

基于MiniOB的数据库管理系统内核实现实验手册 v0.2.3

- 1 实验准备
 - 1.1 了解 MiniOB
 - 1.2 了解实验平台
 - 1.3 实验要求
 - 1.3.1 组队要求
 - 1.3.2 实验代码及实验报告要求
 - 1.3.3 实验安排
- 2 实验一（10 分）
 - 2.1 管理自己的 MiniOB 代码
 - 2.1.1 在 Gitee 上创建自己的 private 仓库
 - 2.1.2 下载并安装 git 工具（以下以 Windows 为例，Linux 环境类似）
 - 2.1.3 将 MiniOB 源码下载到本地
 - 2.1.4 将本地的 MiniOB 代码 push 到自己在 Gitee 上的仓库
 - 2.1.5 赋权官方测试账号和山东大学数据库实验课程测试账号
 - 2.1.6 Git 开发常用命令
 - 2.1.7 团队开发建议
 - 2.2 掌握利用 Visual Studio Code + Docker 开发 MiniOB
 - 2.2.1 搭建 Docker 环境
 - 2.2.2 使用 Visual Studio Code 调试代码
 - 2.3 提测代码
- 3 实验二（20 分）
 - 3.1 查询元数据校验实验（10 分）
 - 3.2 删除表实验（10 分）
- 4 实验三（20 分）
 - 4.1 数据更新实验（10 分）
 - 4.2 多表笛卡尔积关联查询实验（10 分）
- 5 实验四（10 分）
 - 5.1 增加 date 字段实验（10 分）
- 6 实验五（20 分）
 - 6.1 聚合函数实验（10 分）
 - 6.2 插入多行数据实验（10 分）
- 7 参考资料

1 实验准备

1.1 了解 MiniOB

MiniOB 设计的目标是让不熟悉数据库设计和实现的同学能够快速的了解与深入学习数据库内核，期望通过 minioB 相关训练之后，能够对各个数据库内核模块的功能与它们之间的关联有所了解，并能够在使用时，设计出高效的 SQL。面向的对象主要是在校学生，并且诸多模块做了简化，比如不考虑并发操作。

minioB 作为一个具有“基本”功能的数据库，包含了需要的基本功能模块。包括：

- 网络模块：负责与客户端交互，收发客户端请求与应答；
- SQL 解析：将用户输入的 SQL 语句解析成语法树；
- 执行计划缓存：执行计划缓存模块会将该 SQL 第一次生成的执行计划缓存在内存中，后续的执行可以反复执行这个计划，避免了重复查询优化的过程（未实现）。
- 语义解析模块：将生成的语法树，转换成数据库内部数据结构（未实现）；
- 查询缓存：将执行的查询结果缓存在内存中，下次查询时，可以直接返回（未实现）；
- 查询优化：根据一定规则和统计数据，调整/重写语法树。当前实现为空，留作实验题目；
- 计划执行：根据语法树描述，执行并生成结果；
- 会话管理：管理用户连接、调整某个连接的参数；
- 元数据管理：记录当前的数据库、表、字段和索引元数据信息；
- 客户端：作为测试工具，接收用户请求，向服务端发起请求。

对 minioB 架构感兴趣的同学可以参考 [minioB代码架构框架设计和说明](#)。

数据库实现的基础知识相关教案请查阅参考资料《数据库系统概念》（原书第 7 版、本科教学版）第 10 章至第 14 章内容，[《数据库管理系统实现基础讲义》](#)，以及 [OceanBase 数据库基础概念](#)。

1.2 了解实验平台

本实验采用 OceanBase 训练营平台，OceanBase 训练营是 OceanBase 为帮助开发者如何更好的学习数据库基本知识而设计的练习平台，帮助开发者从 0 到 1 理解数据库底层原理和实现方式，参与实践而开发的练习平台。

训练营地址：<https://open.oceanbase.com/train>

训练营的使用说明：<https://ask.oceanbase.com/t/topic/35600372>

1.3 实验要求

1.3.1 组队要求

- 1-3 人一组，选出组长。
- 组长每周记录每人完成的任务、进度。
- 第 12 周汇总提交报告、代码。

1.3.2 实验代码及实验报告要求

- 提交的代码及实验报告会进行查重判别。

- 每个小组只需提交 1 份实验报告，但必须在实验报告中注明每个人的贡献情况。

1.3.3 实验安排

请大家按时完成实验一至实验五，各个实验的时间及分数安排如下（参加大赛的队伍可以不参考此表）：

项目	完成时间	分数
实验一	第 9 周周日	10 分
实验二	第 9 周周日	20 分
实验三	第 9 周周日	20 分
实验四	第 12 周周日	10 分
实验五	第 12 周周日	20 分
实验报告	第 12 周周日	20 分
满分		100 分

2 实验一（10 分）

本实验主要用于熟悉训练营平台的使用流程和 MiniOB 开发环境的搭建，为后续实验奠定基础，**本实验必须全部完成才能进行后续实验。**

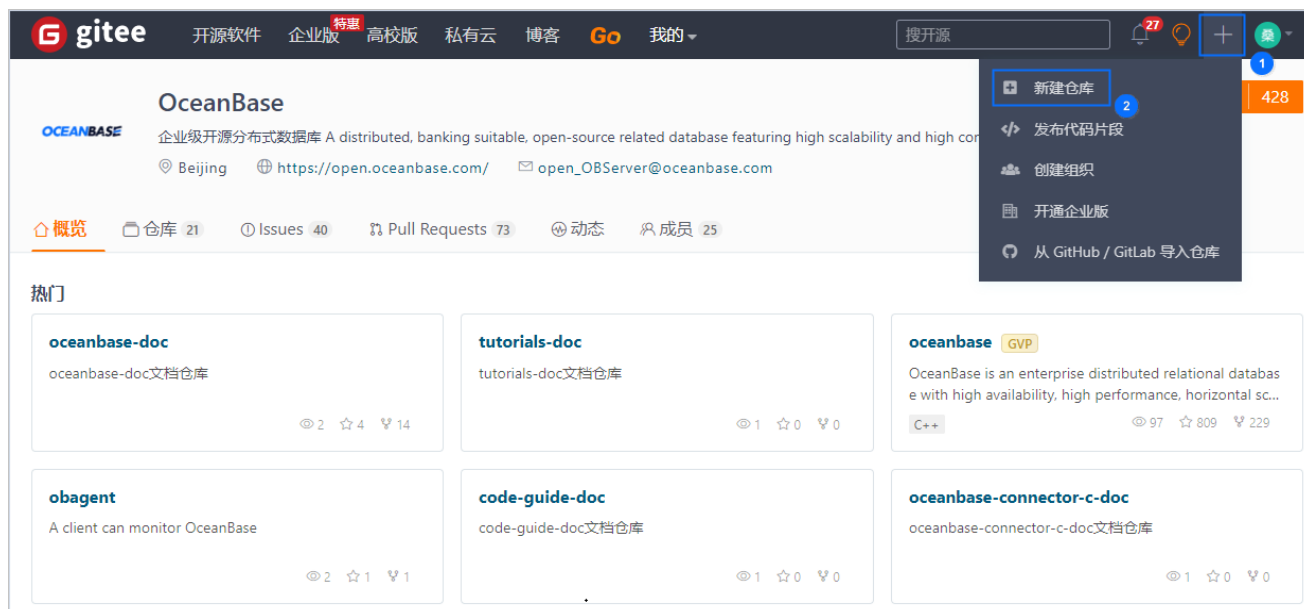
2.1 管理自己的 MiniOB 代码

2.1.1 在 Gitee 上创建自己的 private 仓库

训练营平台使用 Gitee 来管理 MiniOB 代码，因此需要首先在 Gitee 上创建自己的 private 仓库，用于管理自己的代码。

注册 Gitee 账号，Gitee 官网地址：<https://gitee.com>。

登录 Gitee 平台，选择新建仓库。



输入仓库信息，单击 创建。设置为私有仓库后其他人无法查看到你的代码。

新建仓库

在其他网站已经有仓库了吗？[点击导入](#)

仓库名称 *

MiniOB

1

归属

泉

路径 *

MiniOB

2

仓库地址：<https://gitee.com/MiniOB>

仓库介绍

0/100

用简短的语言来描述一下吧

☐ 开源（所有人可见）?

☒ 私有（仅仓库成员可见）

☐ 企业内部开源（仅企业成员可见）?

