课程内容

第一天上午

第一部分原理介绍：FusionDB简介（30分钟）

知识点: 1.什么是FusionDB 2.FusionDB历史简介 3.FusionDB产品特性

第二部分实验操作：

实验 1安装、部署与开发工具

（材料参考服务器上《安装指南》）

知识点: 1.安装准备 2.环境检查 3. FusionDB安装部署

实验 2基本概念及操作

知识点: 1.Linux相关命令2.数据库连接命令 3. 其他常用命令

实验 3 FusionDB的简单维护

（材料参考服务器上《管理员指南》）

知识点: 1.FusionDB启停 2.运行健康状态检查 3.备份和还原 4.数据库性能检查5.日志检查6.最大连接数检查 7.例行表、索引的维护

第一天下午

实验 4 金融数据模型设计

知识点: 1.ER图 2.关系模式 3.物理模型

实验 5 FusionDB数据库表操作

知识点: 1.数据库的初始化2.表的创建 3.插入表数据 4.添加约束 5.查询数据 6.视图管理 7. 索引管理 8.数据的修改和删除 9.新用户的创建和授权

实验 6 使用JDBC连接FusionDB数据库

知识点: 1.连接环境准备 2.JDK下载安装 3.连接FusionDB并执行Java代码

挑战 1：数据库设计

# 实验1 安装、部署与开发工具

一、实验介绍

**1.1 实验内容**

本节主要介绍FusionDB的历史、特性，与其他数据库产品的区别等入门知识。并学习安装和部署FusionDB。

**1.2 知识点**

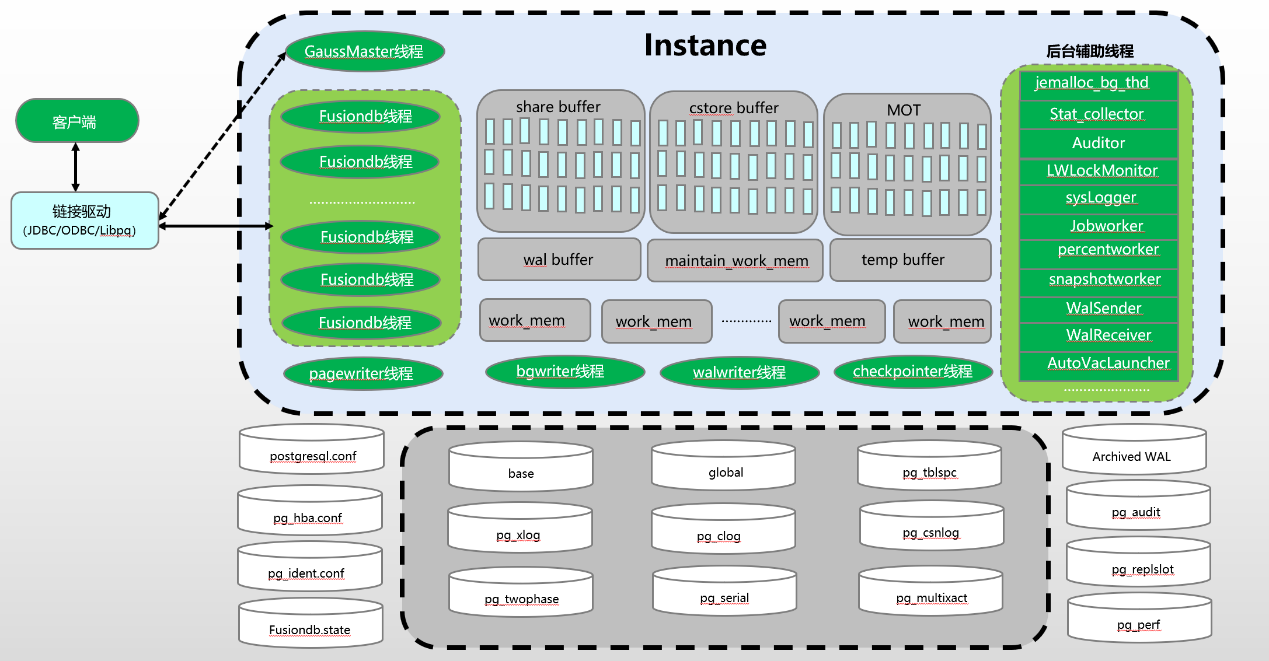
* 安装准备
* 环境检查
* FusionDB安装部署

二、实验内容

**2.1安装准备**

* **历史简介**

FusionDB是一个企业级数据库产品，是基于openGauss的可信易用商业发行版。在开源的openGauss上增强，在社区里与高斯团队从接口到内核进行共同规划、分工合作，持续共建产品核心竞争力，同时保持企业版的实用功能和更高商业价值。兼容性增强，SQL引擎可选Oracle、MySQL、PG模式，完全自主的Oracle，Mysql兼容，PG补全。



* **FusionDB的特性**

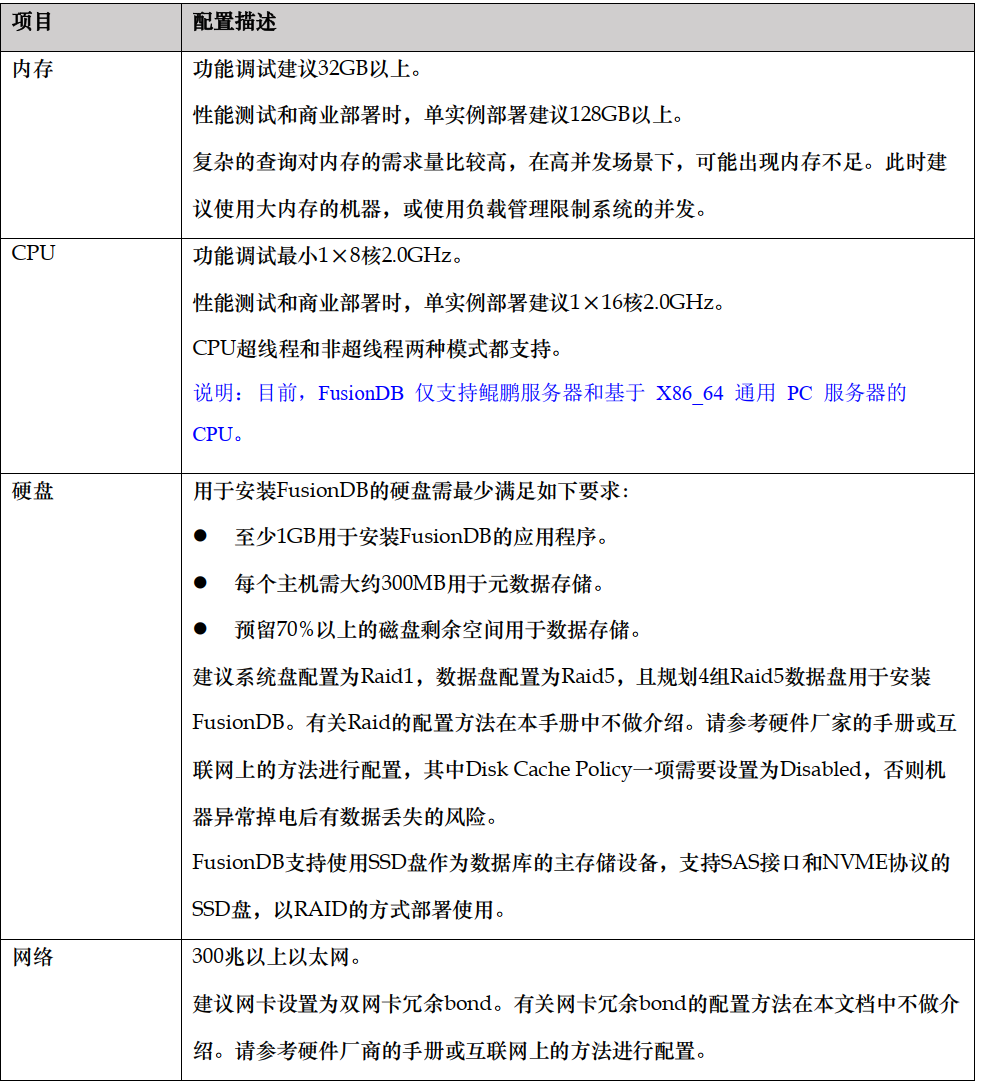
1. 极致性能特性
2. 高可用特性
3. 安全可靠性特性
4. 易用性特性
5. 兼容性
6. 平滑迁移特性

* **FusionDB与华为GaussDB到底有哪些不同**

****

**2.2 环境检查**

* **硬件环境**



* **软件环境**

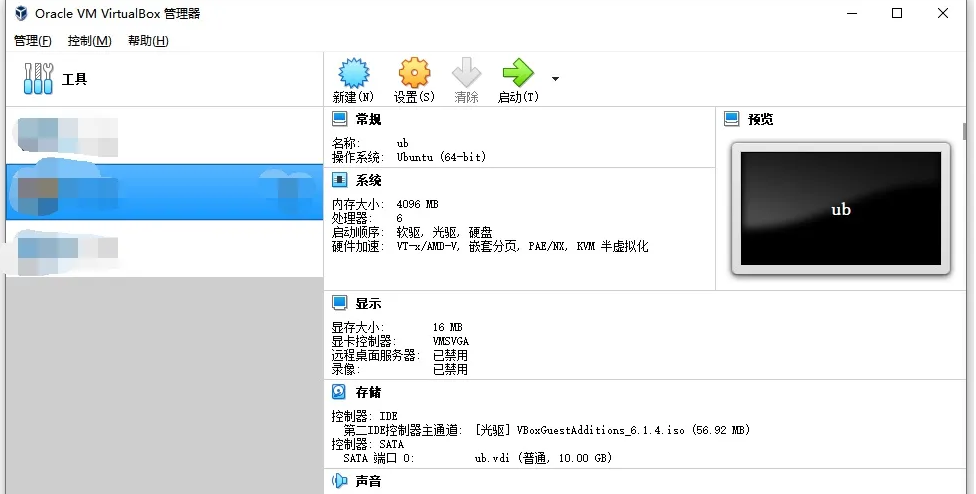


* **软件环境预安装**

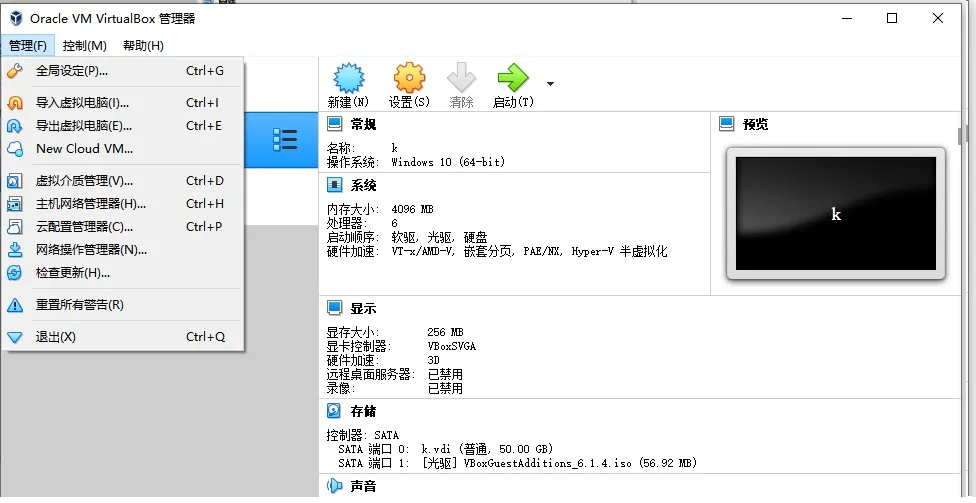
1. **安装虚拟机**

VMware和Virtual BOX，Hyper-V均可，还没有安装虚拟机的同学本实验推荐使用Oracle VM Virtual BOX。下载链接：https://www.virtualbox.org/

1. 先安装下载包virtual box-xx-xxx-win，安装成功后启动如下图



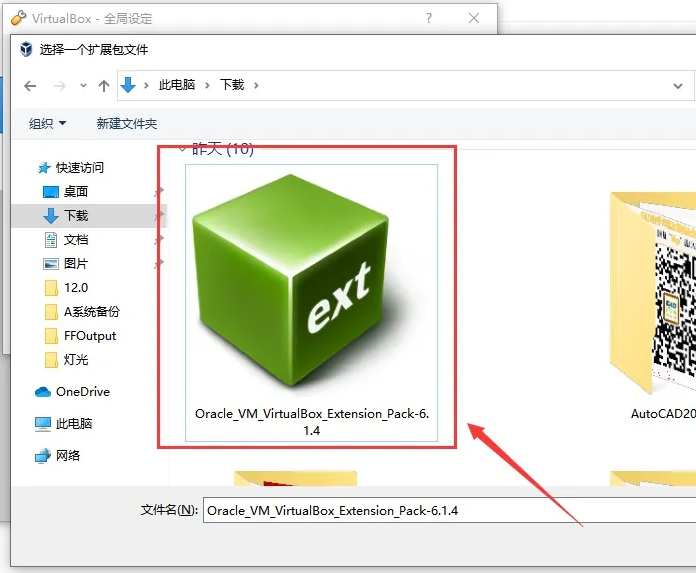
1. 安装virtual box扩展程序—VirtualBox\_Extension\_Pack，点击“管理”-》“全局设定”



