

实验六 Block Nested Loop Join Description

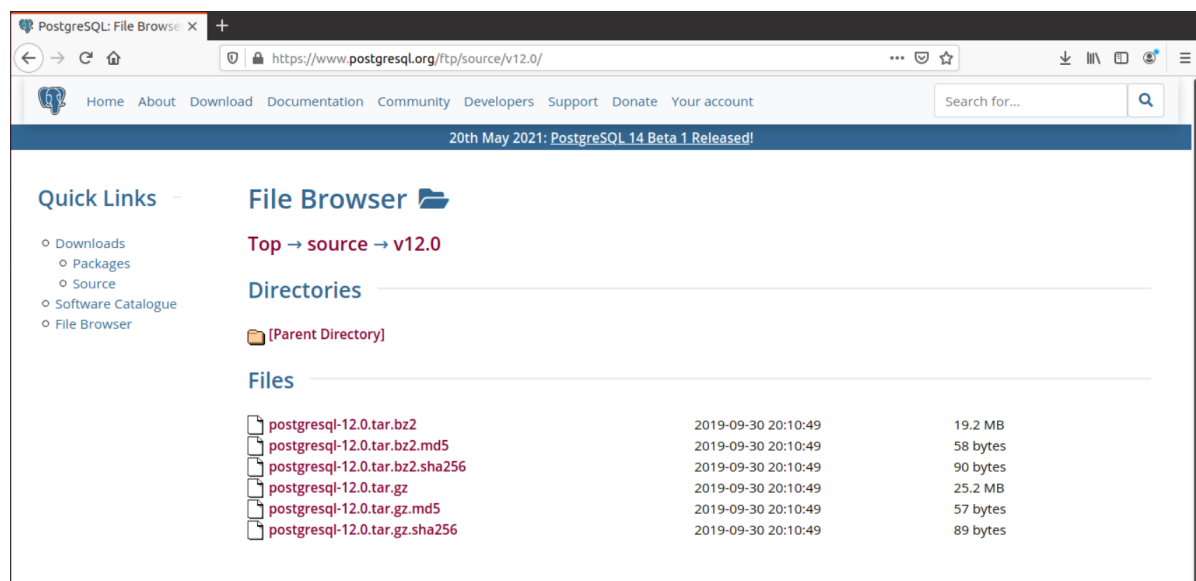
Overview and Motivation

- Before starting, please read the entire assignment fully and carefully.
- For this assignment, you will be implementing the block nested loop join algorithm in Postgres. Block nested loop join is an optimization over simple nested loop join. Simple nested loop join scans the inner relation for every single outer relation tuple to find a match. This may be very slow if there are many outer relation tuples. Block nested loop join improves on that by scanning the inner relation for each "block" of outer relation tuples. This may greatly **reduce the disk IOs performed**.
- We suggest that you get started on this assignment with plenty of time to spare: although the actual code you will need to write for this assignment is relatively simple, understanding the existing code will take some time, as will debugging and testing your changes.

Preparation

Get source code

- URL: [PostgreSQL: File Browser](https://www.postgresql.org/ftp/source/v12.0/)
<https://www.postgresql.org/ftp/source/v12.0/>