3

☐ 初始化仓库（设置语言、.gitignore、开源许可证）

☐ 设置模板（添加 README、Issue、Pull Request 模板文件）

☐ 选择分支模型（仓库创建后将根据所选模型创建分支）

创建

4

2.1.2 下载并安装 git 工具（以下以 Windows 为例，Linux 环境类似）

git与SVN、CSV等都是常用的，团队开发工具，能够记录代码修改历史记录，多人协作。

git 已经有很久的历史，人们也摸索出了一套开发流程建议，在多人团队共同开发代码时，遵守这套流程规范，能够减少很多错误。

本实验使用开源代码管理工具 git 进行代码管理。下载地址：<https://git-scm.com/download/win>

2.1.3 将 MiniOB 源码下载到本地

MiniOB 源码地址：<https://github.com/oceanbase/minioh>。

定位到存放代码的目录，使用 **Windows cmd 或者 powershell 环境** 命令行键入以下命令，下载源代码：

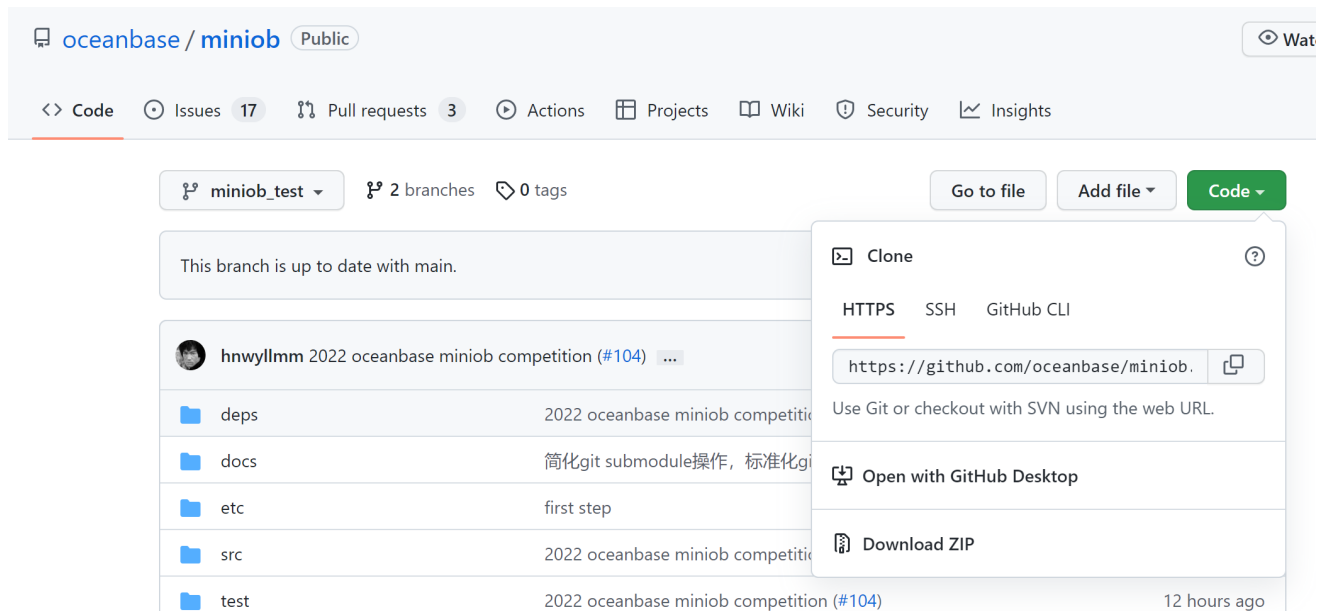
SHELL

```
d: # 切换到D盘
cd ./Download # 以Download目录为例（代码存放的目录很重要，接下来很多操作都在此处进行，应建立单独目录存放代码）
```

```
git clone https://github.com/oceanbase/miniob # 将代码拉到本地（默认Download
为当前目录）

cd miniob # 此时代码目录为 d:/Download/miniob，接下来会用到
rm -r -fo .git # 删除 miniob 下的 .git 目录
```

或者通过 Download ZIP 直接在 GitHub 上下载代码压缩包。



2.1.4 将本地的 MiniOB 代码 push 到自己在 Gitee 上的仓库

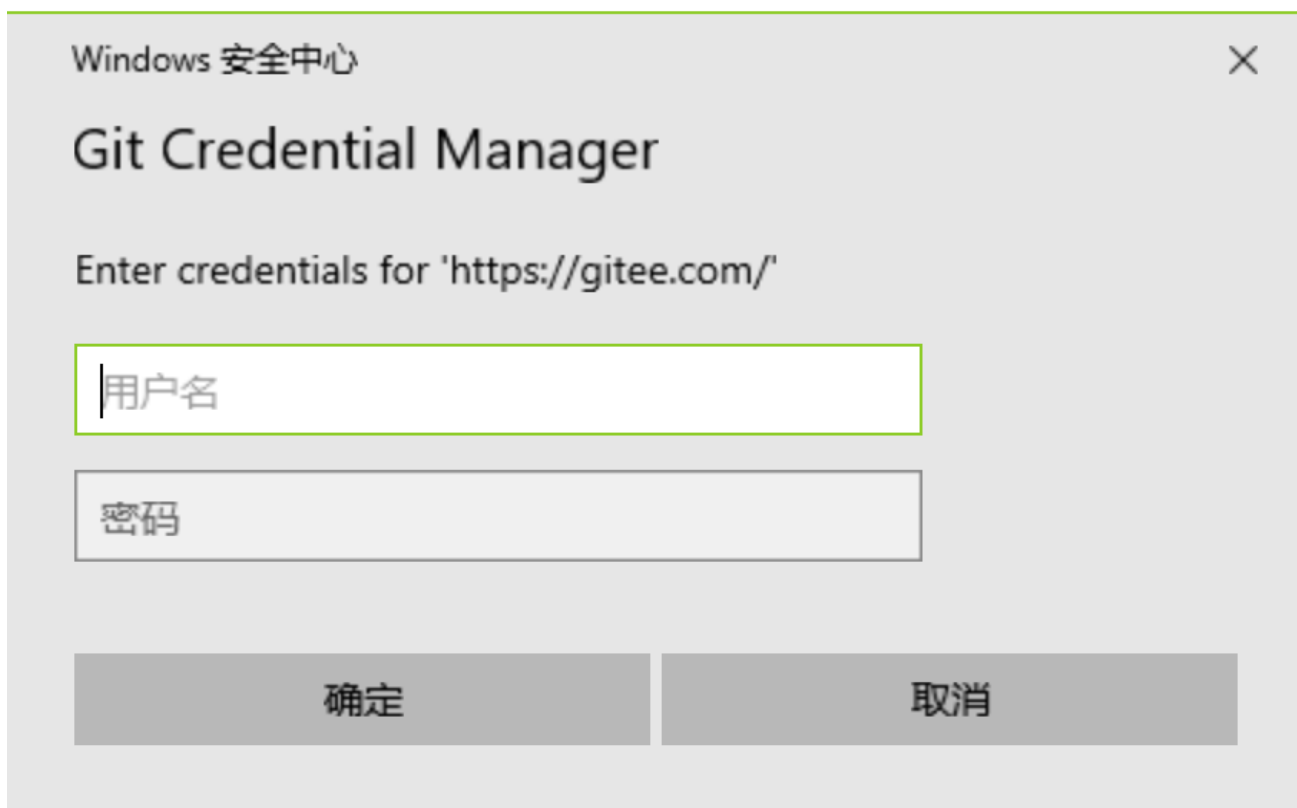
使用 **Windows cmd 或者 powershell 环境** 命令行键入以下命令：

```
# 重新初始化 git 信息，并将代码提交到本地仓库
cd miniob

git init
git add .
git commit -m 'init' # 提交所有代码到本地仓库

# 将代码推送到远程仓库
git remote add origin https://gitee.com/xxx/miniob.git # 注意替换命令中xxx为自
己在Gitee上的个人空间地址
git branch -M main
git push -u origin main
```

一般情况下 git push 命令会提示输入用户名密码：



此时，登陆自己仓库页面就可以看到代码已经推送到 Gitee 上。

2.1.5 赋权官方测试账号和山东大学数据库实验课程测试账号

对于私有仓库，默认情况下其他人看不到，同样 OceanBase 测试后台也无法拉取到代码，这时想要提交自己的代码给训练营平台测试，需要先给 OceanBase 的官方测试账号和山东大学数据库实验课程测试账号增加一个权限。

OceanBase 官方测试账号： `oceanbase-ce-game-test`

山东大学数据库实验课程测试账号： `SDU-database`

首先在Gitee上打开自己的仓库，然后按照如下顺序操作即可。

- 选择**管理 > 仓库成员管理 > 观察者**



- 选择**直接添加**，搜索 `oceanbase-ce-game-test` 和 `SDU-database`。

邀请用户

仓库成员配额说明

个人私有仓库最多支持 5 人协作（如个人拥有多个私有仓库，所有协作人数总计不得超过 5 人）

链接邀请

直接添加

通过仓库邀请成员

直接添加 Gitee 用户

权限

观察者

Gitee 用户

oceanbase-ce-game-test x

添加

添加完成后，单击**提交**。

2.1.6 Git 开发常用命令

SHELL

1. 查看当前分支

```
git branch # 查看本地分支
```

```
git branch -a # 查看所有分支，包括远程分支
```

2. 创建分支

```
git checkout -b 'your branch name' # 创建分支
```

```
git branch -d 'your branch name' # 删除一个分支
```

3. 切换分支

```
git checkout 'branch name' # 切换分支
```

4. 提交代码

```
git add 'the files or directories you want to commit' # 添加想要提交的文件或文件夹
```

```
# 这一步也可以用 git add . 添加当前目录
```

```
# 提交到本地仓库
```

```
git commit -m 'commit message' # -m 中是提交代码的消息，建议写有意义的信息，方便后面查找
```

5. 推送代码到远程仓库

```
git push # 可以将多次提交，一次性 push 到远程仓库
```

6. 合并代码

```
git merge feature/update # 假设当前处于分支 develop 下，会将 feature/update 分支的修改，merge 到 develop 分支
```

```
# 7.临时修改另一个分支的代码
# 有时候，正在开发一个新功能时，突然来了一个紧急 BUG，这时候需要切换到另一个分支去开发
# 这时可以先把当前的代码提交上去，然后切换分支。
# 或者也可以这样：
git stash # 将当前的修改保存起来
git checkout main # 切换到主分支，或者修复 BUG 的分支
git checkout -b fix/xxx # 创建一个新分支，用于修复问题
# 修改完成后，merge 到 main 分支
# 然后，继续我们的功能开发
git checkout feature/update # 假设我们最开始就是在这个分支上
git stash pop
```

另外还可以查阅 [git 简明教程 https://www.runoob.com/manual/git-guide/](https://www.runoob.com/manual/git-guide/)。