1. 点击扩展



1. 找到你下载的扩展文件VirtualBox\_Extension\_Pack，然后点击“安装”

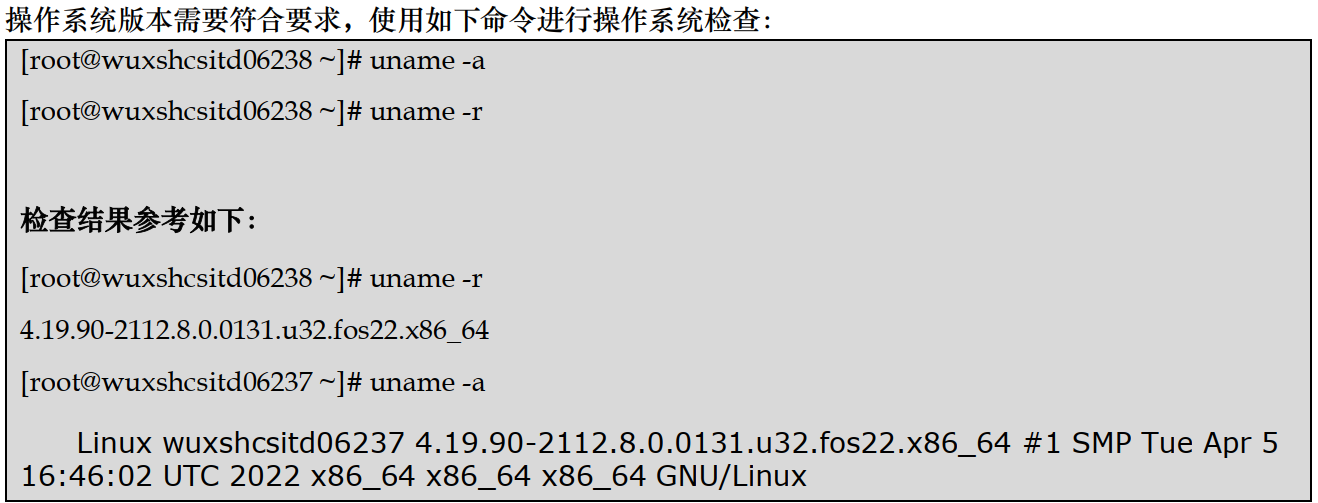


1. **安装FusionOS操作系统**

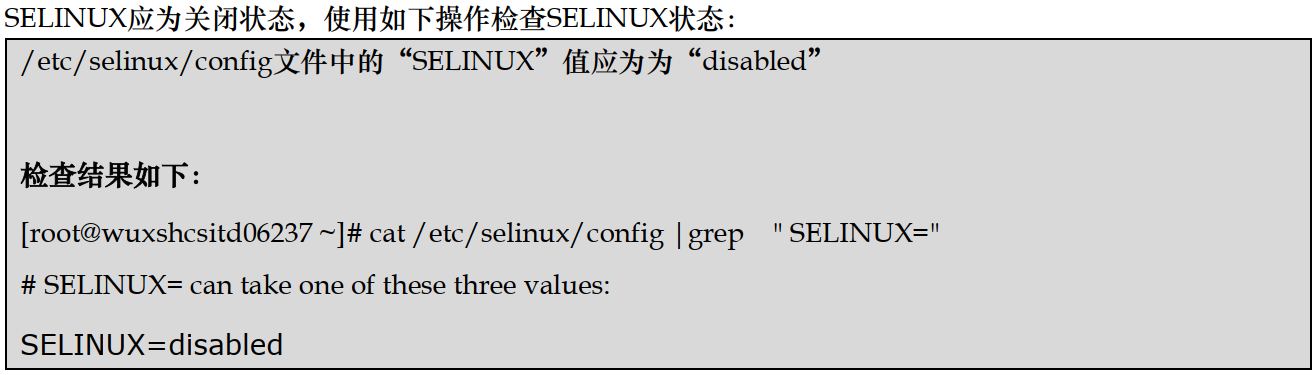
本实验接下来所有需要的安装包下载地址为：<https://pan.baidu.com/s/1NS1rItznndqVC7D9zVM_4g>，提取码：64nn