Building Postgres

- After getting source code, configure, compile and install PostgreSQL. We will configure PostgreSQL to be installed in the "pgsql" subdirectory of your account (postgres):

```
cd postgresql-12.0

./configure --enable-depend --enable-cassert --enable-debug CFLAGS="-O0" --
prefix=$HOME/pgsql

make

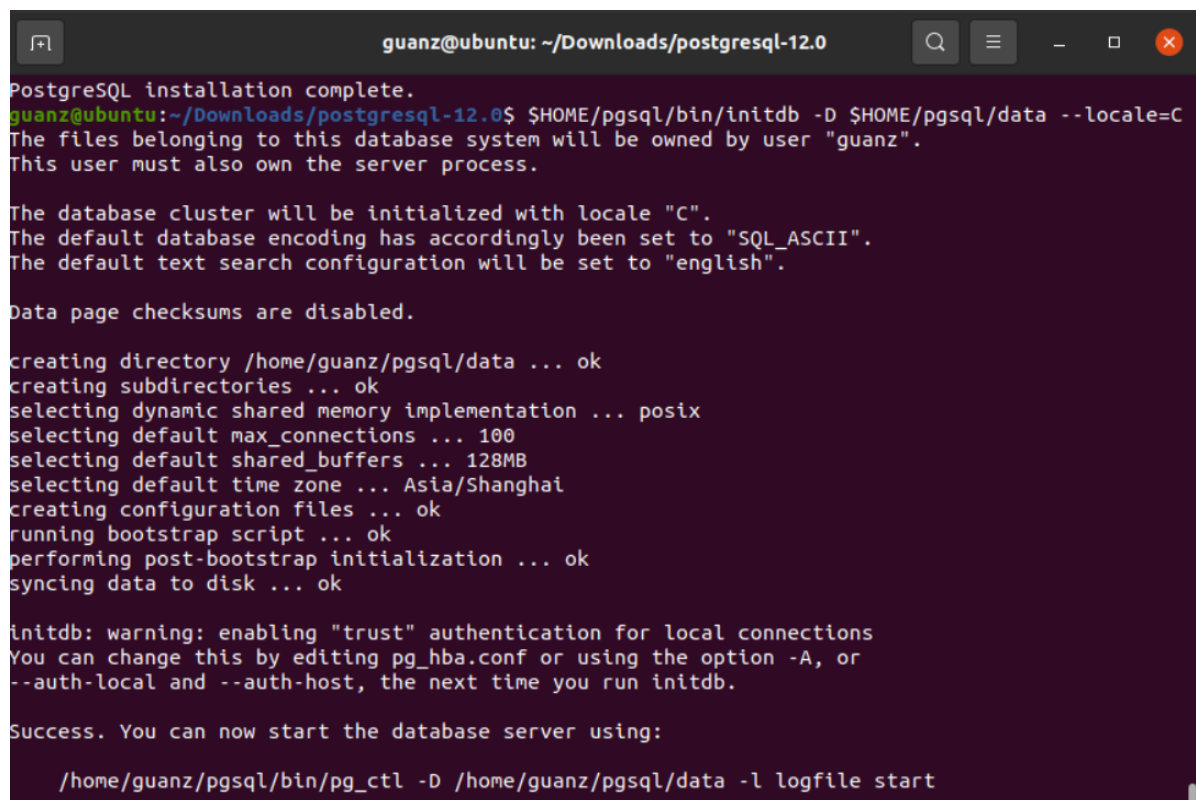
make install
```

- **Note that** after you have modified PostgreSQL (as described below), you should recompile and reinstall the modified version of PostgreSQL using the "make" and "make install" commands, respectively.

Starting PostgreSQL Server

- After building and installing Postgres, and after you delete your old database, **use the initdb command to initialize a local PostgreSQL database cluster**. This probably only has to be done once.

```
$HOME/pgsql/bin/initdb -D $HOME/pgsql/data --locale=C
```



```
guanz@ubuntu: ~/Downloads/postgresql-12.0
PostgreSQL installation complete.
guanz@ubuntu:~/Downloads/postgresql-12.0$ $HOME/pgsql/bin/initdb -D $HOME/pgsql/data --locale=C
The files belonging to this database system will be owned by user "guanz".
This user must also own the server process.

The database cluster will be initialized with locale "C".
The default database encoding has accordingly been set to "SQL_ASCII".
The default text search configuration will be set to "english".

Data page checksums are disabled.

creating directory /home/guanz/pgsql/data ... ok
creating subdirectories ... ok
selecting dynamic shared memory implementation ... posix
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default time zone ... Asia/Shanghai
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... ok
syncing data to disk ... ok

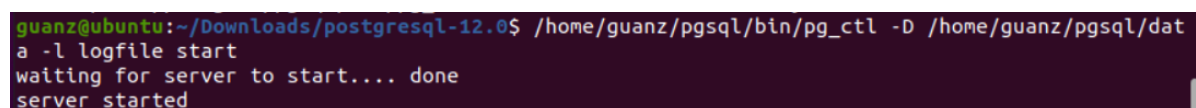
initdb: warning: enabling "trust" authentication for local connections
You can change this by editing pg_hba.conf or using the option -A, or
--auth-local and --auth-host, the next time you run initdb.

Success. You can now start the database server using:

    /home/guanz/pgsql/bin/pg_ctl -D /home/guanz/pgsql/data -l logfile start
```

This command depends on your installation location, please modify it

```
/home/[guanz]/pgsql/bin/pg_ctl -D /home/[guanz]/pgsql/data -l logfile start
```



```
guanz@ubuntu:~/Downloads/postgresql-12.0$ /home/guanz/pgsql/bin/pg_ctl -D /home/guanz/pgsql/dat
a -l logfile start
waiting for server to start.... done
server started
```

