数据库系统基础设计及应用

Charles

2023 年 5 月 14 日

目录

1 数据库系统 5
1.1 数据库的的概念 5
1.2 数据库系统组成与结构 5
1.3 数据库管理系统 6

2 关系数据模型 6
2.1 数据模型的概念 6
2.1.1 实体与数据 6
2.1.2 数据模型的构造与分类 6
2.2 概念数据模型 7
2.3 逻辑数据模型 7
2.4 关系及关系约束 8
2.4.1 关系 8
2.4.2 关系的性质 8
2.4.3 主键和外键 8
2.4.4 关系模式 8
2.4.5 关系完整性约束 9
2.5 关系运算 9
2.5.1 传统的集合运算 9
2.5.2 专门的关系运算 9
2.5.3 扩充的关系代数运算 9
2.5.4 元组关系演算 10
2.5.5 域关系演算 10
2.6 数据依赖与关系规范化 10
2.6.1 基于主键的范式和 BC 范式 10
2.6.2 多值依赖和第四范式 11
2.6.3 关系规范化的过程与原则 11

3 openGauss 数据库系统 11
3.1 创建华为云服务器 11
3.1.1 登录华为云 11
3.1.2 购买弹性云服务器 ECS（openEuler ARM 操作系统） 11
3.2 修改操作系统配置 12
3.2.1 连接服务器 12
3.2.2 设置字符集参数 12
3.2.3 修改 python 版本并安装 libaio 包 12
3.3 安装 openGauss 数据库 13
3.3.1 下载数据库安装包 13
3.3.2 创建 XML 配置文件 13
3.3.3 初始化安装环境 14
3.3.4 执行安装 15
3.4 使用 openGauss 数据库 16
3.4.1 配置使用环境 16
3.4.2 操作数据库 17

3.5 其他操作 18

4 SQL 语言 18
4.1 SQL 语言的功能及特点 18
4.2 数据定义 19
4.2.1 数据库的创建、修改与删除 19
4.2.2 数据表的创建、修改与删除 19
4.3 数据更新 20
4.4 数据查询 20
4.4.1 单表查询 20
4.4.2 聚合函数与分组查询 21
4.4.3 连接查询 21
4.4.4 子查询 22
4.4.5 查询结果操作 22
4.5 SQL 语言的视图 23
4.6 SQL 语言的索引 23
4.7 openGauss 实现 24

5 数据库完整性 27
5.1 数据库完整性的概念 27
5.2 DBMS 中的数据完整性 28
5.3 触发器 29
5.4 两个实例 30

6 数据库应用程序 32
6.1 存储过程和函数 32
6.2 函数实例 34
6.3 Python 可视化数据库应用系统 36
6.3.1 设置防火墙安全规则 36
6.3.2 设置 DB 加密方式和监听 IP 36
6.3.3 创建数据库用户并设密码 36
6.3.4 Python 编程 37

7 数据库安全性 37
7.1 数据库安全性 37
7.2 OpenGauss 的用户和权限管理 38
7.2.1 用户 38
7.2.2 角色 38
7.2.3 PUBLIC 39
7.2.4 Schema 39
7.2.5 用户权限的设置与回收 39
7.3 事务 40
7.4 备份和恢复 40
7.4.1 SQL 转储（备份数据库） 40
7.4.2 文件系统级备份 41
7.4.3 恢复 41
7.5 安全策略 42

7.5.1 账户安全策略 42
7.5.2 账号有效期 42
7.5.3 密码安全策略 42
7.6 审计 43

8 数据库设计与创建 **44**
8.1 数据库设计方法 44
8.2 数据库设计过程 44
8.2.1 需求分析 44
8.2.2 概念设计 45
8.2.3 逻辑设计 45