安装FusionOS：FusionOS-22\_22.0.4\_everything\_x86-64（安装程序）

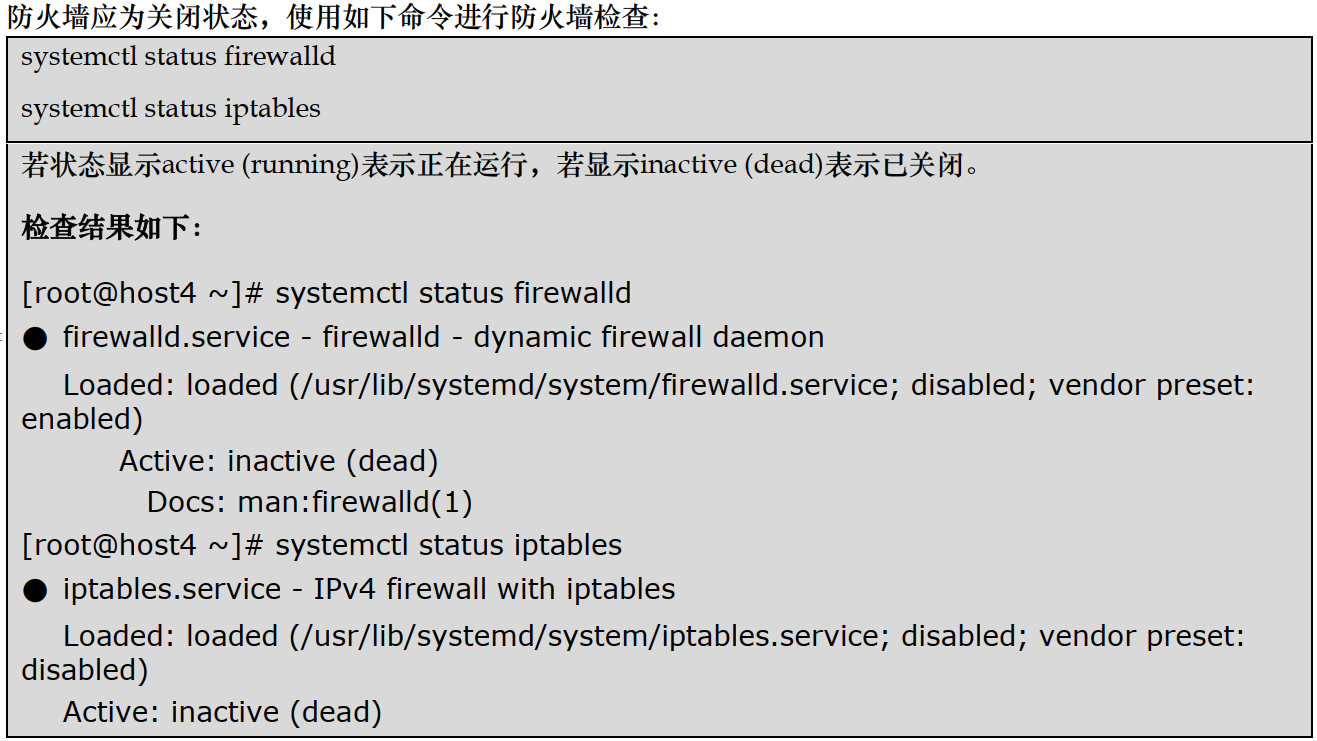
* **检查操作系统**



* **检查selinux**



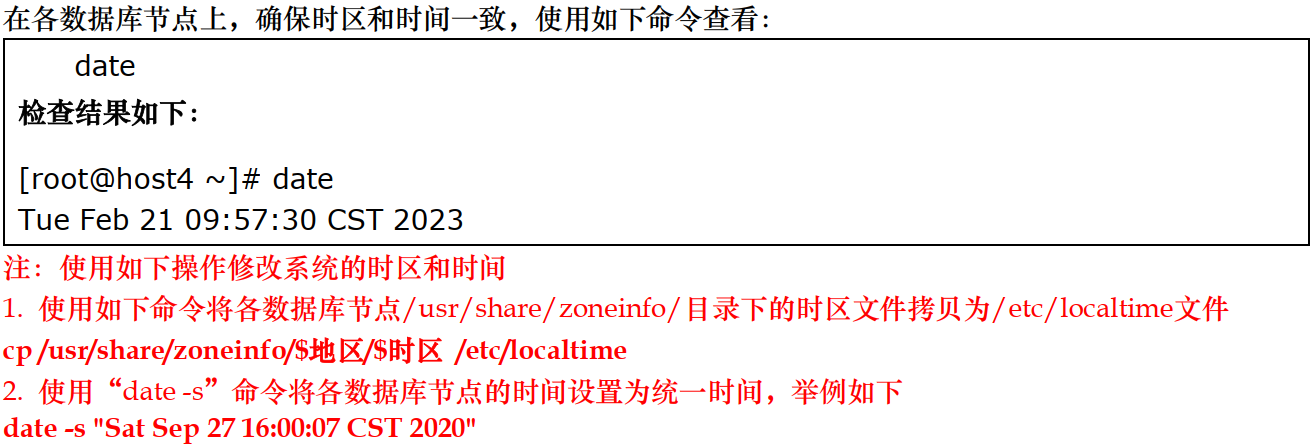
* **检查防火墙**



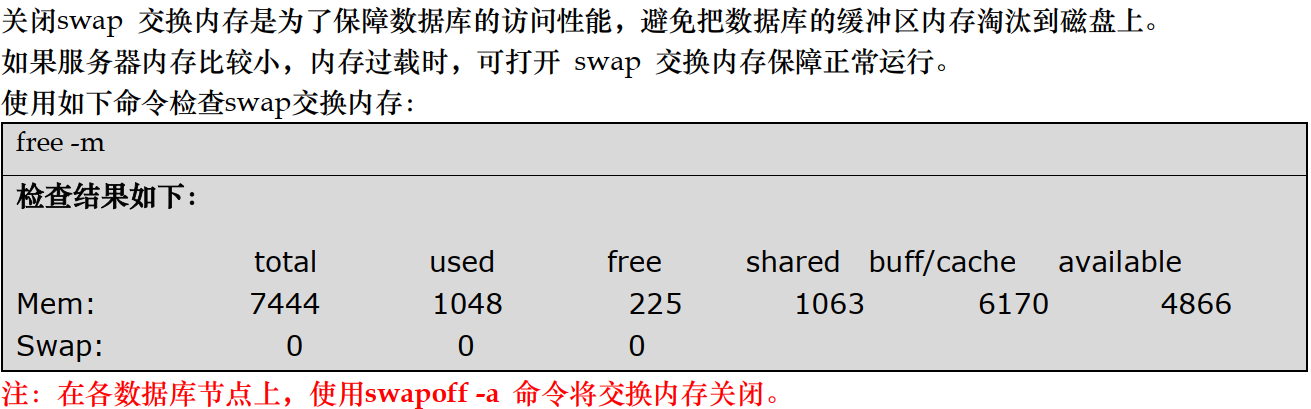
* **检查字符集参数**



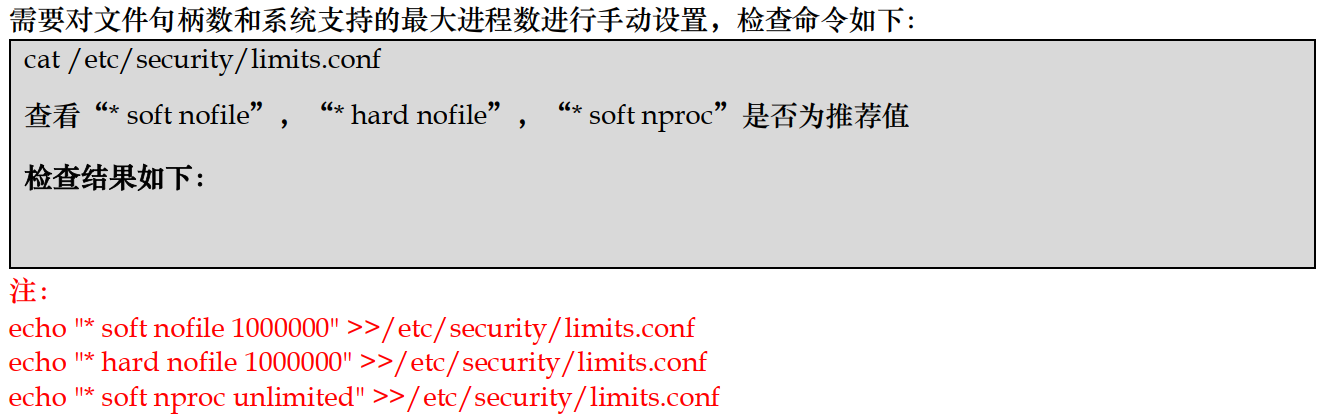
* **检查时间和时区设置**



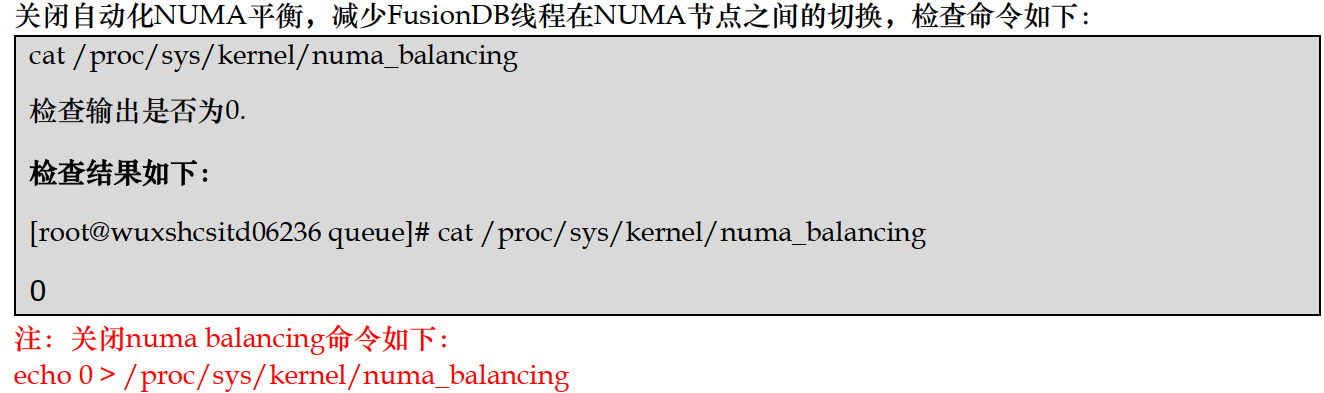
* **检查是否关闭SWAP交换内存**



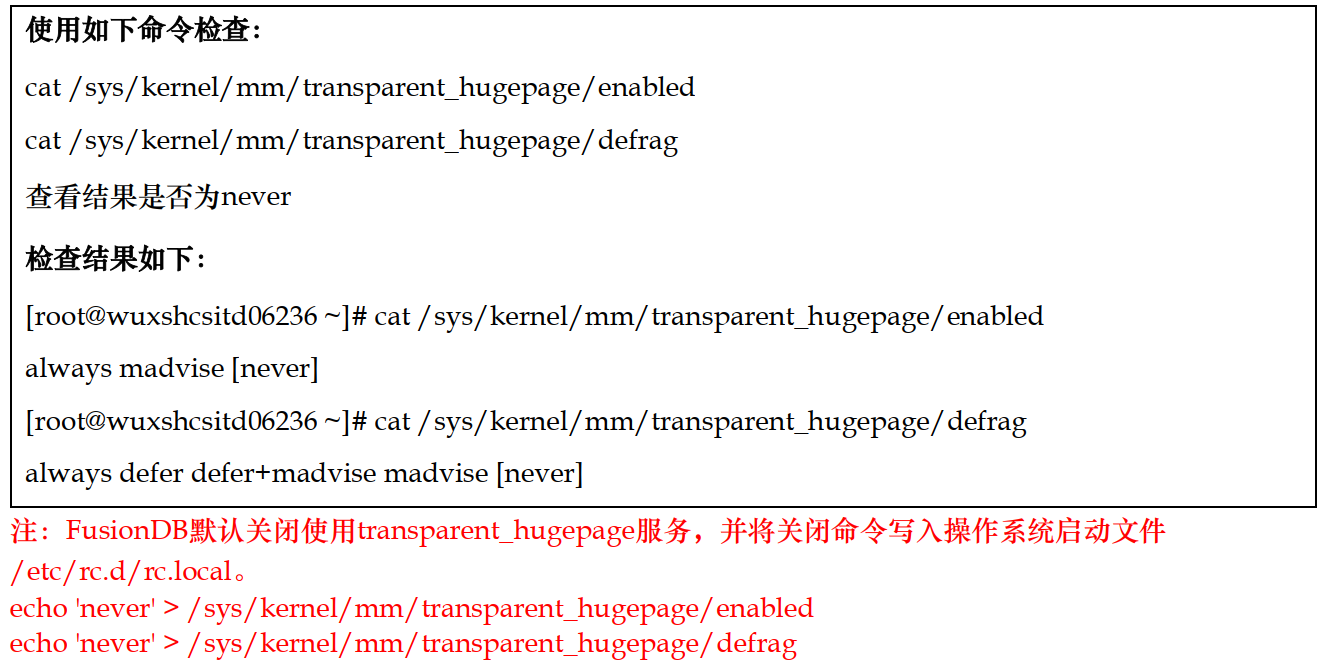
* **设置文件句柄**

****

* **关闭numa\_balancing**



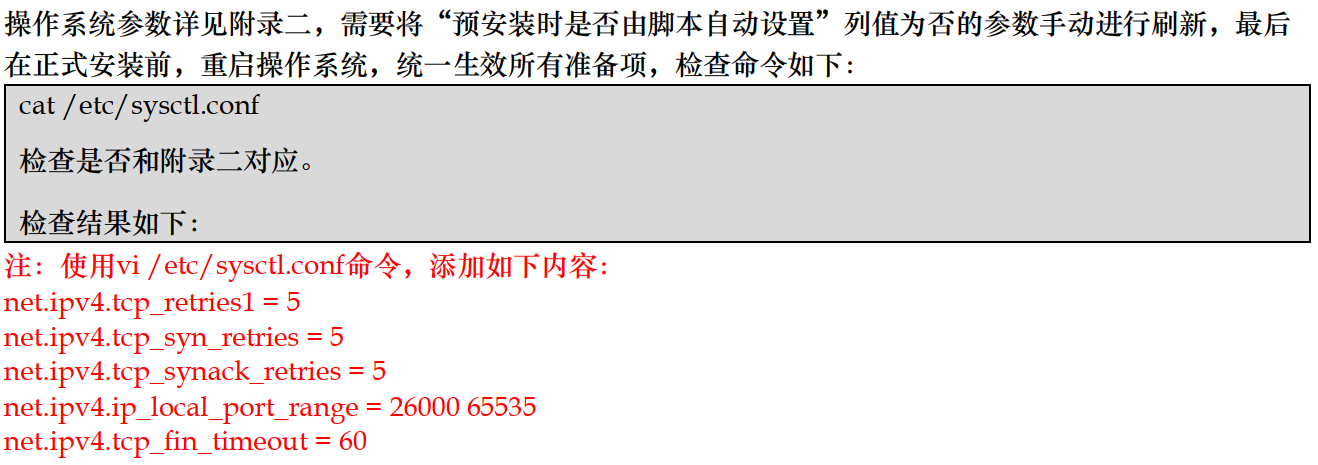
* **关闭transparent\_hugepage**



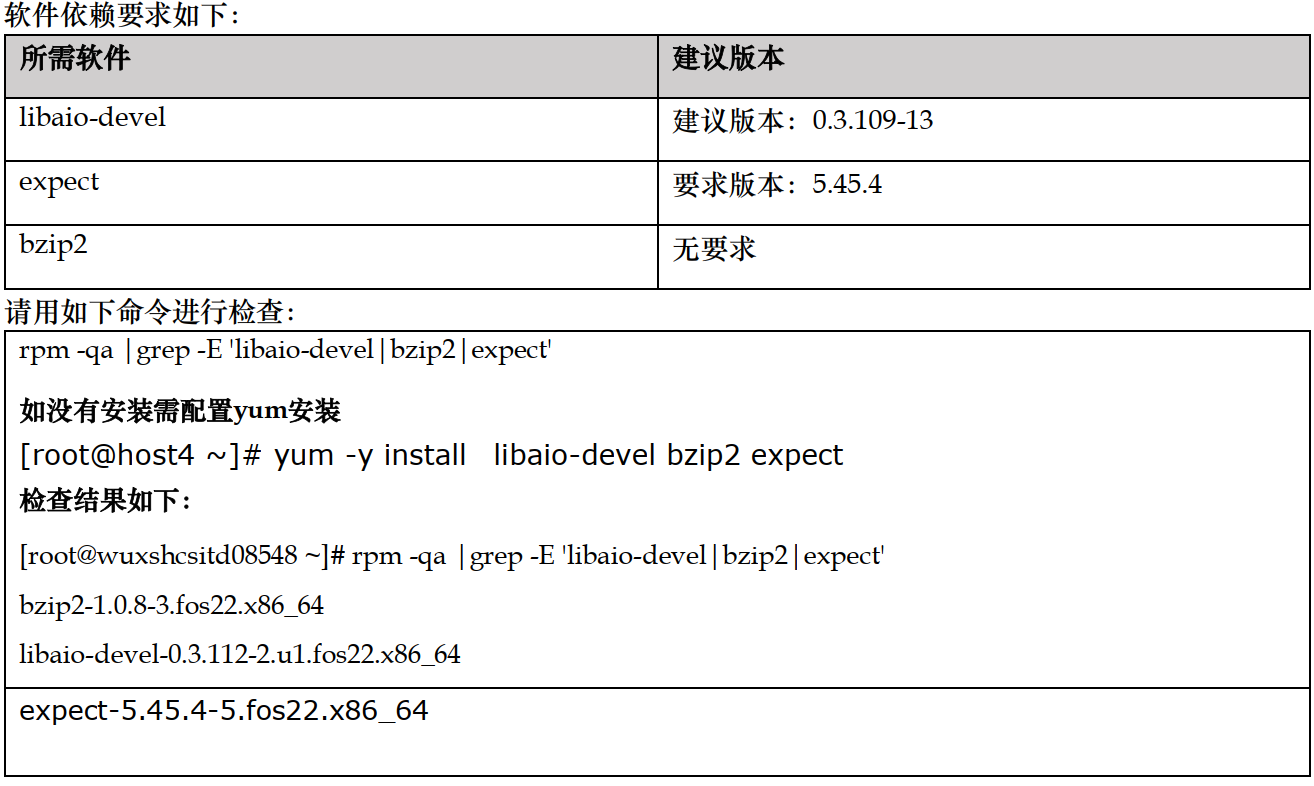
* **IO参数调优**



* **检查操作系统参数设置**



* **检查软件依赖包**



**2.3 FusionDB安装部署**

* **上传数据库软件安装包并解压**

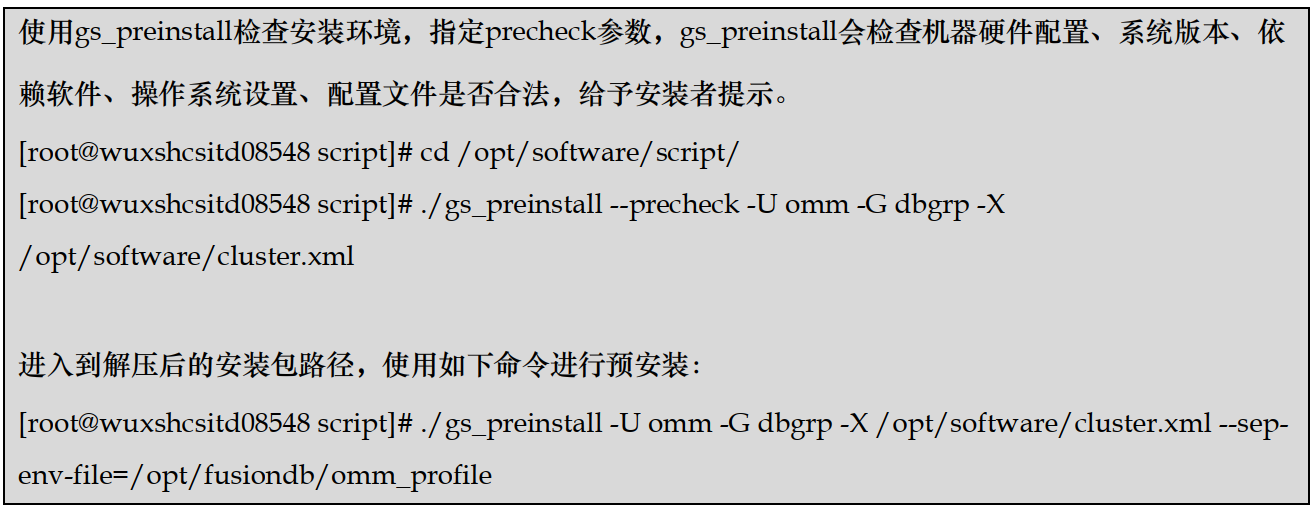


注：下载地址为：<https://pan.baidu.com/s/1NS1rItznndqVC7D9zVM_4g>，提取码：64nn

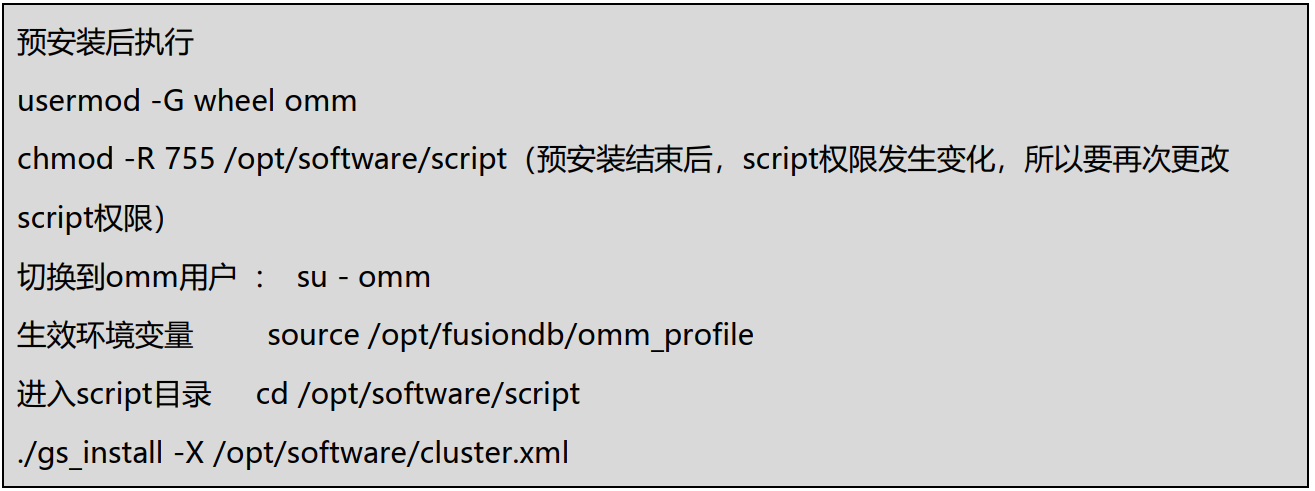
* **配置XML文件**

文件cluster\_one-node.xml放到/opt/software，需修改对应主机号、端口号便可使用

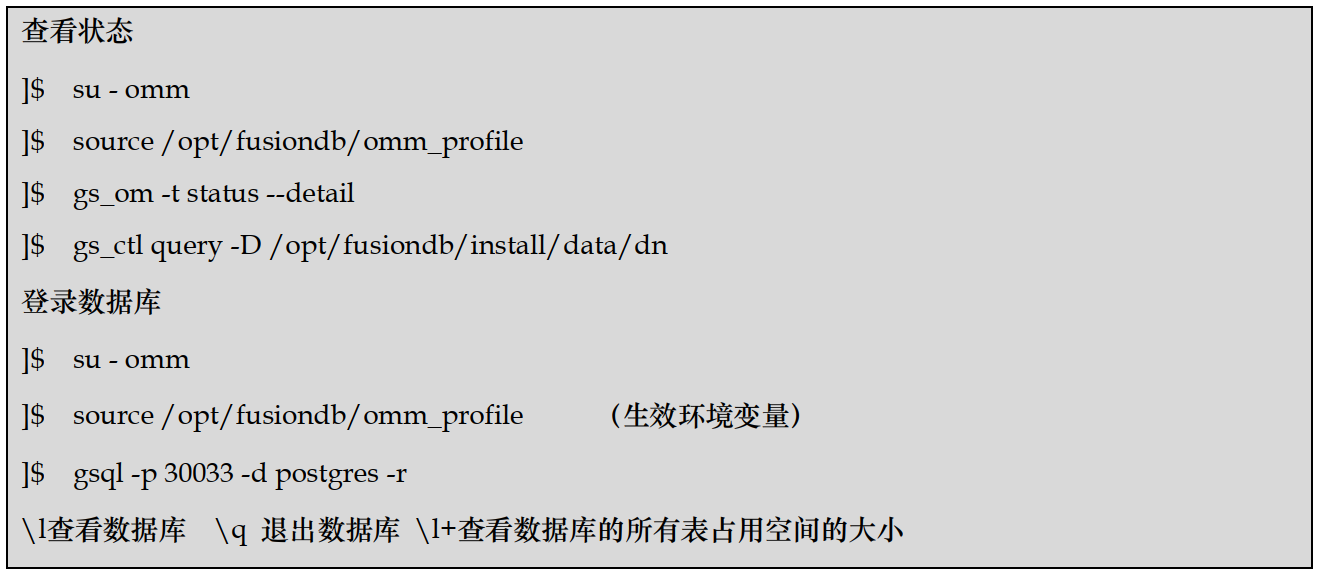
* **预安装**

****

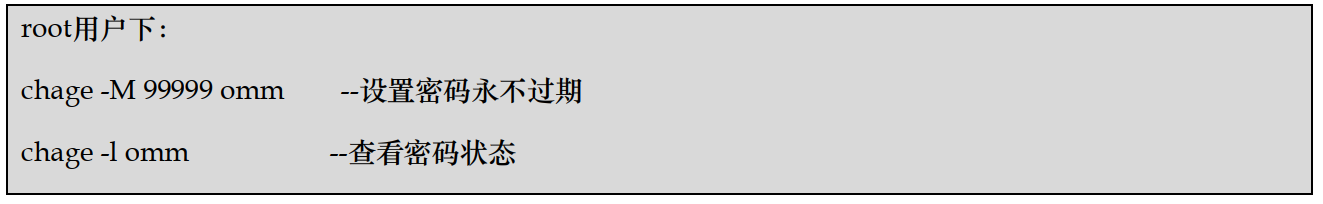
* **正式安装**

****

* **安装后检查**

****

* **修改omm用户账号永久生效**

****

# 实验2 基本概念及操作

一、实验介绍

**1.1 实验内容**

本节主要介绍FusionDB使用相关的Linux命令，并学常用的数据类型、函数使用、数据库基本操作及SQL语法。

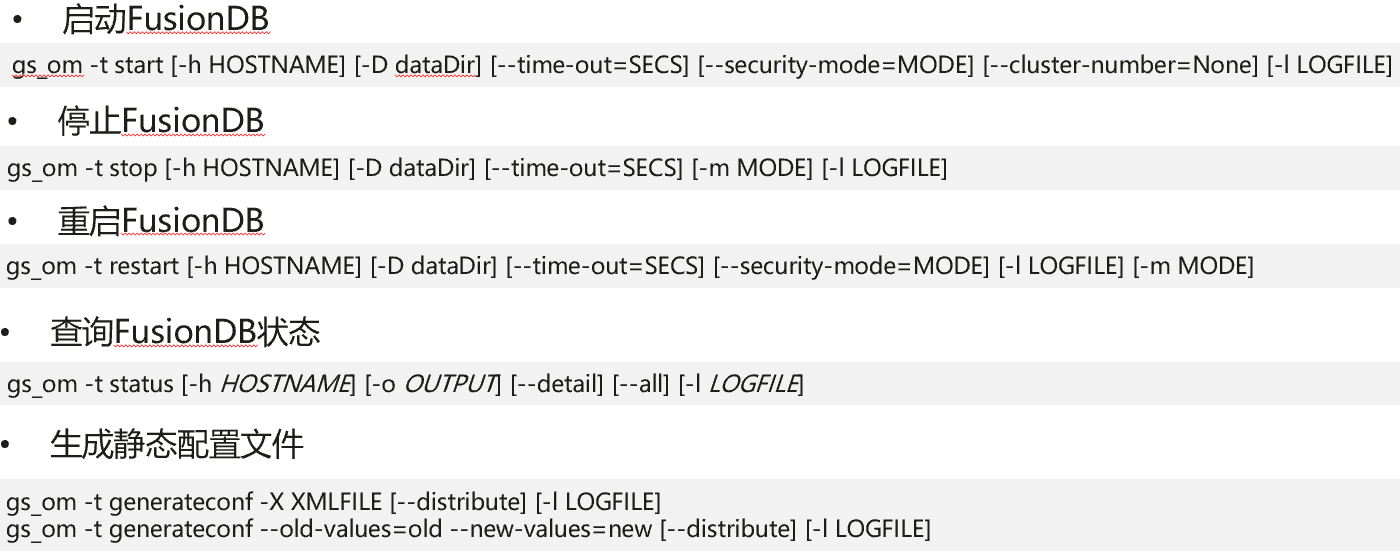
**1.2 知识点**

* Linux相关命令
* 数据库连接命令
* 其他常用命令

二、实验内容

**2.1 Linux相关命令**

**FusionDB服务管理相关命令**



**gsql是FusionDB提供的在命令行下运行的数据库连接工具，用户可以通过该工具连接服务器并对其进行操作和维护。**

* 连接数据库：创建客户端与数据库的通讯通道，在创建连接时，默认会有5分钟超时时间，数据库未正确地接受连接并对身份进行验证，gsql将超时退出。
* 执行SQL语句：支持交互式地键入并执行SQL语句，也可以执行一个文件中指定的SQL语句。
* 执行元命令：元命令可以帮助管理员查看数据库对象的信息、查询缓存区信息、格式化SQL输出结果以及连接到新的数据库等。元命令的详细说明请参见元命令参考。



****

**gsql里的元命令就是那些输入的任何以不带引号的反斜杠开头的命令。**

****

**gs\_dump：**

-f：将导出文件发送至指定目录文件夹。

-F：选择导出文件格式。-F 参数值如下： p：纯文本格式 c：自定义归档 d：目录归档格式 t：tar 归档格式

-n：只导出与模式名称匹配的模式，此选项包括模式本身和所有它包含的对象。

-t：指定导出的表（或视图、序列、外表），可以使用多个-t 选项来选择多个表，也可以使用 通配符指定多个表对象。

-T：不转储的表（或视图、或序列、或外表）对象列表，可以使用多个-T 选项来选择多个 表，也可以使用通配符指定多个表对象。

**gs\_dump部分重要指令解释：**

gs\_dumpall在导出FusionDB所有数据库时分为两部分：

1）gs\_dumpall自身对所有数据库公共的全局对象进行导出，包括有关数据库用户和组，表 空间以及属性（例如，适用于数据库整体的访问权限）信息。