- You can use "tail logfile" command to fileprint a few informational messages to the console:

```

guanz@ubuntu:~/Downloads/postgresql-12.0$ /home/guanz/pgsql/bin/pg_ctl -D /home/guanz/pgsql/data -l logfile start
waiting for server to start.... done
server started
guanz@ubuntu:~/Downloads/postgresql-12.0$ tail logfile
2021-06-16 16:32:42.667 CST [21945] LOG:  starting PostgreSQL 12.0 on x86_64-pc-linux-gnu, compiled by gcc (Ubuntu
9.3.0-17ubuntu1~20.04) 9.3.0, 64-bit
2021-06-16 16:32:42.667 CST [21945] LOG:  listening on IPv4 address "127.0.0.1", port 5432
2021-06-16 16:32:42.671 CST [21945] LOG:  listening on Unix socket "/tmp/.s.PGSQL.5432"
2021-06-16 16:32:42.687 CST [21946] LOG:  database system was shut down at 2021-06-16 16:29:46 CST
2021-06-16 16:32:42.691 CST [21945] LOG:  database system is ready to accept connections

```

Connect to the database

- Next, connect to PostgreSQL and run a few simple SQL queries. To start, open a new terminal session (e.g., open a new SSH session). Since PostgreSQL is still running in the foreground in your first terminal session, we'll let it continue running.
- In your new terminal session, connect to the "postgres" database using the "psql" command-line client:

```
$HOME/pgsql/bin/psql postgres
```

```

guanz@ubuntu:~/Downloads/postgresql-12.0$ $HOME/pgsql/bin/psql postgres
psql (12.0)
Type "help" for help.

postgres=# 

```

- Psql is a simple command-line tool for executing SQL queries and examining query results -- it will be useful for testing the correctness of your changes. The "postgres=#" prompt indicates that psql is waiting for you to enter a command which will be executed in the "postgres" database.
- Next, execute a simple SQL command to create a new database table:

```
postgres=# create table test (a int, b int);
```

- Remember that every SQL query must be terminated with a semi-colon (";").

Stopping Postgres

- To exit psql, you can use "Ctrl+D" or the special command "\q".
- To shutdown the PostgreSQL server, you can also use the pg_ctl command:

```

**This command depends on your installation location, please modify it**
/home/[guanz]/pgsql/bin/pg_ctl -D /home/[guanz]/pgsql/data stop

```

Loading Test Data

- You can use the following commands to create a new database name and import the data with these commands:

```

postgres=# CREATE DATABASE similarity;
postgres=# \c similarity
postgres=# \i /home/[guanz]/data/[similarity_data.sql]

```

```
similarity=# select * from addressphone;
```

address	phone
11 North Michigan Avenue Chicago	(312) 521-7275
17 W Adams St Chicago	(312) 427-3170
565 West Jackson Boulevard Chicago	(312) 939-3111
24 South Michigan Avenue Chicago	(312) 372-4243
1723 North Halsted Street Chicago	(312) 867-0110
150 N. Dearborn Chicago	(312) 422-0150
700 E Grand Ave # 134 Chicago	(312) 644-7482

Source Code

- This assignment requires modifying the implementation of the PostgreSQL query executor. You should familiarize yourself with the following files to understand the implementation of the executor:
 - src/backend/executor/execMain.c
 - src/backend/executor/execProcnode.c
 - src/backend/executor/execScan.c
 - src/backend/executor/execTuples.c
 - src/backend/executor/nodeNestloop.c
 - src/backend/utils/fmgr/funcapi.c
 - src/include/catalog/pg_proc.h
 - src/include/executor/nodeNestloop.h
 - src/include/nodes/execnodes.h

Experiments -- Block Nested Loop Join Description

- In this assignment, you will also modify PostgreSQL to add the block nested loop optimization for nested loop joins. The pseudocode for nested loop join is:

```
for (each tuple i in outer relation) {
    for (each tuple j in inner relation) {
        if (join_condition(tuple i, tuple j) is true)
            emit (tuple i, tuple j)
        else
            continue
    }
}
```

- However, this algorithm will scan the inner table as many times as there are rows in the outer tuple. Block nested loop join can improve this by scanning the inner table for a block of outer tuples. By scanning the inner table for each BLOCK of outer tuples, fewer scans will be performed. The pseudocode for block nested loop join is:

```

for (each block of tuples B in outer relation) {
    for (each tuple j in inner relation) {
        for (each tuple i in block B) {
            if (join_condition(tuple i, tuple j) is true)
                emit (tuple i, tuple j)
            else
                continue
        }
    }
}