2.1.7 团队开发建议

本次实验，最多支持 3 人一组，而且组内交流比较容易，所以这里给出一个简单的开发流程建议。

(1) git 分支

我们在git上开发时，都是在某个分支上进行操作，修改、提交代码。一个分支的代码，修改后不会影响其它分支。除非明确执行合并。

分支的一个好处就是，多个人在开发新功能，或者修改BUG时，都在自己的分支上修改，修改完成，测试正确后，再合并到共同的分支。

分支可以非常容易的创建和销毁。当我们开发一个新功能时，可以基于某一个分支(比如 main/master)，创建一个分支 feature/test，在新的分支上修改代码，完成后，合并到 main/master分支上。

(2) 本地仓库 vs. 远程仓库

大家在开发时，首先会从 github 或者Gitee上 clone 代码到本地机器，那么这时本地就会有一个仓库。

简单来说，github 或者Gitee上面的仓库，是远程仓库，是所有人都能够看到的仓库。

拉到本地后，本地上也会有一个仓库，称为本地仓库(local)。我们在本地上做的任何修改，都不会直接影响到远程仓库，除非我们执行push操作。

(3) 仓库 & 分支

不管是远程仓库，还是本地仓库，都可以创建分支。也可以关联本地分支到远程分支，这样本地开发完成后，执行push命令，就可以将本地修改同步到远程。

(4) 管理仓库的分支

在仓库创建出来的时候，就会有一个默认分支，一般是 main 或者 master。训练营后台测试，每次会拉取默认分支进行编译测试。

因此，建议将main/master 作为“稳定”分支，就是所有的修改，都测试完成后，再merge到 main/master。

后续，所有的默认分支都称为 **main**。

如果是小组内团队开发，建议创建一个 **dev** 或 **develop** 分支（后面使用 **develop**，作为共同开发的分支，大家修改的新功能，先 merge 到 develop，在 develop 测试通过后，再 merge 到 **main**。

对于团队中的成员，可以开启一个自己的分支，或者以功能名称命名的分支，比如 **feature/update**。

(5) 开发流程

1.对于实验小组团队

1. 创建主分支 **main**
2. 创建开发分支 **develop**

团队准备提交代码到主分支时，执行：

SHELL

```
# 切换到开发分支
git checkout develop
# 拉取最新代码
git pull
# 执行一系列测试，没有问题的时候
# 切换到主分支，并更新到最新
git checkout main
git pull
# 合并开发分支
git merge develop
# 推送到远程仓库，这时官方测试人员，就可以拉到最新的代码
git push
```

2.对于个人

1. 拉取团队仓库代码(git clone)
2. 切换到 **develop** 分支(git checkout develop)
3. 创建新分支，用于开发新的功能，比如 **git checkout -b feature/update**
4. 功能开发完成，测试通过后，提交代码到本地仓库 **git add . && git commit -m 'your commit message'**
5. 同步代码到开发分支 **develop**

```
# 切换到develop分支
git checkout develop

# 同步远程分支代码
git pull

# 合并 feature/update 到develop
git merge feature/update

# 合并完成后，推送到远程仓库
git push

# 然后删除自己的feature分支
git branch -d feature/update
```

2.2 掌握利用 Visual Studio Code + Docker 开发 MiniOB

从实验一到实验五，建议使用 **vscode + Docker** 进行开发。

2.2.1 搭建 Docker 环境

Docker 是一个开源的应用容器引擎，基于 [Go 语言](#) 并遵从 Apache2.0 协议开源。Docker 可以让开发者打包他们的应用以及依赖包到一个轻量级、可移植的容器中，然后发布到任何机器上，也可以实现虚拟化。

(1) 安装docker

请参考如下教程安装Docker，[如何在Windows上使用Docker开发miniob](#)

(2) 挂载本地目录到 docker 镜像中

为了更加方便安全和优雅的进行后续开发，通常我们将代码保存至自己本地电脑上，**Docker 中运行的容器仅作为代码编译的环境**。开发时，代码存放于本地，如果容器出现什么异常，自己最重要的数据不会丢失。这里使用 **mount** 挂载的方式，将一个本地目录映射到 Docker 容器中。

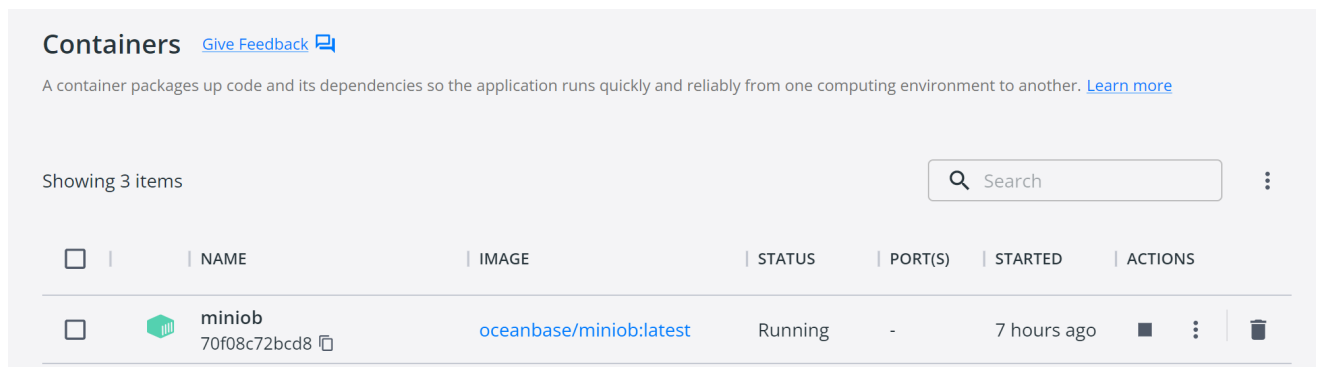
安装完毕之后，可以使用 **Windows cmd 或者 powershell 环境** 执行如下命令，下载并运行 miniob 镜像。

```
docker run -d --name miniob --privileged -v $PWD/miniob:/root/miniob
oceanbase/miniob
```

其中 **-d** 参数是后台运行容器，并返回容器 ID，**--name miniob** 参数是重命名容器名称为 miniob，这里也可以用其他名称命名，**--privileged** 参数是为了方便在容器中进行调试，**-v \$PWD/miniob:/root/miniob** 参数是将本地的 **\$PWD/miniob** 目录映射到容器中的 **/root/miniob** 目录，这里假设 **\$PWD/miniob** 是你本地代码存放的位置，需要按照实际目录来调

整。本手册默认代码存放位置为 `d:\Download\miniob` , `$PWD` 应为 `d:\Download` , `oceanbase/miniob` 参数就是镜像的名称, Docker 会自动从云端拉取该镜像的最新版本。

此时可以看到 Docker Desktop 中已经有一个名为 miniob 的容器在运行。



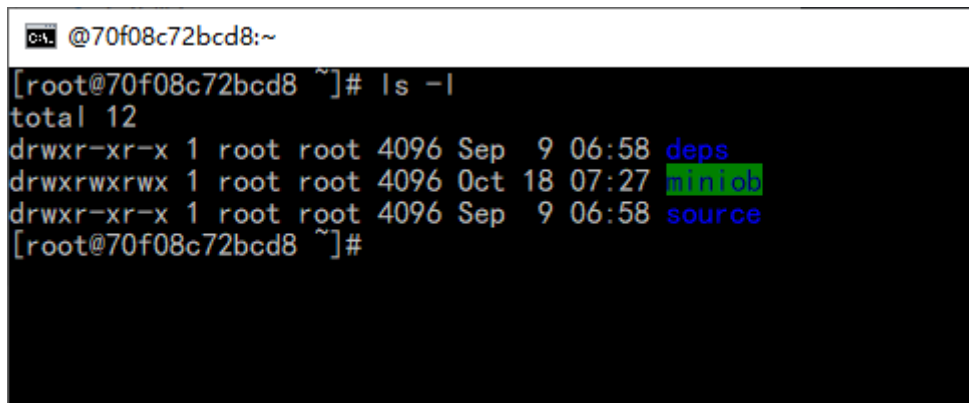
(3) 使用镜像内的 bash 命令行

运行下面的命令可以进入到容器并进行开发, 这个命令可以在终端上执行多次, 开启多个窗口方便操作。

```
docker exec -it miniob bash
```

SHELL

此时, 我们应该可以看到本地目录下, 是这样目录结构。



其中 `/deps` 和 `/source` 是该镜像中自带的源码目录, 而反色显示的 `/miniob` 是用挂载方式映射到容器中的本地目录的软链接, 在容器可以直接使用 `cd` 命令进入到 `miniob` 目录中查看文件。

(4) 创建 ./build 目录存放编译产生的文件

进入到该目录中, 为了存放编译之后生成的文件, 需要使用 `mkdir ./build` 命令创建 build 目录。

```
CS: @70f08c72bcd8:~/miniob

[root@70f08c72bcd8 ~]# cd miniob/
[root@70f08c72bcd8 miniob]# ls -l
total 24
-rwxrwxrwx 1 root root 4528 Oct 18 04:02 CMakeLists.txt
-rwxrwxrwx 1 root root 7031 Oct 18 04:02 COPYING
-rwxrwxrwx 1 root root 2749 Oct 18 04:02 Dockerfile
-rwxrwxrwx 1 root root 1987 Oct 18 06:27 README.md
drwxr-xr-x 1 root root 4096 Oct 18 09:33 build
drwxrwxrwx 1 root root 4096 Oct 18 04:02 deps
drwxrwxrwx 1 root root 4096 Oct 18 04:02 docs
drwxrwxrwx 1 root root 4096 Oct 18 04:02 etc
drwxrwxrwx 1 root root 4096 Oct 18 04:02 src
drwxrwxrwx 1 root root 4096 Oct 18 04:02 test
drwxrwxrwx 1 root root 4096 Oct 18 04:02 unittest
[root@70f08c72bcd8 miniob]#
```