2）gs\_dumpall通过调用gs\_dump来完成FusionDB中各数据库的SQL脚本文件导出，该 脚本文件包含将数据库恢复为其保存时的状态所需要的全部SQL语句。

以上两部分导出的结果为纯文本格式的SQL脚本文件，使用gsql运行该脚本文件可以恢复FusionDB数据库。

使用gs\_dumpall一次导出FusionDB的所有数据库

**gs\_restore数据导入**  
gs\_restore是FusionDB提供的针对gs\_dump导出数据的导入工具。通过此工具可由 gs\_dump生成的导出文件进行导入。gs\_restore工具由操作系统用户omm执行。

**2.2 数据库连接命令**

**连接数据库命令示例**

* 切换用户环境：切换至omm（数据库安装用户）操作系统用户环境。

$ su -omm

* 启动FusionDB

$ gs\_om -t start

* 查看数据库服务是否启动

$ gs\_om -t status

* 使用gsql登录默认数据库

$ gsql -d postgres -p 26000 -U username -W passwd -r

–username, passwd为自己的相应信息

postgres为FusionDB安装完成后默认生成的数据库，初始可以连接到此数据库进行新数据库的创建

26000为数据库端口号，可根据实际情况做替换，要确认端口有没有被占用

**2.3 其他常用命令**

* 创建数据库test

CREATE DATABASE test ENCODING ‘UTF8’ template = template0;

* 连接test数据库

\connect test

* 在test数据库下创建名为test的Schema。

CREATE SCHEMA test;

* 将默认搜索路径设为test。

SET search\_path TO test;

* 创建数据表。

create table t1( id int,name varchar(20));

insert into t1 values(1,'Tom');

insert into t1 values(2,'Jerry');

select \* from t1;

* 退出SQL编辑框，创建sql文件并执行。

$ cat >>test1.sql<<EOF

insert into t1 values(3,'Jack');

EOF

$ gsql -d test -p 26000 -U username -W passwd -f test1.sql

$ gsql -d test -p 26000 -U username -W passwd -c "select \* from t1;"

$ gsql -d test -p 26000 -U username -W passwd -l

–username, passwd为自己的相应信息

* 查询输出到指定文件。

$ gsql -d test -p 26000 -U username -W passwd -r -L /home/omm/test1.log

select \* from t1;

$ cat /home/omm/test1.log

* 格式化输出。

$ gsql -d test -p 26000 -U username -W passwd -r -A

select \* from t1;

$ gsql -d test -p 26000 -U username -W passwd -r -A -t

select \* from t1;

$ gsql -d test -p 26000 -U username -W passwd -r -A -t -F

select \* from t1;

# 实验3 FusionDB的简单维护

一、实验介绍

**1.1 实验内容**

本节主要介绍FusionDB的维护。

**1.2 知识点**

* 检查操作系统
* FusionDB启停
* 运行健康状态检查
* 备份和还原
* 数据库性能检查
* 日志检查
* 常规维护操作

二、实验内容

**2.1 FusionDB启停**

**操作系统用户omm（可根据自己的信息替换）登录数据库节点。后续操作用户默认都是omm用户下**

**启动FusionDB**

$ gs\_om -t start

Starting cluster.