1 数据库系统

1.1 数据库的的概念

1. **数据 (Data)**：能够被计算机处理的描述信息（或事物）的符号。
2. **数据库 (DataBase)**：在计算机中长期存储的、有组织、能共享的数据的集合。
3. 数据库中的数据的基本单位是数据元素。
4. **数据库管理系统 (DBMS)**：为数据库的建立、使用和维护而配置的系统软件。
5. **数据库系统 (DBS)**：在计算机系统中引入数据库，由计算机系统、数据库、数据库管理系统（及开发工具）、应用系统、数据库管理人员构成的整个系统。
6. 数据管理系统的发展：
   (1) 人工管理阶段：应用程序管理数据、数据不保存、数据不共享、数据不具有独立性、冗余大。
   (2) 文件系统阶段：操作系统管理数据、可长期保存、可被同种或同类应用程序共享、具有一定的独立性、有很大冗余。
   (3) 数据库系统阶段：数据库管理系统统一管理和控制数据、数据结构化、有组织、可长期存储、可共享、较少的冗余、较大的独立性、易扩展。

1.2 数据库系统组成与结构

1. 组成：
   (1) 硬件，网络
   (2) 数据库：基础
   (3) DBMS：管理，核心
   (4) 应用程序：利用数据库
   (5) 数据库设计人员
   (6) 软件开发、维护人员
   (7) 数据库管理员（DBA）：创建、维护数据库
   (8) 用户
2. **三级模式 (Schema)**：
   (1) **概念模式/模式**
      (a) 数据库中全体数据的逻辑结构和特征的描述，即数据库采用的数据模型。
      (b) 反映数据的类型、限制条件、数据之间的联系等。
      (c) 是最全的数据的组织形式。
      (d) 一个数据库只有一个概念模式。
      (e) 向下与数据库的存储方式无关。
      (f) 向上与应用程序无关。
      (g) 与具体的硬件、软件无关。
      (h) 通常由 DBA 统一组织管理，也称为 DBA 视图。
   (2) **外模式/子模式**
      (a) 是局部数据的逻辑结构和特征的描述。
      (b) 是用户（包括程序员和最终用户）看到的数据模式。
      (c) 与具体的应用需求有关。
      (d) 一个数据库有多个外模式。
      (e) 一个应用程序有一个外模式。
   (3) **内模式/存储模式**
      (a) 是数据的物理结构和存储方式的描述。
      (b) 是数据在数据库内部的表示方式。

    (c) 对一般用户是透明的。
    (d) 影响数据库的性能，DBMS 实现。
    (e) 一个数据库只有一个内模式。
3. **两级映射/映像**：
   (1) **外模式/模式映像**：保证逻辑独立性。
   (2) **模式/内模式映像**：保证物理独立性。
4. 数据库系统体系结构：
   (1) 分时系统环境下的集中式数据库系统
   (2) 微型计算机上的单用户数据库系统
   (3) 网络环境下的客户/服务器数据库系统
   (4) 分布式数据库系统
   (5) 因特网上的数据库

1.3 数据库管理系统

功能：

1. 数据库定义：数据库描述。可从用户、概念和物理三个不同层次出发定义、创建数据库。
2. 数据的组织、存储和管理：分类、结构、存取方式、数据联系。
3. 数据库操纵：接收、分析和执行提出的数据库操作要求，如：检索、插入、删除、更新等。
4. 数据库运行控制：包括执行访问数据库时的安全性、完整性检查以及数据共享的并发控制、故障恢复等，目的是保证数据库的可用性和可靠性。
5. 数据字典：对内模式的文件、索引，外模式的表、数据类型、联系，外模式的视图，以及用户表、用户权限公用程序等对象的描述。
6. 其他：网络通信、数据交换等。

2 关系数据模型

2.1 数据模型的概念

2.1.1 实体与数据

1. **数据**：信息和事物的符号表示。
2. **现实世界**：事物的客观存在。
3. **信息世界**：事物的性质、状态的描述。
4. **实体**：客观事物在信息世界中的描述。
5. **属性**：事物的性质在信息世界中的描述。
6. 信息世界中，实体可有若干属性来描述。
7. **数据世界**：信息数据化。
8. **记录**：每个实体的一组属性。
9. **数据项**：每个属性。
10. 实体之间的关系：
    (1) 一对一联系 (1:1)
    (2) 一对多联系 (1:n)
    (3) 多对多联系 (m:n)