```

- You will implement the block nested loop join algorithm for this assignment. You will **add functionality to read a block of outer relation tuples**, and join with the inner relation.
- Your implementation of block nested loop join should not change the results of the join for different block sizes. It is only an optimization which does not alter the semantics of the join. **However, rows in the result may be out of order, and you should take that into account when testing your code.**

Tuple Tables

- The PostgreSQL executor often has to pass tuples around between operators. To make this more efficient, the executor uses something called a **TupleTable**. The tuple table consists of a collection of slots; each slot has type **TupleTableSlot**. This allows any tuple to be fully materialized only once, and the query operators can just pass along **TupleTableSlots** (which can be thought of as pointers). This makes the execution more efficient since you don't have to always copy tuples to create a new result tuple.
- In order to implement block nested loop join, you will have to create extra **TupleTableSlots** for the block of outer tuples. During the execution of the join, you will also have to copy tuples for block nested loop join, since the simple nested loop join never had to re-examine a tuple once it was read.
- You should read **execTuples.c** carefully to figure out which functions you will need in order to implement block nested loop join. In particular, you should pay attention to `ExecInitTupleSlot()` and **ExecCopySlot()**.

Rebuilding Postgres with your new code

- Use the following command to rebuild and install a modified version of postgres that uses your new modified code:

```

make
make install

```

Experiments

- We are interested in the performance gain of block nested loop join. If the query timing information is not on, turn it on with:

```

similarity-# \timing
Timing is on.

```

- When the timing is turned on, at the bottom of every query, you will see timing information like:

Time: 1234.022 ms

- You will be running the experiments with several different nested loop join block sizes. The 6 different block sizes you will experiment with are: 1, 2, 8, 64, 128, 1024
- You should report the performance of PostgreSQL using different block sizes for the block nested loop join. Run the following:
 - Run this query TWICE:

```
SELECT count(*) FROM restaurantaddress ra, restaurantphone rp WHERE ra.name = rp.name;
```

- SELECT count(*) FROM restaurantaddress ra, restaurantphone rp WHERE ra.name = rp.name;

Time: 1234.033 ms

- Record the number of shared blocks read of the SECOND run (from the statistics in the log file) and briefly explain/justify the performance numbers you saw from the different block sizes.

block_performance.txt

1 100.11

2 90.22

8 80.33

These performance numbers make sense because ...

实验流程

- 准备Linux开发环境，你可以在Windows下的虚拟机（VMware）里安装，也可以直接安装Linux系统。进行必要的配置。
- 下载Postgres安装文件源码，安装前面的介绍进行编译，安装。
- 理解Postgres的源码，可以利用附录介绍的Source Insight查看源码。
- 修改源码，并调试（可利用GDB）。
- 完成后重新编译，安装Postgres。
- 运行实验测试例子，记录运行花费时间。
- 撰写实验报告。

实验提交内容

本次实验提交内容包括：

- 代码
 - 实验中修改的代码文件（不用整个工程）。
 - 运行脚本：安装Postgres，实验测试时的命令脚本。（以自身情况而定）。
- 实验报告
 - 对实验修改的源代码进行解释。
 - 对PostgreSQL数据库内部的实现过程进行说明。（依自己的理解程度而定）
 - 实现过程说明及实验结果性能比较。（可选）

不要直接写死一个for循环，要搞什么pipeline

避坑指南

- 提示缺少 readline

redhat 系列下这个软件包叫 readline-devel ; ubuntu 下叫readline-dev 细分又分为libreadline5-dev 和 libreadline6-dev

如果使用Ubuntu则可使用：

```
$ sudo apt-get install libreadline6-dev
```

- 提示缺少 zlib
与上同理

```
$ sudo apt-get install zlib1g  
$ sudo apt-get install zlib1g.dev
```

- 第一次修改过源码后需要configure后再执行make和make install
- 建议使用gdb调试，比如利用断点进行代码追踪，利用栈帧链打印出函数调用关系，对于阅读源码也有非常大的帮助。
- 使用Source Insight 阅读源码。Source Insight是一个面向项目开发的程序编辑器和代码浏览器，它拥有内置的对C/C++，C#和Java等程序的分析。Source Insight能分析源代码并在用户工作的同时动态维护它自己的符号数据库，并自动为用户显示有用的上下文信息。Source Insight不仅仅是一个强大的程序编辑器，它还能显示reference trees，class inheritance diagrams和call trees。Source Insight提供了最快速的对源代码的导航和任何程序编辑器的源信息。

可以使用gdb来做
调试，不一定要使
用Source Insight