=========================================

[SUCCESS] node01

=========================================

Successfully started.

**停止FusionDB**

$ gs\_om -t stop

Stopping cluster.

=========================================

Successfully stopped cluster.

=========================================

End stop cluster..

**2.2 运行健康状态检查**

**执行如下命令查询FusionDB状态**

$ gs\_om -t status --detail

[ Cluster State ]

cluster\_state : Normal

redistributing : No

current\_az : AZ\_ALL

[ Datanode State ]

node node\_ip port instance state

-----------------------------------------------------------------------------------------------

1 node01 10.5.2.47 15400 6001 /opt/fusiondb/install/data/dn P Primary Normal

通过Cluster State下的“cluster\_state”字段验证集群的状态是否正常。如果“cluster\_state”字段显示为“Normal”，说明集群的状态正常。

通过Datanode State下的“state”字段验证FusionDB节点的状态是否正常。如果“state”字段显示为“Primary Normal”，说明节点的状态正常

**2.3备份和还原**

**gs\_dump数据导出**

使用gs\_dump，导出test数据库，导出文本格式：

gs\_dump -U username -W passwd -f /backup/gsdump\_post1.sql - p 26000 test -F p

使用gs\_dump，导出test数据库，导出为tar格式：

gs\_dump -U username -W passwd -f / backup/gsdump\_post1.tar - p 26000 test -F t

**使用 gs\_restore导入gsdump\_post.tar文件内数据到test1数据库**

FusionDB =# create database test1;

$ gs\_restore / backup/gsdump\_post.tar -p 26000 -d test1

**2.4数据库性能检查**

* **杀问题会话**

查询当前会话：

FusionDB =# select datid ,pid ,state , query from pg\_stat\_activity;

datid | pid | state | query

-------+-----------------+--------+----------------------------------------------------------

15571 | 140369480042240 | idle |

15571 | 140369496823552 | active |

根据线程id结束会话

FusionDB =# select pg\_terminate\_backend(140369699206912);

pg\_terminate\_backend

----------------------

t

(1 row)

* **锁信息查看和处理**

当系统发现有大量会话阻塞时，可以执行查阻塞锁sql，查询命令见附件，查询结果见截图

Pid：139786652088064 阻塞会话

Pid：139786618525440 等待会话

根据线程id阻塞会话

FusionDB =# select pg\_terminate\_backend(139786652088064);

* **cpu使用率超过阈值**

查找数据库高消耗线程

ps -ef | grep "/gaussdb -D" | grep -v grep | awk '{print $2}' | xargs -I {} echo "top -Hp "{};

按照线程id查询高消耗sql语句：

select t1.datname,

t1.usename,

t1.pid,t1.query\_id,

t2.thread\_name,

to\_char(t2.creation\_time,'yyyy-mm-dd hh24:mi:ss') creation\_time,

t1.state,t1.query

from pg\_stat\_activity t1, dbe\_perf.OS\_THREADS t2

where t1.pid = t2.pid and t2.lwpid='线程id';

* **内存使用率&归档文件系统满**

内存使用率超过阈值

客户端或日志里出现错误：memory usage reach the max\_dynamic\_memory。

运维处理方法：

出现内存不足可能因GUC参数max\_process\_memory值设置较小相关，该参数限制一个FusionDB实例可用最大内存通过工具gs\_guc适当调整max\_process\_memory参数值。注意需重启实例生效。

归档文件系统满

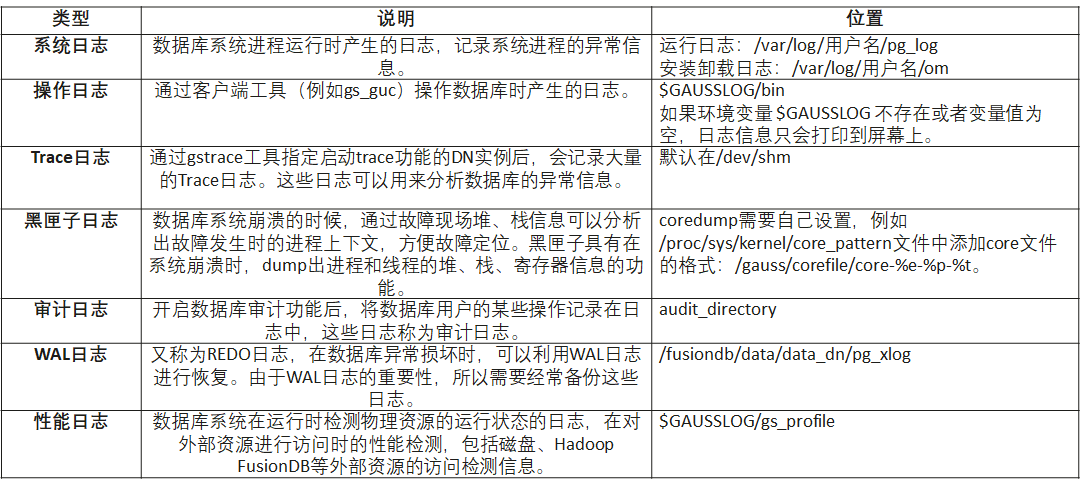
以归档路径archive\_dest='/opt/fusiondb/install/data/arch'为例，执行删除命令，删除一天之前的归档数据释放空间。

删除命令：

find /opt/fusiondb/install/data/arch -mtime +1 -name "0000000\*" -exec rm -rf {} \;

**2.5 日志检查**

* **查看数据库日志信息**

****

* **收集日志**

使用如下命令，收集问题时间段的日志信息：

gs\_collector --begin-time="20230711 01:01" --end-time="20230711 23:59“

收集完成后会有如下输出：

Collecting catalog statistics.

Successfully collected catalog statistics.

do log check interval 0 : count 1

Collecting Log files.

Successfully collected Log files.

do Config check 0:1

Collecting Config files.

Successfully collected Config files.

Collecting files.

Successfully collected files.

All results are stored in /opt/fusiondb/install/log/omm/collector\_20230711\_101731.tar.gz

将生成的日志信息提交给相关运维人员分析。

**2.6 常规维护操作**

收集完成后会有如下输出：

创建角色并赋予权限

Create role testrole with password ‘1qaz!QAZ’;

Alter role testrole login;

创建用户并赋予角色

Create user jack with password ‘1qaz!QAZ’;

Grant testrole to jack;

创建表空间

--创建表空间test，所有者指定为用户jack，指定表空间位置为tablespace/test\_1。

CREATE TABLESPACE test OWNER jack RELATIVE LOCATION 'tablespace/test\_1’;

创建数据库

--创建数据库testdb，所有者指定为jack，将数据库指定表空间为test。

CREATE DATABASE testdb OWNER jack TABLESPACE test;

查看数据库参数

方法一：show命令参看

查看单独参数：show parameter\_name;

查看所有参数：show all;

方法二：通过系统函数查看select \* from pg\_sesstings;

参数设置

方法一：此方法设置参数需重启生效

使用如下命令修改参数。

gs\_guc set -D datadir -c "paraname=value“

说明：如果参数是一个字符串变量，则使用-c parameter="'value'"或者使用-c "parameter = 'value'"。

使用以下命令在数据库节点上同时设置某个参数。

gs\_guc set -N all -I all -c "paraname=value“

使用以下命令在数据库节点上设置cm\_agent某个参数。

gs\_guc set -Z cmagent -c "paraname=value"

gs\_guc set -Z cmagent -N all -I all -c "paraname=value"

# FusionDB数据库设计规范

1. **Database设计建议**

【规则】在实际业务中，根据需要创建新的Database，不建议直接使用集群默认的postgres数据库。

【建议】为适应全球化需求，使数据库编码能够存储与表示绝大多数字符，建议创建Database时使用UTF-8编码。

【关注】创建Database时，需要重点关注字符集编码(ENCODING)和兼容性(DBCOMPATIBILITY)两个配置项。GaussDB(for openGauss)支持TD、ORA、MYSQL和PG四种兼容模式，分别兼容Teradata语法、Oracle语法、MYSQL语法和PostgreSQL语法，不同兼容模式下的语法行为存在一定差异，默认为Oracle兼容模式。

【关注】Database的owner默认拥有该Database下所有对象的所有权限，包括删除权限。删除权限影响较大，请谨慎使用。

1. **Schema设计建议**

* 如果该用户不具有sysadmin权限或者不是该Schema的owner，要访问Schema下的对象，需要同时给用户赋予Schema的usage权限和对象的相应权限。
* 如果要在Schema下创建对象，需要授予操作用户该Schema的create权限。
* Schema的owner默认拥有该Schema下对象的所有权限，包括删除权限。删除权限影响较大，请谨慎使用。

1. **字段设计**

在字段设计时，基于查询效率的考虑，一般遵循以下原则：

* 尽量使用高效数据类型。选择数值类型时，在满足业务精度的情况下，选择数据类型的优先级从高到低依次为整数、浮点数、NUMERIC。
* 当多个表存在逻辑关系时，表示同一含义的字段应该使用相同的数据类型。
* 对于字符串数据，建议使用变长字符串数据类型，并指定最大长度。请务必确保指定的最大长度大于需要存储的最大字符数，避免超出最大长度时出现字符截断现象。除非明确知道数据类型为固定长度字符串，否则，不建议使用CHAR(n)、BPCHAR(n)、NCHAR(n)、CHARACTER(n)。

1. **表设计**

* 设计表结构时，建议规划好，避免经常需要添加字段，或者修改字段类型或长度。
* 对于非临时表，禁止使用unlogged关键建表，默认创建非压缩行表。
* 建表不需要指定表空间，会使用建库时指定的默认表空间。
* 表之间的关联字段应该尽量少。
* 作为表间连接关系的字段，数据类型必须保持严格一致，避免索引无法正常使用。
* 关联字段在命名上，应该可以明显体现出关联关系。例如，采用同样名称来命名。
* 不建议使用VARCHAR或其他字符类型来存储日期值，如果使用，则不能在此字段上做运算，需要在数据规范中严格定义。
* 字段必须添加能够清楚表示其含义的注释，状态类字段的注释中必须明确列出个状态值的说明。
* 对于频繁更新的表，建议建表时指定表的fillfactor=85，每页预留15%的空间给HOT更新使用。
* 表结构中字段定义的数据类型与应用程序中的定义保持一致，表之间字段校对规则一致，避免报错或无法使用索引的情况发生。
* 不建议使用宽表，建议每张表不超过25列。

1. **索引设计**

* 建议create 或 drop index 时,加 CONCURRENTLY参数,这是个好习惯，达到与写入数据并发的效果，列存表、分区表和临时表不支持CONCURRENTLY方式创建索引。
* 建议对于频繁update、delete的包含于index 定义中的column的table, 用create index CONCURRENTLY , drop index CONCURRENTLY的方式进行维护其对应index。
* 建议对where 中带多个字段and条件的高频查询，参考数据分布情况，建多个字段的联合索引，区分度高的字段放索引前面。
* 建议对固定条件的（一般有特定业务含义）且选择比好（数据占比低）的查询语句，创建带where的Partial Indexes。
* 经常在条件列上使用表达式或者函数的查询，建议尽量改写为表达式或函数使用在条件值上，如果无法改写可以使用函数索引或表达式索引来提高查询效率。
* 每个表的index数量不建议太多(建议不超过5个)。
* 频繁DML(写次数明显超过读次数)的表,不要建立太多的索引(建议不超过3个)。
* 无用的索引以及重复索引应及时删除，避免对执行计划及数据库性能造成负面影响。
* 索引字段的总长度不超过50字节，否则，索引大小会膨胀比较严重。
* INDEX类型推荐：INT,NUMBER,NUMERIC,date,短varChar(),Char()。

