(7,2)not null,

ordDate timestamp,

ordShipDate timestamp,

ordCustId int ,

CONSTRAINT fk\_order\_product FOREIGN KEY(ordCustId)

REFERENCES customer(id)

);

create table orderItem(

ordItemOrdId int,

ordItemProdId int,

ordItemQty smallint not null ,

ordItemPrice numeric(7,2) not null ,

ordItemDiscount numeric(7,2) not null ,

ordItemTax numeric(7,2) not null,

constraint pk\_ordItem primary key(ordItemOrdId,ordItemProdId) ,

constraint fk\_orditem\_prod foreign key(ordItemProdId) references product(prodId),

constraint fk\_orditem\_ord foreign key(ordItemOrdId) references orders(ordId)

);

create table product(

prodId serial primary key ,

prodName varchar(40) not null,

prodDescr varchar(60) ,

prodLongDescr varchar(80),

prodSize numeric,

prodColor varchar(40),

-- numaric because it can be used in math operations decimal will make it has more decimal points

prodWeight numeric,

--we can use numaric(7,2)

prodPrice money,

-- it is better to use integer 0 or 1 rather than boolean

availability bool,

prodImageUrl varchar(70),

prodComments varchar(90),

prodDiscount money,

prodShipCost money,

-- available products until now

prodInHand smallint, );

-- assosative entity for manay to many relationship

create table productCategory(

prodId int,

catId int ,

-- we may use boolean or numaric 0,1

prodCatDefault char(1),

constraint fk\_prod\_prodcat foreign key (prodId) references product(prodId) ,

constraint fk\_cat\_prodcat foreign key (catId) references category(catId) ,

-- a given product is containd in only one category , that why i primary key is composite , also to reduce number of coulmns

CONSTRAINT pk\_prodCat primary key (prodId,catId)

);

create table category(

catId serial primary key ,

catName varchar(40) not null,

catDiscr varchar(80),

catParentId int ,

catProdId int ,

-- catParentId refers to catId

constraint fk\_category\_subcategory foreign key (catParentId) references category(catId),

CONSTRAINT fk\_cateogory\_product FOREIGN KEY (catProdId) REFERENCES product(prodId)

);

-- total revenue for a specific data

select *sum*(total\_cost) as totalrevenue from ""

where *to\_char*(orders.order\_data,'MMM') = 'JAN' ;

--top selling products for a given month

SELECT product\_name, *sum*(quantity) as nofitems FROM product p

inner join "" o on o.id = p.order\_id

inner join order\_details od on o.id = od.order\_id

where *to\_char*(order\_data,'MMM') = 'JAN'

group by product\_name

order by nofitems;

--list of customers who have placed orders more than 500 in a given month

select c.name, *sum*(total\_cost) as total

from customer c

inner join "" o on c.id = o.customer\_id

inner join order\_details od on o.id = od.order\_id

where *to\_char*(order\_data,'MMM') = 'JAN'

group by c.name

HAVING total> 500

order by total

**Queries optimized**

-- total no of products in each category

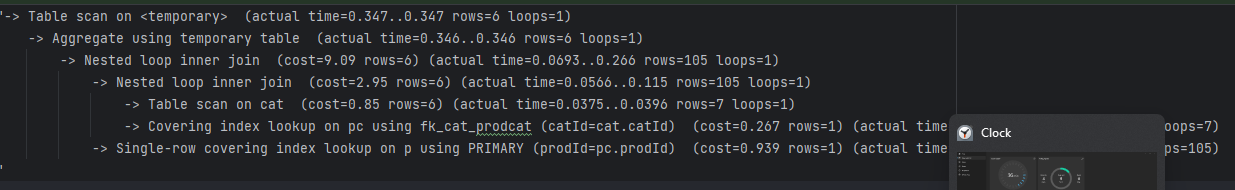
-- normal table scan

explain analyze select *count*(pc.prodId) as totalproducts,cat.catName from category cat

inner join productcategory pc on cat.catId = pc.catId

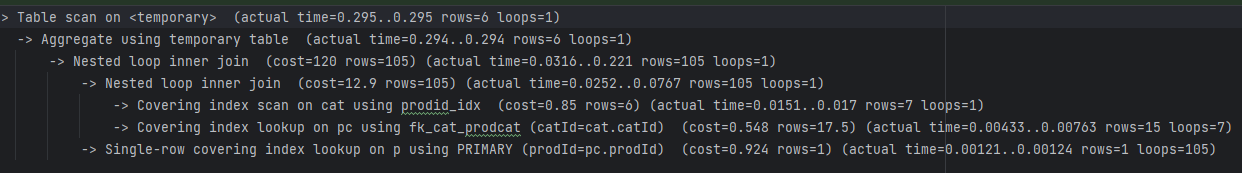
inner join product p on pc.prodId = p.prodId

group by pc.catId;



Adding index to include catname so as to reduce reading from disk and directly reading from index

create index prodid\_idx on category(catId,catName) ;



-- top customers by top spending

explain analyze

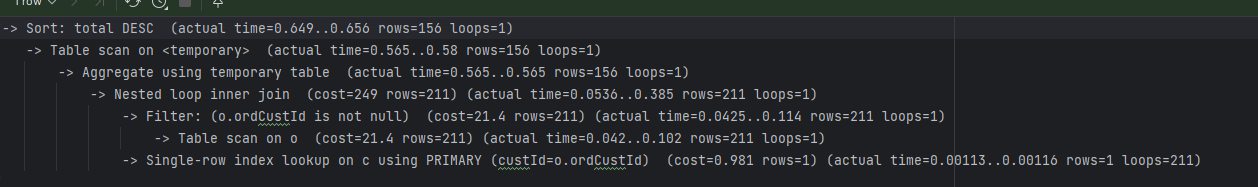
select c.custName,*sum*(o.ordTotal) as total

from customer c

inner join orders o on c.custId = o.ordCustId

group by c.custName

order by *sum*(o.ordTotal) desc;



-- optimized

explain analyze

select c.custName,*sum*(o.ordTotal) as total

from customer c

inner join orders o on c.custId = o.ordCustId

group by c.custId

order by *sum*(o.ordTotal) desc

limit 20;

-- also covering index

create index idx\_custid\_custName on customer(custId, custName) ;



-- most recent orders with customer information with 1000 orders

explain analyze

select c.custName, custEmail, c.custAdress, c.custCity,c.custCountry, o.ordDate

from customer c

inner join orders o on c.custId = o.ordCustId

order by o.ordDate desc

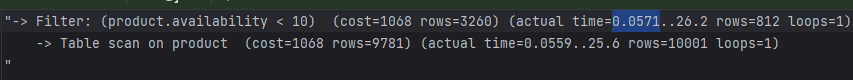
limit 1000;

-- products that has low stock quantities of less than 10 quantities

explain analyze

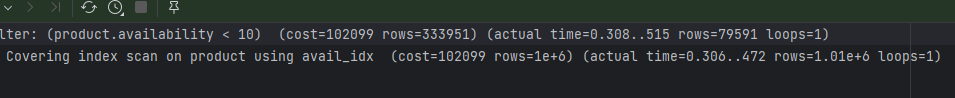
select prodName, availability from product

where availability <10;



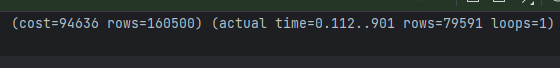
Creating covering index

create index avail\_idx on product(prodId,availability,prodName);



Creating index only on availability field, time reduced

create index producaqty\_idx on product(availability);



-- revenue generated from each product category

explain analyze

select *sum*(price) as revenue , productName

from saleshisory

group by productName;