2.1.2 数据模型的构造与分类

1. **模型**：依照原物或计划中的事物的形式做成的物品。

2. **数据模型**：实体及实体间联系的描述。
3. 数据模型的构造：
   (1) 数据结构：是数据模型的基础。
   (2) 数据操作
   (3) 数据约束
4. 数据模型的分类：
   (1) 概念数据模型
      (a) 信息世界中事物的描述。
      (b) 按用户的观点来对数据和信息建模。
      (c) 与 DBMS 无关；与计算机平台无关。
      (d) 是所有数据模型的基础。
   (2) 逻辑数据模型
      (a) 面向数据库系统的模型；是概念模型的进一步抽象。
      (b) 是实体在计算机世界的逻辑描述。
      (c) 按计算机系统的观点对数据建模，用于 DBMS 实现。
      (d) 考虑在计算机中如何表示和组织。
      (e) 但仍与具体的数据库管理系统无关；是一般的逻辑描述；类似于算法。
      (f) 层次模型、网状模型、关系模型。
   (3) 物理数据模型
      (a) 是对数据最底层的抽象，描述数据在系统内部的表示方式和存取方法，在磁盘或磁带上的存储方式和存取方法。
      (b) 是实现方式的描述。
      (c) E-R 模型，扩充 E-R 模型，面向对象模型等。

2.2 概念数据模型

E-R 图：

1. **实体（型）**：用矩形表示，矩形框内写明实体集的名字。
2. **属性**：用椭圆表示，并用无向边将其与相应的实体连接起来。
   属性不能脱离实体，属性是相对实体而言的。
3. **联系**：用菱形表示，菱形内写明联系名，并用无向边分别与有关实体连接起来，同时在无向边旁标上联系的类型（1:1、1:n、n:1 或 m:n）。
   联系也可以有属性。
   如果一个联系具有属性，则这些属性也要用无向边与该联系连接起来。
4. **UML(Unified Modeling Language)**: 统一建模语言。

2.3 逻辑数据模型

1. 层次模型
   (1) 用树型结构表示实体及实体之间的联系。
   (2) 每个结点是一个实体（型）。
   (3) 根无父结点。
   (4) 适合表示 1:n 的联系。
   (5) 信息的查找只能按树型路径查找。
2. 网状模型
   (1) 用网状结构表示实体及实体之间的联系。

(2) 每个结点代表一个记录型。
(3) 每个联系都是一对多的联系。
(4) 结点的联系用记录指针实现。

3. 关系模型
   (1) 将数据组织成由若干行，每行由若干列组成的表格形式。
   (2) 每个实体集是一个关系表示。
   (3) 每个联系也是一个关系表示。

2.4 关系及关系约束

2.4.1 关系

1. **域**：属性所取值的变化范围。
2. 笛卡尔积：$D_1 \times D_2 \times \cdots \times D_n = \{(d_1, d_2, \ldots, d_n) | d_i \in D_i, i = 1, 2, \ldots, n\}$
3. 关系：笛卡尔集的一个子集。
4. 关系数据模型中的关系必须是有限集合。
5. 关系数据模型中需要为各列加上属性名来取消元组的有序性。

2.4.2 关系的性质

1. 每列都是不可再分的基本字段。
2. 列名是唯一的。(列名不能相同)
3. 列是同质的。(类型相同、值域、约束相同)
4. 不同的列可以取值同一个域。
5. 列的次序可任意交换，不会改变关系的意义。
6. 行的次序可任意交换，不会改变关系的意义。
7. 不允许有相同的元组。

2.4.3 主键和外键

1. **候选键 (candidate key)**：可以唯一地决定一个元组的最小属性集。
2. **主属性**：包含在任何一个候选键中的属性。
3. **非主属性**：不包含在候选键中的属性。
4. **超键 (supper key)**：包含冗余属性的属性。
5. **全键**：候选键包含所有属性。
6. **主键 (primary key)**：选定的一个候选键。
7. **实体完整性约束**：在数据库中，每个元组的主键的值不能为空，也不能与其他元组的主键值相同。
8. 键与数据的逻辑意义有关，在 DBMS 中由用户设定
9. 主键的选择应该是那些从不或极少变化的属性。
10. **外键**：设 X 是关系 R 的一个属性组，它并非 R 的主键，但却是另一个关系 S 的主键，则称 X 为 R 关于 S（可为 R 本身）的外键。
11. **主表**：主键所在的表。
12. **从表**：外键所在的表。