1. **约束设计**

**DEFAULT和NULL约束**

* 如果能够从业务层面补全字段值，那么，就不建议使用DEFAULT约束，避免数据加载时产生不符合预期的结果。
* 给明确不存在NULL值的字段加上NOT NULL约束，优化器会在特定场景下对其进行自动优化。
* 给可以显式命名的约束显式命名。除了NOT NULL和DEFAULT约束外，其他约束都可以显式命名。

**主键约束**

* 建议采用自增键作为主键。
* 建议不要用有业务含义的名称作为主键,比如身份证或者国家名称,尽管其是unique的。

**唯一约束**

* 除主键外，需存在唯一性约束的，可通过创建以“uk\_”为前缀的唯一索引实现。

**外键约束**

* 为了便于维护，不建议使用外键。

**检查约束**

* 为了便于维护，不建议使用检查约束。

1. **列设计**

* 建议能用数值类型的，就不用字符类型。
* 建议能用varchar(N) 就不用char(N), 能用varchar(N) 就不用text、大对象等。
* 新建数据库默认兼容oracle, not null 约束不允许传入空字符串，空字符串默认会转换为null。
* 建议使用NUMERIC(precision, scale)来存储货币金额和其它要求精确计算的数值, 而不建议使用real、double、float等浮点类型。

1. **视图设计**

* 尽量使用简单视图，尽可能少使用复杂视图。
* 简单视图定义：数据来自单个表,且无分组（DISTINCT/GROUP BY）、无函数。
* 复杂视图定义：数据来自多个表,或有分组,有函数，表的个数不能超过3个。
* 尽量不要使用嵌套视图，如果必须使用，不能超过2层嵌套。

# 实验4 金融数据模型设计

一、实验介绍

**1.1 实验内容**

本节以金融行业为场景，设计数据库模型。

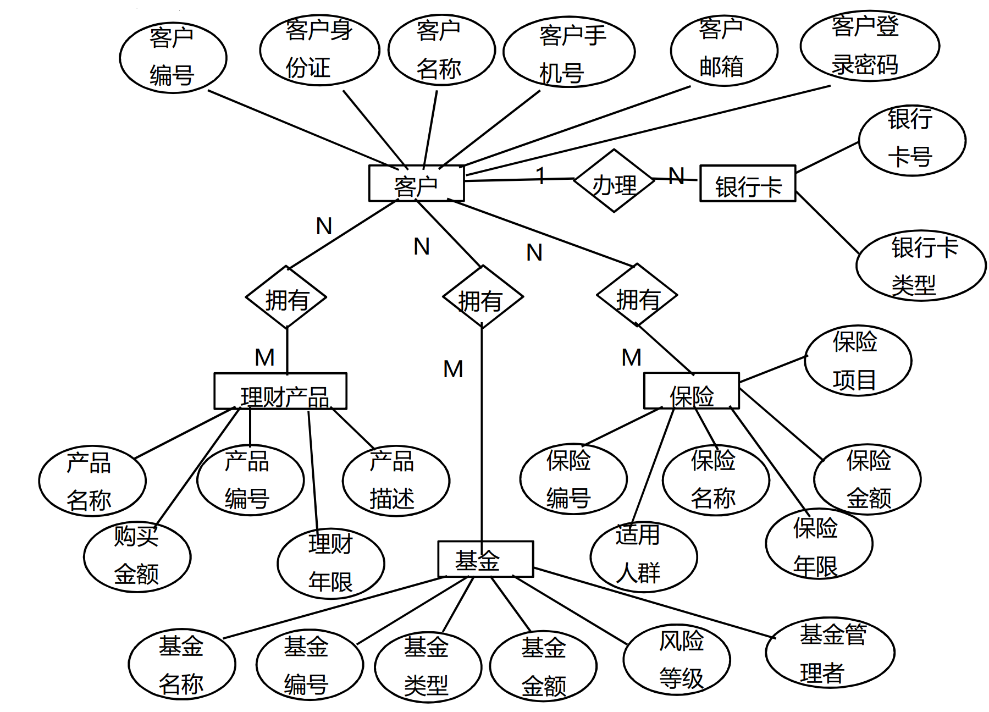
假设A市C银行为了方便对银行数据的管理和操作，引入了超聚变FusionDB数据库。针对C银行的业务，本实验主要将对象分为客户、银行卡、理财产品、保险和基金。因此，针对这些数据库对象，本实验假设C银行的金融数据库存在着以下关系：客户可以办理银行卡，同时客户可以购买不同的银行产品，如理财产品，基金和保险。那么，根据C银行的对象关系，本实验给出了相应的关系模式和ER图。

**1.2 知识点**

* ER图
* 关系模式
* 物理模型

二、实验内容

**2.1 ER图**



**ER图**

**2.2 关系模式**

对于C银行中的5个对象，分别建立属于每个对象的属性集合，具体属性描述如下：

客户（客户编号、客户名称、客户邮箱，客户身份证，客户手机号，客户登录密码）

银行卡（银行卡号，银行卡类型）

理财产品（产品名称，产品编号，产品描述，购买金额，理财年限）

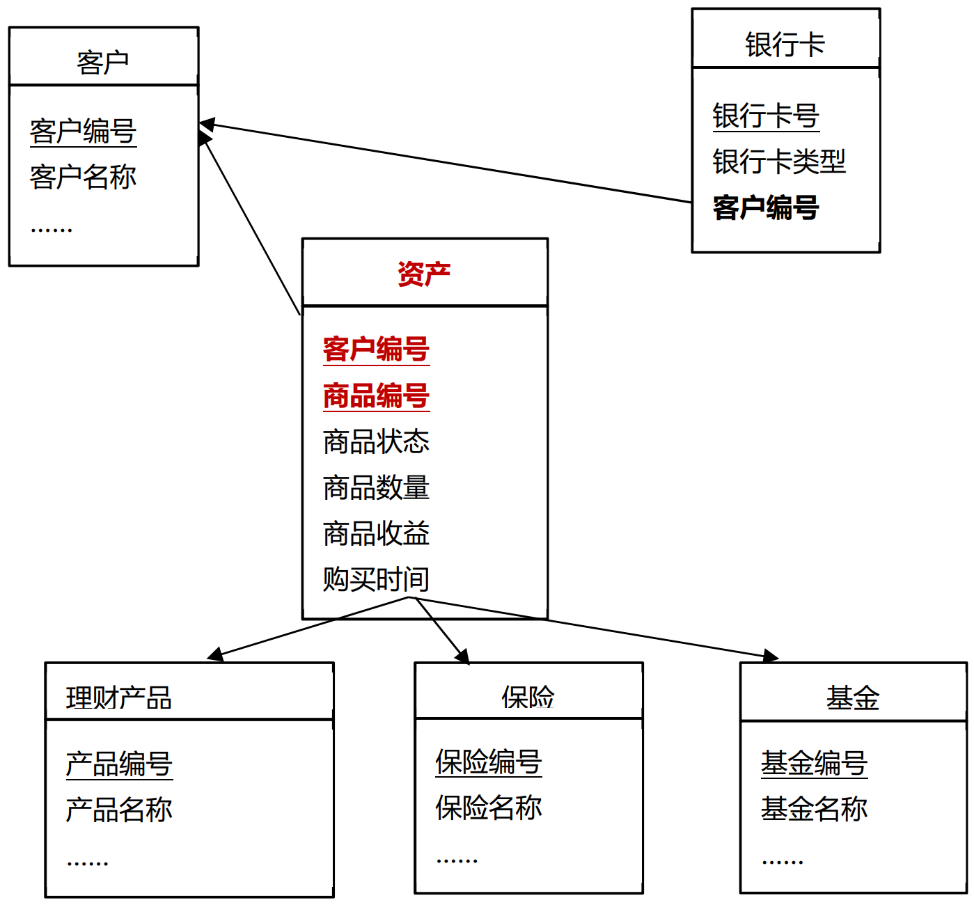
保险（保险名称，保险编号，保险金额，适用人群，保险年限，保障项目）

基金（基金名称，基金编号，基金类型，基金金额，风险等级，基金管理者）

对象之间的关系：

* 一个客户可以办理多张银行卡
* 一个客户可以购买多个理财产品，同一类理财产品可由多个客户购买
* 一个客户可以购买多个基金，同一类基金可由多个客户购买
* 一个客户可以购买多个保险，同一类保险可由多个客户购买

根据关系分析，设计关系模式如下：



金融数据关系模型设计图

说明：

* 由于一个客户可以办理多张银行卡，所以银行卡表引用客户表的客户编号作为外键。
* 由于一个客户可以购买多个理财产品，同一类理财产品可由多个客户购买。所以生成关系表——资产表。资产表引用客户表的商品编号作为外键，引用理财产品表的产品编号作为外键，并且添加商品状态、商品数量、商品收益和购买时间等属性。
* 客户和保险、客户和基金的关系同理，所以资产表同样作为生成的关系表，修改资产表的理财产品编号为商品编号，商品编号引用自理财产品表、保险和基金表的编号。

**2.3 物理模型**

对象及属性对应的编号为：

Client(c\_id，c\_name，c\_mail，c\_id\_card，c\_phone，c\_password)

bank\_card(b\_number，b\_type，**b\_c\_id**)

finances\_product(p\_name，p\_id，p\_description，p\_amount，p\_year)

insurance(i\_name，i\_id，i\_amount，i\_person，i\_year，i\_project)

fund(f\_name，f\_id，f\_type，f\_amount，risk\_level，f\_manager)

property(**pro\_id**,pro\_c\_id，pro\_pif\_id，**pro\_type**，pro\_status，pro\_quantity，pro\_income，pro\_purchase\_time)

说明：

* 资产表（property）中由于商品编号(pro\_pif\_id)字段引用自理财产品表、保险和基金表的id字段，为了防止三商品的id字段相互冲突，添加商品类型(pro\_type)字段区分三种商品。并且资产表添加资产编号(pro\_id)字段作为主键。

Client（客户）表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段类型 | 约束 | 说明 |
| c\_id | INTEGER | PRIMARY KEY | 客户编码 |
| c\_name | VARCHAR(100) | NOT NULL | 客户名称 |
| c\_mail | CHAR(30) | UNIQUE | 客户邮箱 |
| c\_id\_card | CHAR(20) | UNIQUE NOT NULL | 客户身份证 |
| c\_phone | CHAR(20) | UNIQUE NOT NULL | 客户手机号 |
| c\_password | CHAR(20) | NOT NULL | 客户登录密码 |

bank\_card（银行卡）表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段类型 | 约束 | 说明 |
| b\_number | CHAR(30) | PRIMARY KEY | 银行卡号 |
| b\_type | CHAR(20) |  | 银行卡类型 |
| b\_c\_id | INTEGER | NOT NULL | 所属客户编号  注：本字段引用自client表的c\_id字段。 |

finances\_product（理财产品）信息表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段类型 | 约束 | 说明 |
| p\_name | VARCHAR(100) | NOT NULL | 产品名称 |
| p\_id | INTEGER | PRIMARY KEY | 产品编号 |
| p\_description | VARCHAR(4000) |  | 产品描述 |
| p\_amount | INTEGER |  | 购买金额 |
| p\_year | INTEGER |  | 理财年限 |

insurance（保险）表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段类型 | 约束 | 说明 |
| i\_name | VARCHAR(100) | NOT NULL | 保险名称 |
| i\_id | INTEGER | PRIMARY KEY | 保险编号 |
| i\_amount | INTEGER |  | 保险金额 |
| i\_person | CHAR(20) |  | 适用人群 |
| i\_year | INTEGER |  | 保险年限 |
| i\_project | VARCHAR(200) |  | 保障项目 |

fund（基金）表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段类型 | 约束 | 说明 |
| f\_name | VARCHAR(100) | NOT NULL | 基金名称 |
| f\_id | INTEGER | PRIMARY KEY | 基金编号 |
| f\_type | CHAR(20) |  | 基金类型 |
| f\_amount | INTEGER |  | 基金金额 |
| risk\_level | CHAR(20) | NOT NULL | 风险等级 |
| f\_manager | INTEGER | NOT NULL | 基金管理者 |

property（资产）表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 字段类型 | 约束 |  |
| pro\_id | INTEGER | PRIMARY KEY | 资产编号 |
| pro\_c\_id | VARCHAR(100) | NOT NULL | 客户编号  说明：本字段引用自client表的c\_id字段。 |
| pro\_pif\_id | INTEGER | NOT NULL | 商品编号  说明：本字段引用自finances\_product表、insurance表和fund表三个表的id字段。 |
| pro\_type | INTEGER | NOT NULL | 商品类型  说明：1表示理财产品；2表示保险；3表示基金。 |
| pro\_status | CHAR(20) |  | 商品状态 |
| pro\_quantity | INTEGER |  | 商品数量 |
| pro\_income | INTEGER |  | 商品收益 |
| pro\_purchase\_time | DATE |  | 购买时间 |