此时就可以通过 `cmake`、`make` 等终端命令编译代码了，但是在具体使用时会很不方便，因为每次修改代码之后，都需要输入编译的一系列指令，比较繁琐。所以接下来我们利用 `vscode`，使用图形化界面和丰富的扩展插件调试代码。

(5) 使用建议

启动镜像时忘记增加一些参数，但是又不想重新创建怎么办？

有时候启动镜像时（`docker run`），忘记增加一些参数，比如 `privileged`，但是又在容器中做了很多操作，对环境做了调整，不希望启动全新的容器，可以这么做：

- 提交当前的容器，创建新的镜像

SHELL

```
docker commit -m 'commit message' -t miniob:vx miniob
docker stop miniob
docker rm miniob
```

- 启动新的镜像

SHELL

```
docker run -d --name miniob xxxx miniob:vx --privileged -v
$PWD/miniob:/root/miniob oceanbase/miniob
```

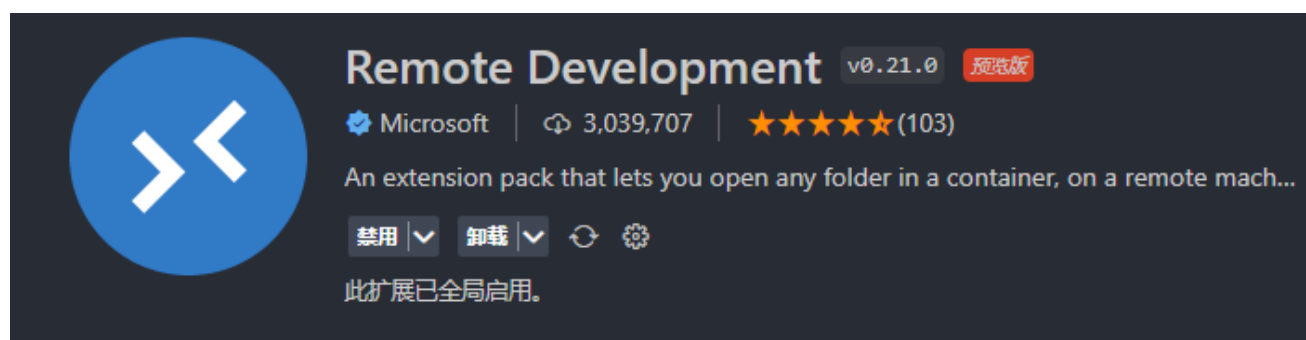
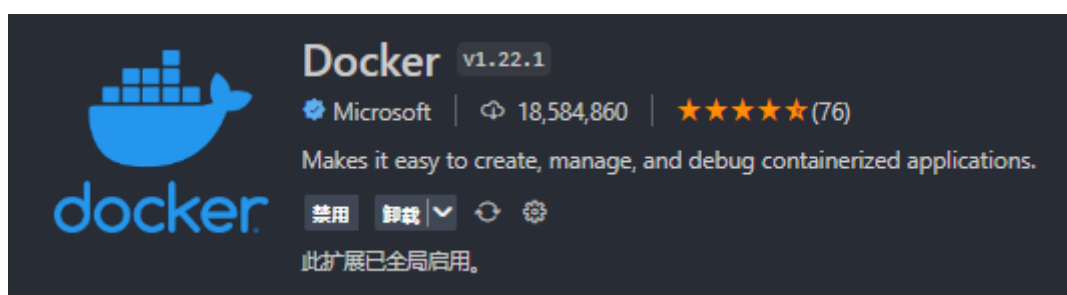
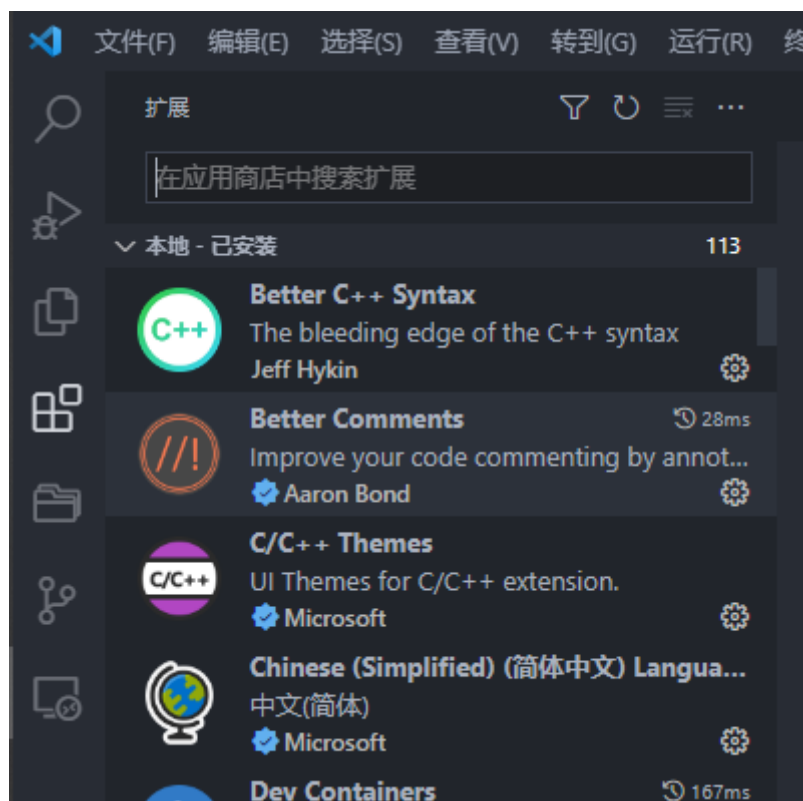
注：参数中 `miniob: vx` 中的 `vx` 是一个版本号，你可以自己设置，每次使用不同的名字即可。

2.2.2 使用 Visual Studio Code 调试代码

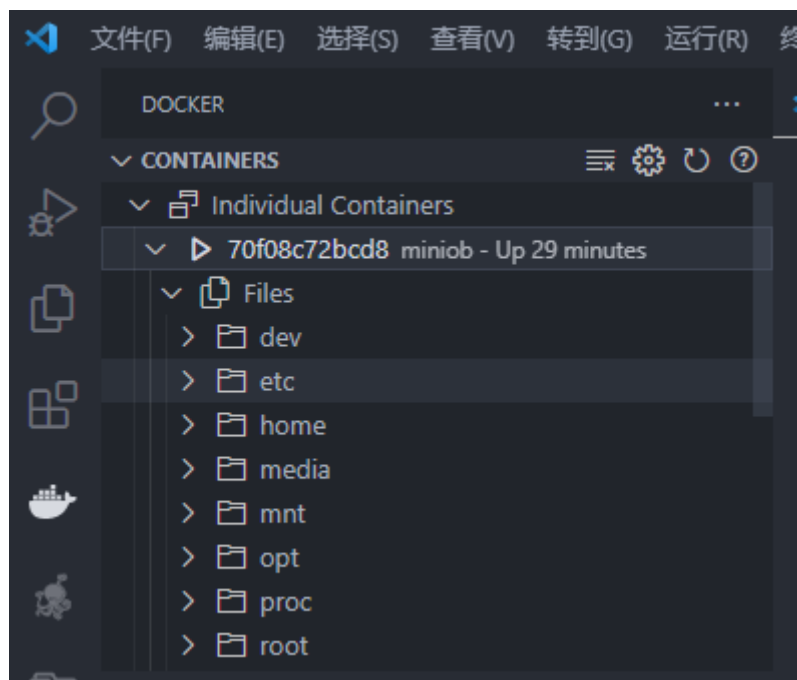
建议使用集成开发环境（IDE）进行数据库系统代码调试，这里以最为常用的 `Visual Studio Code`（简称 `vscode`）为例进行演示。下载地址：<https://code.visualstudio.com/Download>，参考教程：<https://code.visualstudio.com/learn>。

(1) 本地安装扩展

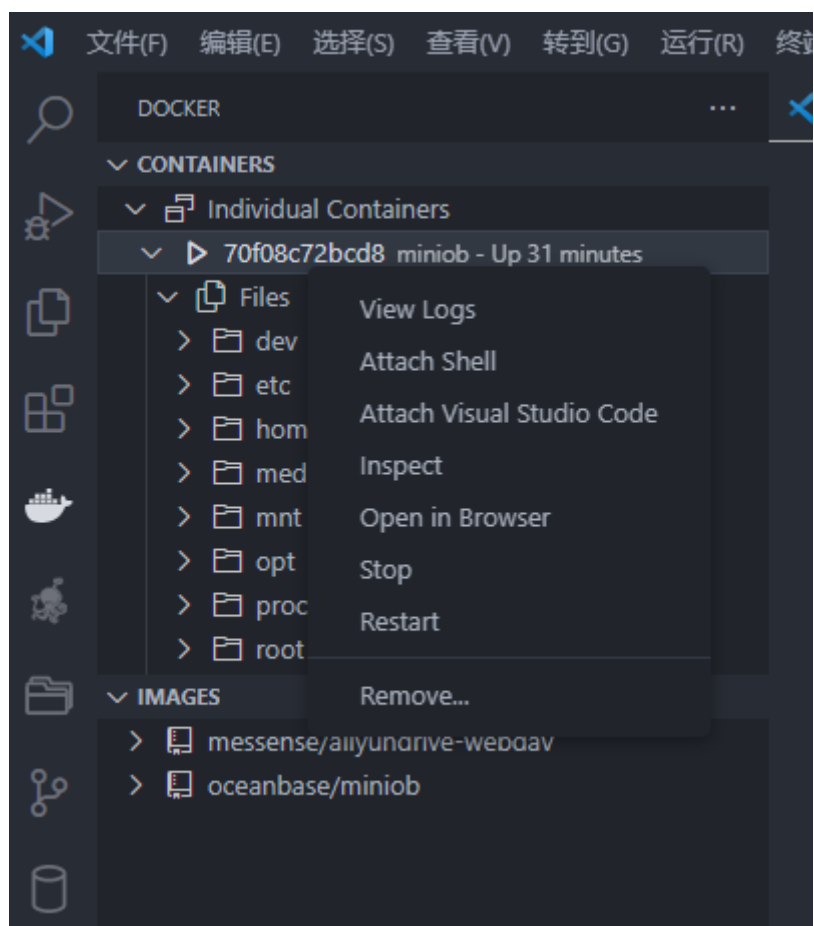
启动 vscode 后，在“扩展”中安装 Docker，C/C++，# C/C++ Extension Pack，Remote Development 等插件，安装方法是在“扩展”-“搜索框”中搜索相应名称，选择“安装”即可。



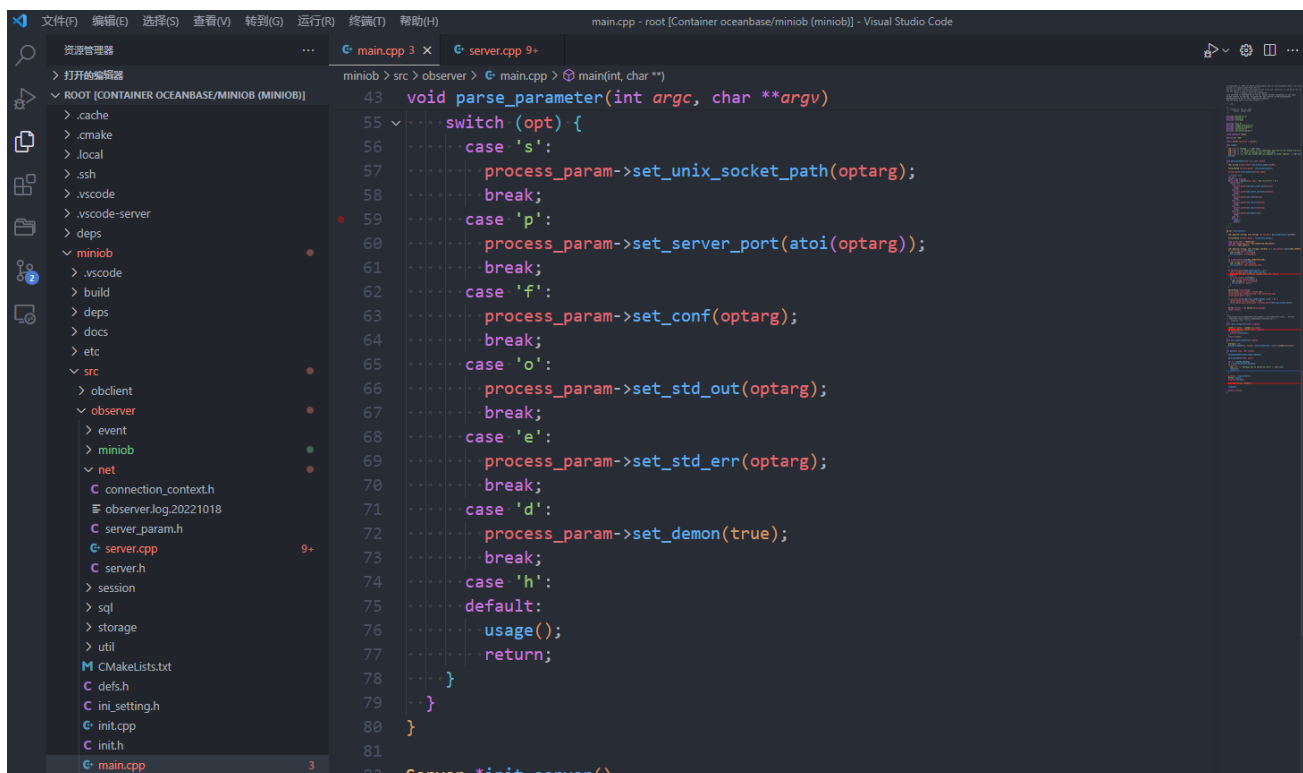
如果成功安装了 docker 环境，此时就可以看到运行的 miniob 容器以及文件列表。



在容器上右键选择 Attach Visual Studio Code, 会启动一个新窗口。

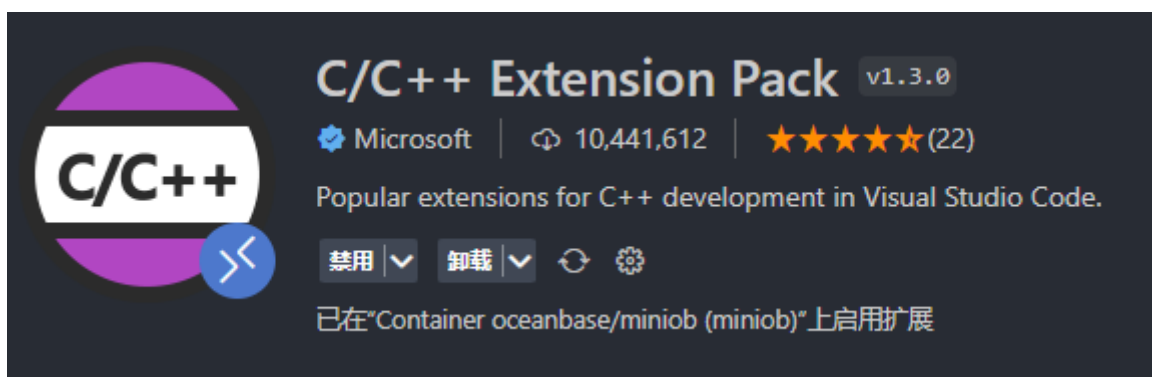


此时，我们就利用 vscode 进入到 docker 镜像中，可以开发并调试代码了。从文件列表中，我们可以直接打开对应的源码文件进行编辑。



(2) 容器内安装扩展

为了在 vscode 中编译代码，我们需要在容器内的“扩展”中安装 C/C++，# C/C++ Extension Pack 等插件，用于编译 C++ 程序。



注意，安装成功后，插件下面会提示“已在 Container 中启动扩展”。以下我们所有操作都是在容器内进行。

(3) 配置编译和运行命令 (vscode 任务)

在容器内/root 目录下创建 .vscode 目录（注意，以 . 为前缀的目录是隐藏目录），用于存放对本项目的各种配置文件。

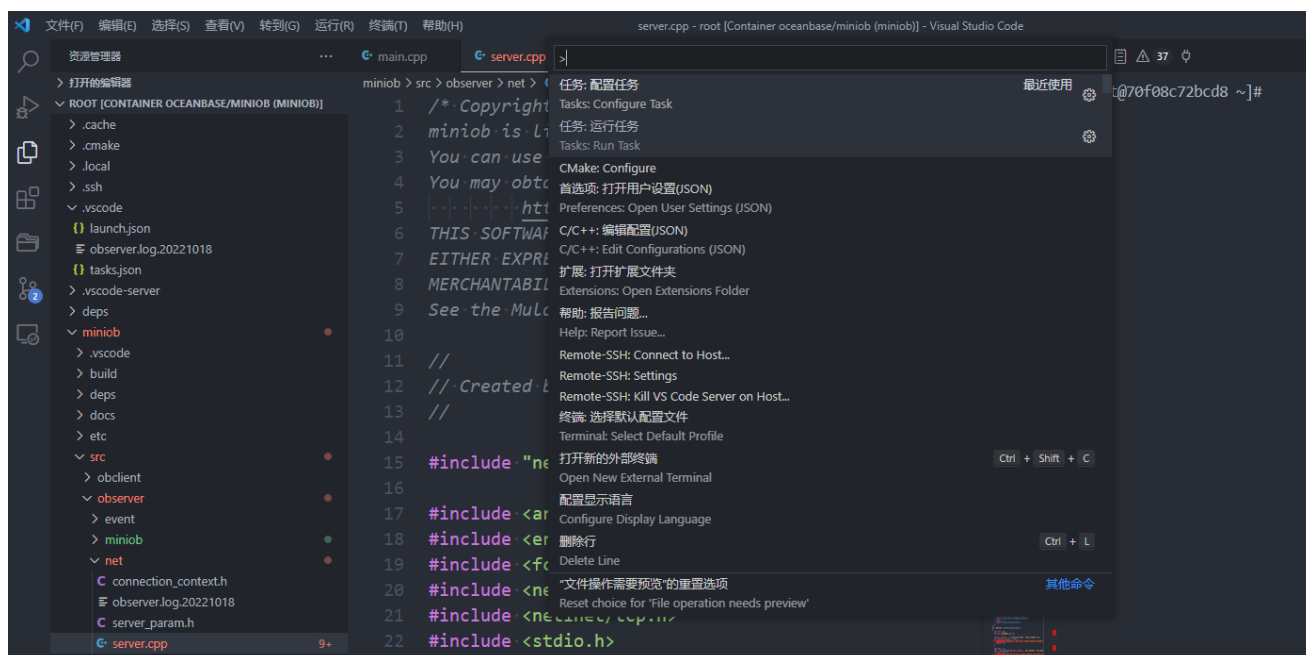
在 .vscode 目录中，创建 task.json 文件，将以下配置命令拷贝粘贴并保存。

```
{
  "options": {
    "cwd": "${workspaceFolder}/miniob/build"
  },
  "tasks": [
    {
      "label": "cmake",
      "command": "cmake",
      "args": [
        "-DDEBUG=ON",
        ".."
      ]
    },
    {
      "label": "make",
      "command": "make"
    },
    {
      "label": "CMake Build",
      "dependsOn": [
        "cmake",
        "make"
      ]
    },
    {
      "label": "Get pid of observer",
      "type": "shell",
      "command": "ps aux | grep observer"
    },
    {
      "label": "Run observer",
      "type": "shell",
      "command": "/root/miniob/build/bin/observer -s miniob-test.sock -f /root/miniob/etc/observer.ini"
    },
    {
      "label": "Run obclient",
      "type": "shell",
      "command": "/root/miniob/build/bin/obclient -s miniob-test.sock"
    }
  ],
}
```