接下来进行FusionDB数据模型表操作。

# 实验5 FusionDB数据库表操作

一、实验介绍

**1.1 实验内容**

本节主要介绍FusionDB的数据库表操作。

**1.2 知识点**

* 数据库的初始化
* 表的创建
* 插入表数据
* 添加约束
* 查询数据
* 视图管理
* 索引管理
* 数据的修改和删除
* 新用户的创建和授权

二、实验内容

**2.1****初始化数据库**

**连接数据库**

gsql -d dbname -p 26000 -U username -W passwd -r

–dbname, username, passwd为自己的相应信息

创建数据库

* 切换用户环境：切换至omm（数据库安装用户）操作系统用户环境。

su - omm

* 启动FusionDB

gs\_om -t start

* 查看服务是否启动

gs\_om -t status

* 使用gsql登录默认数据库

gsql -d postgres -p 26000 -r

postgres为FusionDB安装完成后默认生成的数据库，初始可以连接到此数据库进行新数据库的创建

26000为数据库端口号，可根据实际情况做替换，要确认端口有没有被占用

* 登录数据库实例后，创建名为finance的数据库。

CREATE DATABASE finance ENCODING ‘UTF8’ template = template0;

* 在finance数据库下创建名为finance的Schema。

CREATE SCHEMA finace;

finance Schema创建成功后，将数据库切换至对应的Schema。

**2.2表的创建**

根据C银行的场景描述，本实验分别针对客户(client)，银行卡(bank\_card)，理财产品(finances\_product)，保险(insurance)，基金(fund)和资产(property)创建相应的表。具体的实验步骤如下所示：

1.当前库当前schema下如果有client表，先删除。

DROP TABLE IF EXISTS client;

创建表client

CREATE TABLE client

(

c\_id INT PRIMARY KEY,

c\_name VARCHAR(100) NOT NULL,

c\_mail CHAR(30) UNIQUE,

c\_id\_card CHAR(20) UNIQUE NOT NULL,

c\_phone CHAR(20) UNIQUE NOT NULL,

c\_password CHAR(20) NOT NULL

);

3.银行卡信息表的创建

当前库当前schema下如果有bank\_card表，先删除。

DROP TABLE IF EXISTS bank\_card;

创建表bank\_card

CREATE TABLE bank\_card

(

b\_number CHAR(30) PRIMARY KEY,

b\_type CHAR(20),

b\_c\_id INT NOT NULL

);

3.理财产品信息表的创建

当前库当前schema下如果有finances\_product表，先删除。

DROP TABLE IF EXISTS finances\_product;

创建表finances\_product

CREATE TABLE finances\_product

(

p\_name VARCHAR(100) NOT NULL,

p\_id INT PRIMARY KEY,

p\_description VARCHAR(4000),

p\_amount INT,

p\_year INT

);

4.保险信息表的创建：

当前库当前schema下如果有insurance表，先删除。

DROP TABLE IF EXISTS insurance;

创建表insurance

CREATE TABLE insurance

(

i\_name VARCHAR(100) NOT NULL,

i\_id INT PRIMARY KEY,

i\_amount INT,

i\_person CHAR(20),

i\_year INT,

i\_project VARCHAR(200)

);

5.基金信息表的创建：

当前库当前schema下如果有fund表，先删除。

DROP TABLE IF EXISTS fund;

创建表fund

CREATE TABLE fund

(

f\_name VARCHAR(100) NOT NULL,

f\_id INT PRIMARY KEY,

f\_type CHAR(20),

f\_amount INT,

risk\_level CHAR(20) NOT NULL,

f\_manager INT NOT NULL

);

6.资产信息表的创建：

当前库当前schema下如果有property表，先删除。

DROP TABLE IF EXISTS property;

创建表property

CREATE TABLE property

(

pro\_id INT PRIMARY KEY,

pro\_c\_id INT NOT NULL,

pro\_pif\_id INT NOT NULL,

pro\_type INT NOT NULL,

pro\_status CHAR(20),

pro\_quantity INT,

pro\_income INT,

pro\_purchase\_time DATE

);

**2.3插入表数据**

为了实现对表数据的相关操作，本实验需要以执行SQL的方式对金融数据库的相关表插入部分数据。

1.初始化client表

插入示例数据：

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (1,'张一','zhangyi@huawei.com','340211199301010001','18815650001','gaussdb\_001');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (2,'张二','zhanger@huawei.com','340211199301010002','18815650002','gaussdb\_002');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (3,'张三','zhangsan@huawei.com','340211199301010003','18815650003','gaussdb\_003');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (4,'张四','zhangsi@huawei.com','340211199301010004','18815650004','gaussdb\_004');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (5,'张五','zhangwu@huawei.com','340211199301010005','18815650005','gaussdb\_005');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (6,'张六','zhangliu@huawei.com','340211199301010006','18815650006','gaussdb\_006');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (7,'张七','zhangqi@huawei.com','340211199301010007','18815650007','gaussdb\_007');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (8,'张八','zhangba@huawei.com','340211199301010008','18815650008','gaussdb\_008');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (9,'张九','zhangjiu@huawei.com','340211199301010009','18815650009','gaussdb\_009');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (10,'李一','liyi@huawei.com','340211199301010010','18815650010','gaussdb\_010');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (11,'李二','lier@huawei.com','340211199301010011','18815650011','gaussdb\_011');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (12,'李三','lisan@huawei.com','340211199301010012','18815650012','gaussdb\_012');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (13,'李四','lisi@huawei.com','340211199301010013','18815650013','gaussdb\_013');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (14,'李五','liwu@huawei.com','340211199301010014','18815650014','gaussdb\_014');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (15,'李六','liliu@huawei.com','340211199301010015','18815650015','gaussdb\_015');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (16,'李七','liqi@huawei.com','340211199301010016','18815650016','gaussdb\_016');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (17,'李八','liba@huawei.com','340211199301010017','18815650017','gaussdb\_017');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (18,'李九','lijiu@huawei.com','340211199301010018','18815650018','gaussdb\_018');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (19,'王一','wangyi@huawei.com','340211199301010019','18815650019','gaussdb\_019');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (20,'王二','wanger@huawei.com','340211199301010020','18815650020','gaussdb\_020');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (21,'王三','wangsan@huawei.com','340211199301010021','18815650021','gaussdb\_021');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (22,'王四','wangsi@huawei.com','340211199301010022','18815650022','gaussdb\_022');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (23,'王五','wangwu@huawei.com','340211199301010023','18815650023','gaussdb\_023');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (24,'王六','wangliu@huawei.com','340211199301010024','18815650024','gaussdb\_024');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (25,'王七','wangqi@huawei.com','340211199301010025','18815650025','gaussdb\_025');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (26,'王八','wangba@huawei.com','340211199301010026','18815650026','gaussdb\_026');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (27,'王九','wangjiu@huawei.com','340211199301010027','18815650027','gaussdb\_027');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (28,'钱一','qianyi@huawei.com','340211199301010028','18815650028','gaussdb\_028');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (29,'钱二','qianer@huawei.com','340211199301010029','18815650029','gaussdb\_029');

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (30,'钱三','qiansan@huawei.com','340211199301010030','18815650030','gaussdb\_030');

查询插入结果

select count(\*) from client;

2.初始化bank\_card表：

插入示例数据：

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000001','信用卡',1);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000002','信用卡',3);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000003','信用卡',5);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000004','信用卡',7);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000005','信用卡',9);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000006','信用卡',10);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000007','信用卡',12);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000008','信用卡',14);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000009','信用卡',16);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000010','信用卡',18);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000011','储蓄卡',19);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000012','储蓄卡',21);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000013','储蓄卡',7);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000014','储蓄卡',23);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000015','储蓄卡',24);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000016','储蓄卡',3);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000017','储蓄卡',26);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000018','储蓄卡',27);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000019','储蓄卡',12);

INSERT INTO bank\_card(b\_number,b\_type,b\_c\_id) VALUES ('6222021302020000020','储蓄卡',29);

查询插入结果

select count(\*) from bank\_card;

3.初始化finances\_product

插入示例数据：

INSERT INTO finances\_product(p\_name,p\_id,p\_description,p\_amount,p\_year) VALUES ('债券',1,'以国债、金融债、央行票据、企业债为主要投资方向的银行理财产品。',50000,6);

INSERT INTO finances\_product(p\_name,p\_id,p\_description,p\_amount,p\_year) VALUES ('信贷资产',2,'一般指银行作为委托人将通过发行理财产品募集资金委托给信托公司，信托公司作为受托人成立信托计划，将信托资产购买理财产品发售银行或第三方信贷资产。',50000,6);

INSERT INTO finances\_product(p\_name,p\_id,p\_description,p\_amount,p\_year) VALUES ('股票',3,'与股票挂钩的理财产品。目前市场上主要以港股挂钩居多',50000,6);

INSERT INTO finances\_product(p\_name,p\_id,p\_description,p\_amount,p\_year) VALUES ('大宗商品',4,'与大宗商品期货挂钩的理财产品。目前市场上主要以挂钩黄金、石油、农产品的理财产品居多。',50000,6);

查询插入结果

select count(\*) from finances\_product;

4.初始化insurance表

插入示例数据：

INSERT INTO insurance(i\_name,i\_id,i\_amount,i\_person,i\_year,i\_project) VALUES ('健康保险',1,2000,'老人',30,'平安保险');

INSERT INTO insurance(i\_name,i\_id,i\_amount,i\_person,i\_year,i\_project) VALUES ('人寿保险',2,3000,'老人',30,'平安保险');

INSERT INTO insurance(i\_name,i\_id,i\_amount,i\_person,i\_year,i\_project) VALUES ('意外保险',3,5000,'所有人',30,'平安保险');

INSERT INTO insurance(i\_name,i\_id,i\_amount,i\_person,i\_year,i\_project) VALUES ('医疗保险',4,2000,'所有人',30,'平安保险');

INSERT INTO insurance(i\_name,i\_id,i\_amount,i\_person,i\_year,i\_project) VALUES ('财产损失保险',5,1500,'中年人',30,'平安保险');

查询插入结果

select count(\*) from insurance;

5.初始化fund表：

插入示例数据：

INSERT INTO fund(f\_name,f\_id,f\_type,f\_amount,risk\_level,f\_manager) VALUES ('股票',1,'股票型',10000,'高',1);

INSERT INTO fund(f\_name,f\_id,f\_type,f\_amount,risk\_level,f\_manager) VALUES ('投资',2,'债券型',10000,'中',2);

INSERT INTO fund(f\_name,f\_id,f\_type,f\_amount,risk\_level,f\_manager) VALUES ('国债',3,'货币型',10000,'低',3);

INSERT INTO fund(f\_name,f\_id,f\_type,f\_amount,risk\_level,f\_manager) VALUES ('沪深300指数',4,'指数型',10000,'中',4);

查询插入结果

select count(\*) from fund;

6.初始化property表：

插入示例数据：

INSERT INTO property(pro\_id,pro\_c\_id,pro\_pif\_id,pro\_type,pro\_status,pro\_quantity,pro\_income,pro\_purchase\_time) VALUES (1,5,1,1,'可用',4,8000,'2018-07-01');

INSERT INTO property(pro\_id,pro\_c\_id,pro\_pif\_id,pro\_type,pro\_status,pro\_quantity,pro\_income,pro\_purchase\_time) VALUES (2,10,2,2,'可用',4,8000,'2018-07-01');

INSERT INTO property(pro\_id,pro\_c\_id,pro\_pif\_id,pro\_type,pro\_status,pro\_quantity,pro\_income,pro\_purchase\_time) VALUES (3,15,3,3,'可用',4,8000,'2018-07-01');

INSERT INTO property(pro\_id,pro\_c\_id,pro\_pif\_id,pro\_type,pro\_status,pro\_quantity,pro\_income,pro\_purchase\_time) VALUES (4,20,4,1,'冻结',4,8000,'2018-07-01');

查询插入结果

select count(\*) from property;

* 手工插入一条数据

当C银行有新的信息需要加入数据库时，系统需要在对应的数据表中手动插入一条新的数据。因此，针对主键属性定义的场景，介绍如何手动插入一条数据。

1.在金融数据库的客户信息表中添加一个客户的信息。（属性冲突的场景）

c\_id\_card和c\_phone非唯一

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (31,'李丽','lili@huawei.com','340211199301010005','18815650005','gaussdb\_005');

错误信息如下：

执行失败，失败原因：ERROR: dn\_6001\_6002\_6003: duplicate key value violates unique constraint "client\_c\_id\_card\_key"

Detail: Key (c\_id\_card)=(340211199301010005 ) already exists.