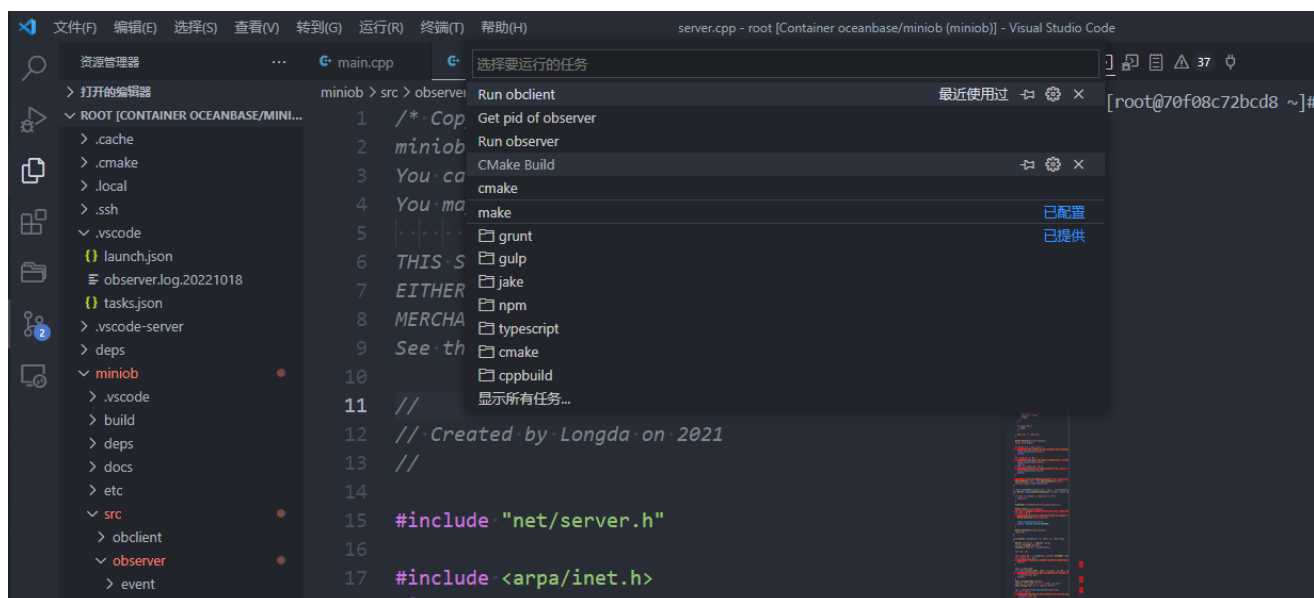


```
"version": "2.0.0"
}
```

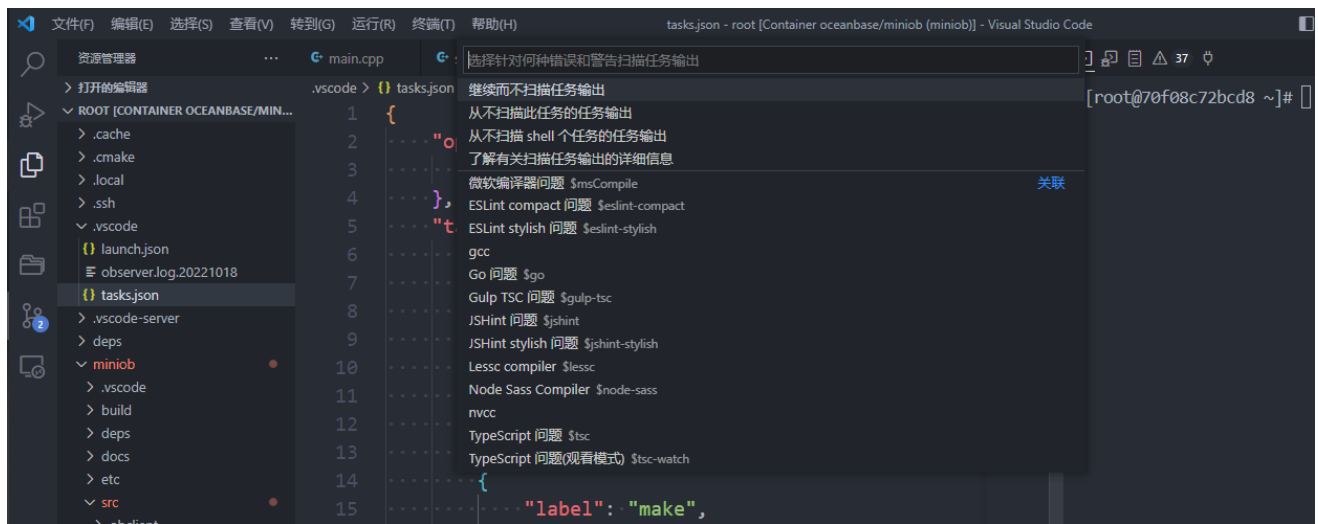
完成以上步骤后，在本项目中就配置了 5 个快捷任务，使用 **Ctrl + Shift + P** 快捷键调出快捷命令，选择“任务：运行任务”。



此时就可以看到刚刚配置好的任务列表，其中 **Cmake Build** 任务的前置任务分别是 **cmake** 和 **make**，依次在 miniob 目录下编译并链接源文件。



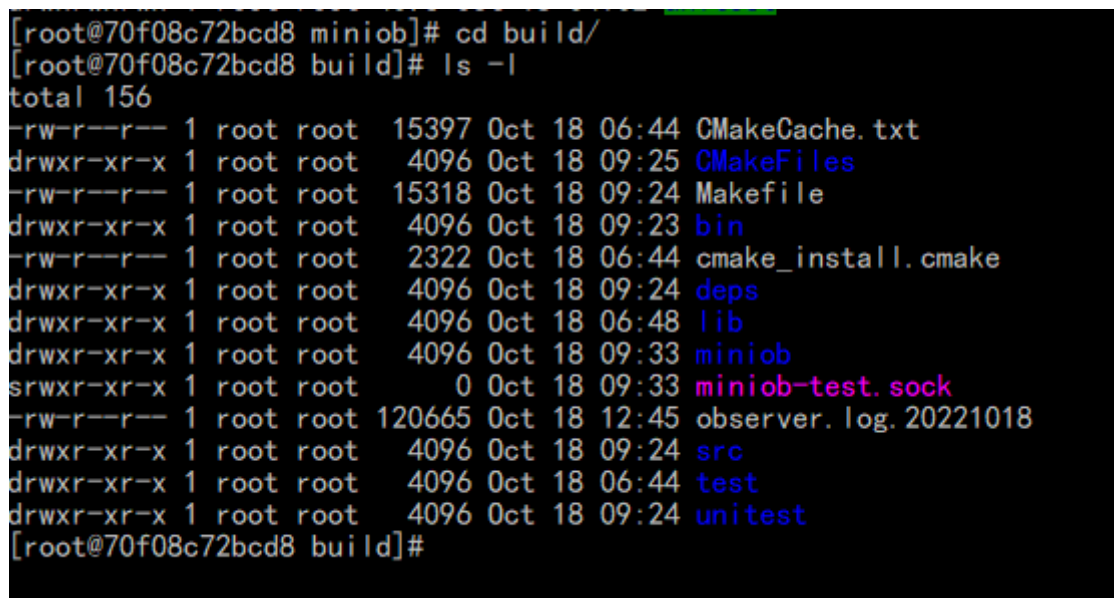
运行 **Cmake Build** 任务，弹出的选项中选择“继续而不扫描任务输出”。



等待一段时间，如果代码无误，即可编译成功。



执行完以上任务，可以在 build 目录下看到生成的文件和目录。其中可执行文件在 bin 目录下。



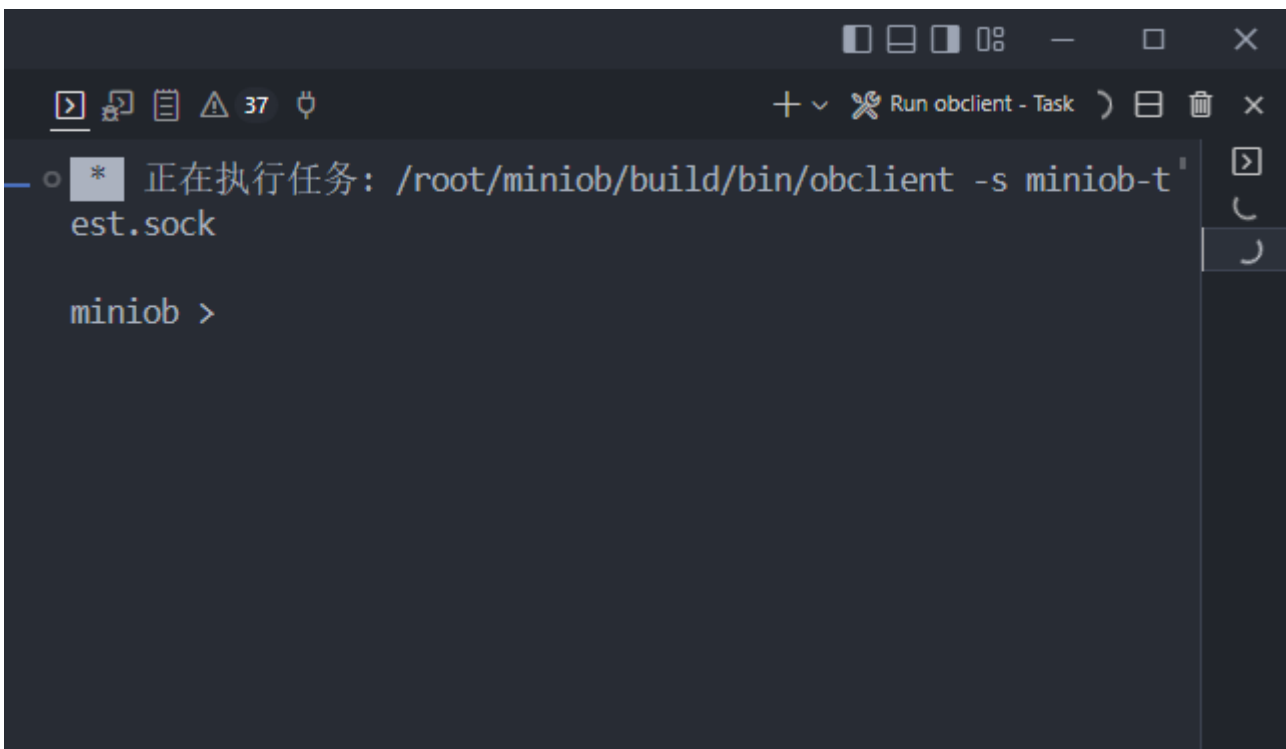
编译好的文件就可以运行调试，使用 **Run observer** 就可以运行 miniob 的 observer 命令，启动一个 miniob 数据库服务器。启动成功后，有如下提示。



```
正在执行任务: /root/miniob/build/bin/observer -s miniob-test.sock -f /root/miniob/etc/observer.ini

Successfully load /root/miniob/etc/observer.ini
```

使用 **Run obclient** 可以运行 miniob 的 obclient 命令，在一个新的终端启动数据库客户端，客户端可以直接连接刚启动的 miniob 数据库。



```
正在执行任务: /root/miniob/build/bin/obclient -s miniob-test.sock

miniob >
```

注 1: 要先启动 observer，再启动 obclient 才能成功连接。

注 2: observer 的启动过程是通过建立一个 miniob-test.sock 的套接字软链接以及通过指定 ../../etc/observer.ini 配置文件启动的。

此时，我们已经可以运行由源代码编译完成的数据库程序了。

通过 obclient 登陆数据库，完成基本的创建表、查看表的测试语句。

SQL

```
show tables;

desc `table name`;

create table `table name` (`column name` `column type`, ...);

create index `index name` on `table` (`column`);

insert into `table` values(`value1`, `value2`);

update `table` set column=value [where `column`=`value`];

delete from `table` [where `column`=`value`];

select [ * | `columns` ] from `table`;
```

(3) 配置 Debug 环境

vscode 最强大的工具应该是其 debug 工具，可以方便使用图形界面的形式展示 debug 过程中的各种参数变量，提高开发效率。

首先，需要在 .vscode 目录中创建 launch.json 文件，并将以下配置信息拷贝粘贴并保存。

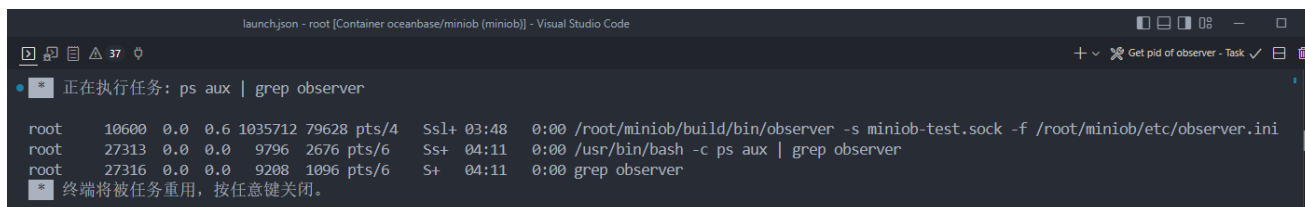
```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "(gdb) Attach",
      "type": "cppdbg",
      "request": "attach",
      "program": "${workspaceFolder}/miniob/build/bin/observer",
      "MIMode": "gdb",
      "setupCommands": [
        {
          "description": "为 gdb 启用整齐打印",
          "text": "-enable-pretty-printing",
          "ignoreFailures": true
        },
        {
          "description": "将反汇编风格设置为 Intel",
          "text": "-gdb-set disassembly-flavor intel",

```

```
        "ignoreFailures": true
      }
    ],
  },
]
}
```

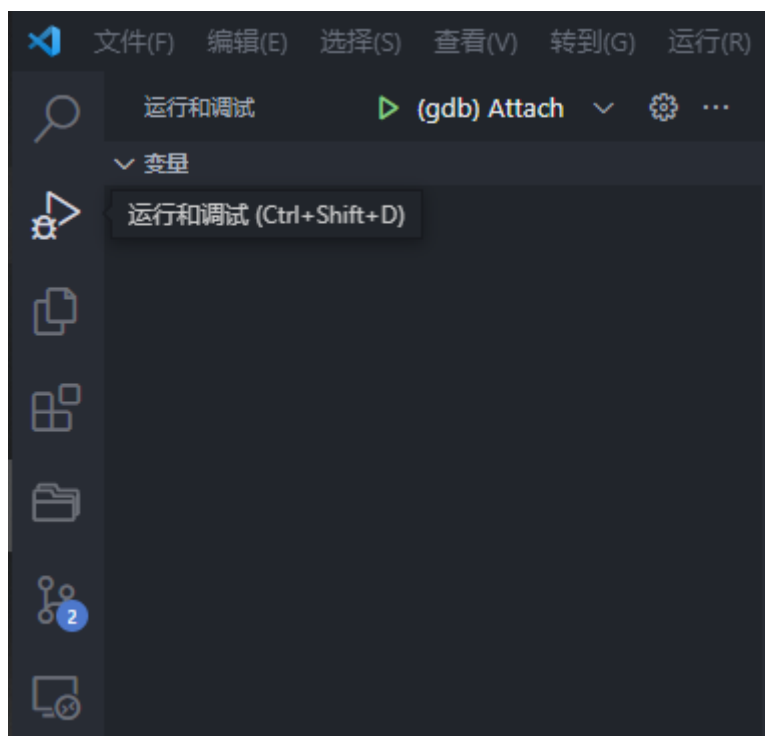
运行和调试的 Attach 功能可以跟踪一个运行中的进程，从而对其进行调试。**数据库系统的调试过程往往是通过运行一个编译好的数据库程序，然后利用 Attach 方法跟踪其进程并在指定代码位置设置断点，从断点处逐步调试编写好的程序。**

所以使用 Attach 功能之前，我们需要获得运行中的 observer 的进程号，此时继续使用上文提到的快捷任务。**Ctrl + Shift + P** 调出快捷任务，运行 **Get pid of observer**，获得运行中的 observer 进程号。一般情况下会得到多个结果，选择运行数据库服务的那条命令对应的行（截图中为第一行），找到进程号为 **10600**。该进程号在每次启动 observer 进程时都会变化，所以需要每次 Attach 之前都查询确定。

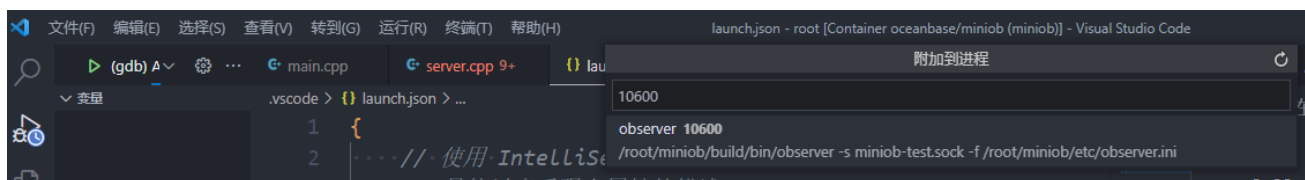


```
launchpad - root [Container oceanbase/miniob (miniob)] - Visual Studio Code
正在执行任务: ps aux | grep observer
root      10600  0.0  0.6 1035712 79628 pts/4    Ssl+  03:48   0:00 /root/miniob/build/bin/observer -s miniob-test.sock -f /root/miniob/etc/observer.ini
root      27313  0.0  0.0  9796  2676 pts/6      Ss+   04:11   0:00 /usr/bin/bash -c ps aux | grep observer
root      27316  0.0  0.0  9208  1096 pts/6      S+    04:11   0:00 grep observer
终端将被任务重用，按任意键关闭。
```