备注：由于在表的创建过程中，实验定义了c\_id\_card和c\_phone为唯一且非空（UNIQUE NOT NULL），所以当表中存在时，插入数据失败。

2.在金融数据库的客户信息表中添加一个客户的信息。（插入成功的场景）

插入成功的示例

INSERT INTO client(c\_id,c\_name,c\_mail,c\_id\_card,c\_phone,c\_password) VALUES (31,'李丽','lili@huawei.com','340211199301010031','18815650031','gaussdb\_031');

**2.4添加约束**

1.在理财产品表、保险信息表和基金信息表中，都存在金额这个属性，在现实生活中，金额不会存在负数。因此针对表中金额的属性，增加大于0的约束条件。

2.为finances\_product表的p\_amount列添加大于等于0的约束

ALTER table finances\_product ADD CONSTRAINT c\_p\_mount CHECK (p\_amount >=0);

3.尝试手工插入一条金额小于0的记录

INSERT INTO finances\_product(p\_name,p\_id,p\_description,p\_amount,p\_year) VALUES ('信贷资产',10,'一般指银行作为委托人将通过发行理财产品募集资金委托给信托公司，信托公司作为受托人成立信托计划，将信托资产购买理财产品发售银行或第三方信贷资产。',-10,6);

执行失败，失败原因：ERROR: dn\_6001\_6002\_6003: new row for relation "finances\_product" violates check constraint "c\_p\_mount"

Detail: Failing row contains (信贷资产, 10, 一般指银行作为委托人将通过发行理财产品募集..., -10, 6).

4.向fund表添加约束

为fund表的f\_amount列添加大于等于0的约束

ALTER table fund ADD CONSTRAINT c\_f\_mount CHECK (f\_amount >=0);

5.向insurance表添加约束

为insurance表的i\_amount列添加大于等于0的约束

ALTER table insurance ADD CONSTRAINT c\_i\_mount CHECK (i\_amount >=0);

**2.5查询数据**

在本章的金融数据库实验中，主要目的是为了让读者学习到更深一层的查询操作，让学习者能够更深入的去了解openGauss数据库的复杂操作。

1.单表查询。

查询银行卡信息表

SELECT b\_number,b\_type FROM bank\_card;

2.条件查询。

查询资产信息中‘可用’的资产数据

select \* from property where pro\_status='可用';

3.聚合查询。

* 查询用户表中有多少个用户

SELECT count(\*) FROM client;

* 查询银行卡信息表中，储蓄卡和信用卡的个数

SELECT b\_type,COUNT(\*) FROM bank\_card GROUP BY b\_type;

* 查询保险信息表中，保险金额的平均值

SELECT AVG(i\_amount) FROM insurance;

* 查询保险信息表中保险金额的最大值和最小值所对应的险种和金额

select i\_name,i\_amount from insurance where i\_amount in (select max(i\_amount) from insurance)

union

select i\_name,i\_amount from insurance where i\_amount in (select min(i\_amount) from insurance);

4.连接查询。

（1）半连接

查询用户编号在银行卡表中出现的用户的编号，用户姓名和身份证。

SELECT c\_id,c\_name,c\_id\_card FROM client WHERE EXISTS (SELECT \* FROM bank\_card WHERE client.c\_id = bank\_card.b\_c\_id);

备注：半连接是一种特殊的连接类型，在SQL中没有指定的关键字，通过在WHERE后面使用IN或EXISTS子查询实现。当IN/EXISTS右侧的多行满足子查询的条件时，主查询也只返回一行与EXISTS子查询匹配的行，而不是复制左侧的行。

（2）反连接

查询银行卡号不是‘622202130202000001\*’（\*表示未知）的用户的编号，姓名和身份证。

SELECT c\_id,c\_name,c\_id\_card FROM client WHERE c\_id NOT IN (SELECT b\_c\_id FROM bank\_card WHERE b\_number LIKE '622202130202000001\_');

备注：反连接是一种特殊的连接类型，在SQL中没有指定的关键字，通过在WHERE后面使用 NOT IN或NOT EXISTS子查询实现。返回所有不满足条件的行。这个关系的概念跟半连接相反。

5.子查询。

通过子查询，查询保险产品中保险金额大于平均值的保险名称和适用人群。

SELECT i1.i\_name,i1.i\_amount,i1.i\_person FROM insurance i1 WHERE i\_amount > (SELECT avg(i\_amount) FROM insurance i2);

6.ORDER BY和GROUP BY。

（1）ORDER BY子句

按照降序查询保险编号大于2的保险名称，保额和适用人群。

SELECT i\_name,i\_amount,i\_person FROM insurance WHERE i\_id>2 ORDER BY i\_amount DESC;

（2）GROUP BY子句

查询各保险信息总数，按照p\_year分组

SELECT p\_year,count(p\_id) FROM finances\_product GROUP BY p\_year;

7.HAVING和WITH AS。

（1）HAVING子句

查询保险金额统计数量等于2的适用人群数。

SELECT i\_person,count(i\_amount) FROM insurance GROUP BY i\_person HAVING count(i\_amount)=2;

备注：HAVING子句依附于GROUP BY子句而存在。

（2）WITH AS子句

使用WITH AS查询基金信息表

WITH temp AS (SELECT f\_name,ln(f\_amount) FROM fund ORDER BY f\_manager DESC) SELECT \* FROM temp;

备注：该语句为定义一个SQL片段，该SQL片段会被整个SQL语句用到。

可以使SQL语句的可读性更高。存储SQL片段的表与基本表不同，是一个虚表。数据库不存放对应的定义和数据，这些数据仍存放在原来的基本表中。若基本表中的数据发生变化，从存储SQL片段的表中查询出的数据也随之改变。

**2.6视图管理**

视图是一个**虚拟表**，是sql的查询结果，其内容由查询定义。对于来自多张关联表的复杂查询，就不得不使用十分复杂的SQL语句进行查询，造成极差的体验感。使用视图之后，可以极大的简化操作，使用视图不需要关心相应表的结构、关联条件等。

1.创建视图。

针对“查询用户编号在银行卡表中出现的用户的编号，用户姓名和身份证” 的查询，创建视图

CREATE VIEW v\_client as SELECT c\_id,c\_name,c\_id\_card FROM client WHERE EXISTS (SELECT \* FROM bank\_card WHERE client.c\_id = bank\_card.b\_c\_id);

使用视图进行查询

SELECT \* FROM v\_client;

2.修改视图。

修改视图，在原有查询的基础上，过滤出信用卡用户

CREATE OR REPLACE VIEW v\_client as SELECT c\_id,c\_name,c\_id\_card FROM client WHERE EXISTS (SELECT \* FROM bank\_card WHERE client.c\_id = bank\_card.b\_c\_id and bank\_card.b\_type='信用卡');

使用视图进行查询

select \* from v\_client;

3.删除视图。

将v\_client视图删除，删除视图不影响基表。

DROP VIEW v\_client;

**2.7索引管理**

1.创建索引。

* 在普通表property上创建索引

CREATE INDEX idx\_property ON property(pro\_c\_id DESC,pro\_income,pro\_purchase\_time);

2.重命名索引。

* 在普通表property上重建及重命名索引

3.重建索引

DROP INDEX idx\_property;

CREATE INDEX idx\_property ON property(pro\_c\_id DESC,pro\_income,pro\_purchase\_time);

4.重命名索引

ALTER INDEX idx\_property RENAME TO idx\_property\_temp;

5.删除索引。

* 删除索引idx\_property\_temp

DROP INDEX idx\_property\_temp;

**2.8数据的修改和删除**

1.修改数据。

* 修改/更新银行卡信息表中b\_c\_id小于10和客户信息表中c\_id相同的记录的b\_type字段。

查看表数据

SELECT \* FROM bank\_card where b\_c\_id<10 ORDER BY b\_c\_id;

2.开始更新数据：

UPDATE bank\_card SET bank\_card.b\_type='借记卡' from client where bank\_card.b\_c\_id = client.c\_id and bank\_card.b\_c\_id<10;

3.重新查询数据情况

SELECT \* FROM bank\_card ORDER BY b\_c\_id;

4.删除指定数据。

* 删除基金信息表中编号小于3的行。

删除前查询结果

SELECT \* FROM fund;

5.开始删除数据：

DELETE FROM fund WHERE f\_id<3;

6.查询删除结果

SELECT \* FROM fund;

**2.9新用户的创建和授权**

在本章中，假设C银行的某新员工想要在自己的用户下去访问sys用户下的金融数据库，则该员工需要向sys申请添加相关权限，具体操作如下：

1.连接数据库后，进入SQL命令界面。创建用户dbuser，密码为Fusion#3demo。

CREATE USER dbuser IDENTIFIED BY 'Fusion#3demo';

2.给用户dbuser授予finance数据库下对象的查询和插入权限。

GRANT SELECT,INSERT ON root.bank\_card TO dbuser;

GRANT ALL ON SCHEMA root to dbuser;

# 实验6 使用JDBC连接FusionDB数据库

一、实验介绍

**1.1 实验内容**

本节主要介绍如何使用JDBC连接FusionDB。支持标准JDBC 4.0的特性、ODBC 3.5特性。支持JDK 1.8及以上版本。

**1.2 知识点**

* 连接环境准备
* JDK下载安装
* 连接FusionDB并执行Java代码、

二、实验内容

**2.1****连接环境准备**

下载连接FusionDB的Java驱动包。

注：驱动包“FusionDB-22.0.3-JDBC.tar.gz”在前面提供的网盘链接里，直接下载即可

根据FusionDB的版本选择相应的驱动包版本，包名为FusionDB-22.0.3-JDBC.tar.gz，解压后文件夹里包含两个文件为“opengauss-jdbc-3.0.0.jar”和“postgresql.jar”。

**2.2 JDK下载安装**

**注：已经安装好的同学不用重复安装**

下载JDK：https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html

**1.检查当前系统中是否已安装JDK**

命令：java -version：查看JDK安装版本，命令执行后显示如下，说明本系统默认已经安装JDK1.8.0\_242

$ java -version

openjdk version "1.8.0\_242"

OpenJDK Runtime Environment (build 1.8.0\_242-b08)

OpenJDK 64-Bit Server VM (build 25.242-b08, mixed mode)

**2.安装JDK**

如果系统还未安装JDK，请参照下述步骤安装

1. 如果系统已安装yum，使用如下命令安装：

yum install -y java-1.8.0-openjdk.x86\_64

1. 下载并安装请参考：https://www.jianshu.com/p/f000e05f3512

**2.3连接FusionDB并执行Java代码**

**1.使用Java程序连接数据库并进行查询。**

代码如下，（注：USER = "dbuser" 连接数据库的用户及密码PASS = "Fusion#3demo"）

创建FusionDBDemo.java文件，文件内容如下，注意红字部分要按照实际情况进行替换：

import java.sql.\*;

public class FusionDBDemo{

static final String JDBC\_DRIVER = "org.postgresql.Driver";

static final String DB\_URL = "jdbc:postgresql:// IP:26000/finance";

// Using your username and password

static final String USER = " dbuser";

static final String PASS = " Fusion#3demo";

public static void main(String[] args) {

Connection conn = null;

Statement stmt = null;

try{

// Register JDBC Driver

Class.forName(JDBC\_DRIVER);

// Open the link

System.out.println("Connecting the database...");

conn = DriverManager.getConnection(DB\_URL,USER,PASS);

// Execute the query

System.out.println("Statement object...");

stmt = conn.createStatement();

String sql;

sql = "SELECT c\_id FROM Client ";

ResultSet rs = stmt.executeQuery(sql);

// Print the data

while(rs.next()){

int c\_id = rs.getInt("c\_id ");

System.out.print("Client ID: " + c\_id);

}

// Close the connection

rs.close();

stmt.close();

conn.close();

}catch(SQLException se){

// JDBC Exception

se.printStackTrace();

}catch(Exception e){

// Class.forName Error

e.printStackTrace();

}finally{

// Close the resource

try{

if(stmt!=null) stmt.close();

}catch(SQLException se2){

}

try{

if(conn!=null) conn.close();

}catch(SQLException se){

se.printStackTrace();

}

}

System.out.println("Goodbye!");

}

}

2.先对Java程序进行编译（进入Java程序的目录）

javac -encoding utf-8 -cp postgresql.jar FusionDBDemo.java

再执行以下命令

java -cp postgresql.jar FusionDBDemo

3.执行结果。

执行结果如下：

Connecting the database...

…

Statement object....

Client ID: 1

…

Goodbye!

# 挑战：数据库设计

假设C银行因为实际业务需要添加一个新对象为理财经理，其与数据库中其他对象存在着以下关系：一个理财经理可以服务多个客户，一个客户只能有一个理财经理。

## 目标

1. 根据C银行新添加的对象关系，画出修改后的关系模式和ER图。
2. 根据修改后的ER图创建新的表格。
3. 插入一些生成数据并设计查询验证新添加的对象和关系。