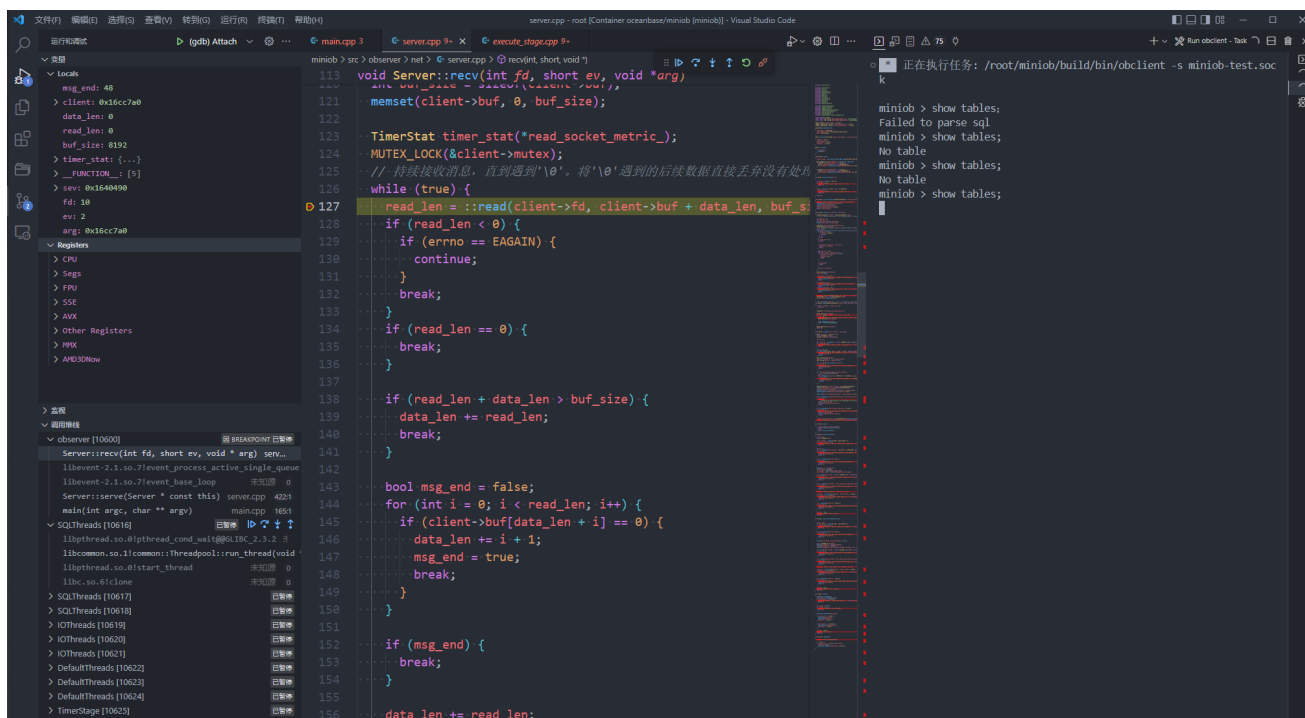
此时，在运行和调试界面下就可以看到 **(gdb) Attach** 选项。



点击绿色箭头，在弹出的对话框中填入刚获知的进程号 10600，就可以进入 Attach 调试状态。

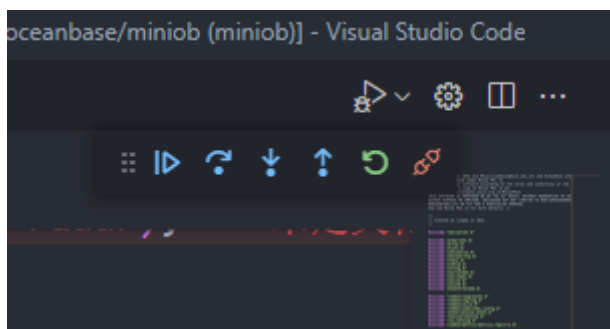


为了便于演示，在 miniob/src/observer/net/server.cpp 文件中的 Server::recv 方法中设置断点，Attach 之后就进入跟踪状态。此时我们通过 obclient（上文启动过的客户端）执行任意 SQL 语句，就会在断点处暂停执行。



可以在屏幕左侧导航窗口看到此时变量、堆栈等信息，同时客户端也处于阻塞状态，并没有回显结果，表示成功对代码进行 Attach 单步调试。

使用 **F10** , **F11** , **Shift + F11** 快捷键可以单步执行、单步跳过或者单步跳出，也可以点击相应图标实现单步调试功能。



2.3 提测代码

完成训练营中 [1.basic](#)。basic 即基础测试，是代码本身就有的功能，比如创建表、插入数据等。如果把原生仓库代码提交上去，就能够测试通过 basic。此主要目的就是为了让同学们将代码测试环境搭建好并且跑通整个调试流程。

basic

状态： — 未通过 通过率：84.2% 难度：简单 题目总分：10

miniob本身具有的一些基本功能。比如创建表、创建索引、查询数据、查看表结构等。在开发其它功能时，需要留意不要破坏这些基础功能。

每次提测之前，首先将自己开发调试好的代码使用 `git push` 等命令推送到远程仓库，然后将仓库地址复制。点击立即提测，将地址粘贴到输入框中之后，点击提交即可。等待一会即可看到提测的结果。每次提测都会对全部问题进行测试。

成绩提测



* 代码仓库地址：

Commit id：

Branch：

长度1-16个字符

重置

提交

最高成绩 ①
10.000最新成绩 ①
10.000提测次数 ①
2

提测时间	代码仓库地址	Branch	Commit id	任务状态	成绩	结果
2022-10-19 12:40	https://gitee.com/SDU-databas...	main	3322fbd55af5e2...	● 执行成功	10.000	成功 1 失败 24
2022-10-19 12:39	https://gitee.com/SDU-databas...	-	-	● 执行失败 ①	0.000	成功 0 失败 1

< 1 >

3 实验二 (20 分)

3.1 查询元数据校验实验 (10 分)

本实验对应于训练营实验 [18.select-meta](#)。

相关参考资料: [select-meta 实现解析](#)。

该实验部分测试用例如下:

SQL

```
create table t(id int, age int);
select * from t where name='a';
select address from t where id=1;
select * from t_1000;
select * from t where not_exists_col=1;
```

3.2 删除表实验 (10 分)

本实验对应于训练营实验 [17.drop-table](#)。

相关参考资料: [drop-table 实现解析](#)。

该实验部分测试用例如下:

SQL

```
create table t(id int, age int);
create table t(id int, name char);
drop table t;
create table t(id int, name char);
```

4 实验三 (20 分)

4.1 数据更新实验 (10 分)

本实验对应于训练营实验 [15.update](#)。

update 更新单个字段即可，可以参考 insert_record 和 delete_record 的实现。目前能支持 update 的语法解析，但是不能执行。需要考虑带条件查询的更新，和不带条件的更新。

该实验部分测试用例如下：

SQL

```
update t set age =100 where id=2;
update set age=20 where id>100;
```

4.2 多表笛卡尔积关联查询实验 (10 分)

本实验对应于训练营实验 [13.select-tables](#)。

当前系统支持单表查询的功能，需要在此基础上支持多张表的笛卡尔积关联查询。需要实现 `select from t1,t2;` `select t1,t2.* from t1,t2;` 以及 `select t1.id,t2.id from t1,t2;` 查询可能会带条件。查询结果展示格式参考单表查询。每一列必须带有表信息，比如：

```
t1.id | t2.id
1 | 1
```

相关参考资料：[select-tables 实现解析](#)。

该实验部分测试用例如下：

SQL

```
select * from t1,t2;
select * from t1,t2 where t1.id=t2.id and t1.age > 10;
select * from t1,t2,t3;
```

5 实验四 (10 分)

5.1 增加 date 字段实验 (10 分)

本实验对应于训练营实验 [14.date](#)。

要求实现日期类型字段。date测试不会超过2038年2月，不会小于1970年1月1号。注意处理非法的date输入，需要返回FAILURE。

当前已经支持了int、char、float类型，在此基础上实现date类型的字段。

这道题目需要从词法解析开始，一直调整代码到执行阶段，还需要考虑 DATE 类型数据的存储。

注意：

- 需要考虑date字段作为索引时的处理，以及如何比较大小；
- 这里限制了日期的范围，所以简化了溢出处理的逻辑，测试数据中也删除了溢出日期，比如没有 2040-01-02；
- 需要考虑闰年。

相关参考资料：[date 实现解析](#)，[date 测试case解析](#)。

该实验部分测试用例如下：

SQL

```
create table t(id int, birthday date);
insert into t values(1, '2020-09-10');
insert into t values(2, '2021-1-2');
select * from t;
```

6 实验五 (20 分)

6.1 聚合函数实验 (10 分)

本实验对应于训练营实验 [12.aggregation-func](#)。

实现聚合函数 max/min/count/avg，包含聚合字段时，只会出现聚合字段，不会出现如 select id, count(age) from t; 这样的测试语句。聚合函数中的参数不会是表达式，比如 age + 1。

该实验部分测试用例如下：

SQL

```
select max(age) from t1;
select count(*) from t1;
select count(1) from t1;
select count(id) from t1;
```

6.2 插入多行数据实验 (10 分)

本实验对应于训练营实验 [11.insert](#)。

实现单条插入语句插入多行数据。一次插入的数据要同时成功或失败。

7 参考资料

- [《数据库管理系统实现基础讲义》](#)
- [《数据库系统实现》](#)
- [《数据库系统概念》](#)
- [《flex_bison》](#) [flex/bison 手册](#)
- [flex开源源码](#)
- [bison首页](#)
- [cmake官方手册](#)
- [libevent官网](#)
- [SEDA wiki百科](#)
- [OceanBase数据库文档](#)
- [OceanBase开源网站](#)