2.4.4 关系模式

完整性约束：语义施加于数据上的限制。

1. **关系模式**：用关系名及写在括号中用逗号隔开的属性名表示关系的结构。
2. 关系模式是关系数据结构的描述。

3. 主码用下划线表示，且列在前面。
4. 一般形式：R(A1,A2,A3,A4,...,An)。
5. 关系数据库模式：是关系数据库的型，包括若干域的定义以及在这些域上定义的若干关系模式。

2.4.5 关系完整性约束

1. **实体完整性规则**：每个关系都有一个主键，每个元组的主键值是唯一的。既不空，也不重复。
2. **引用完整性规则**：外键的取值要么为空，要么是主表中相应属性的现有值之一。
3. **用户定义完整性规则**：是针对具体的列、元组的取值要求设置的。

2.5 关系运算

1. **关系运算**：关系代数和关系演算。
2. **关系代数**：以集合代数运算的方法对关系进行操作。
   以关系作输入，结果为关系。
   过程化。
3. **关系演算**：以谓词表达式来描述关系操作的条件和要求。
   非过程化。
4. **谓词表达式**：对事物的性质、类别、关系的形式化描述。

2.5.1 传统的集合运算

1. 并：$R \cup S = \{t|t \in R \vee t \in S\}$
2. 交：$R \cap S = \{t|t \in R \wedge t \in S\}$
3. 差：$R - S = \{t|t \in R \wedge t \notin S\}$
4. 广义笛卡尔积：$R \times S = \{\widehat{t_r t_s}|t_r \in R \wedge t_s \in S\}$

```
关系“并”操作实例：
(select * from stu1) union (select * from stu2)
insert into stu1 select * from stu2
#union自动去除重复   union all 保留重复
关系“交”操作实例：
select * from stu1 where number in(select number from stu2)
(select * from stu1) intersect (select * from stu2)
关系“差”操作实例：
(select * from stu1) except (select * from stu2)
```

2.5.2 专门的关系运算

1. 选择：$\sigma_F = \{t|t \in R \wedge F(t) \quad is \quad Ture\}$
2. 投影：$\prod_X = \{t[X]|t \in R\}$
3. 连接：$R \bowtie_{X\theta Y} S = \{\widehat{t_r t_s}|t_r \in R \wedge t_s \in S \wedge t_r[X]\theta t_s[Y]\}$
   (1) 等值连接：连接条件为相等条件。
   (2) 自然连接：去掉重复属性。
4. 除：$R \div S = \{t_r[X]|t_r \in R \wedge \prod_y(S) \subseteq Y_x\}$
5. 关系代数运算集合 $\{\sigma, \prod, \cup, -, \times\}$ 已被证明是一个**完备集合**。

2.5.3 扩充的关系代数运算

1. 外连接
   (1) 左外连接：连接结果中纳入左边关系的所有元组。
   (2) 右外连接：连接结果中纳入右边关系的所有元组。

(3) 全外连接：连接结果中纳入左、右两边关系的所有元组。

2. 外并

```
左外连接：
SELECT * from teacher left outer JOIN teaching ON teacher.tID=teaching.tID
右外连接：
SELECT * from teaching right outer JOIN teacher ON teaching.tID=teacher.tID
全外连接：
SELECT * from book01 FULL JOIN people01 ON book01.author=people01.name
```

2.5.4 元组关系演算

以元组为变量，查询满足条件的元组中的某些属性。

1. 选择：$\sigma_F(R) = \{t|t \in R\ AND\ F\}$
2. 投影：$\prod_X(R) = \{t|t[X] \in R\}$
3. 差：$R - S = \{t|t \in R\ AND \neg(t \in S)\}$

2.5.5 域关系演算

以域为变量。

域演算表达式：$\{x_1, x_2, \ldots, x_n | \phi(x_1, x_2, \ldots, x_n, \ldots, x_{n+m})\}$

2.6 数据依赖与关系规范化

1. **数据依赖**：通过一个关系中属性间值的相等与否体现出来的数据间的相互关系。
   是现实世界属性间相互联系的抽象，是语义的体现。
2. **函数依赖**：设 R(U) 是一个属性集 U 上的关系模式，X 和 Y 是 U 的子集。若对于 R(U) 的任意一个可能的关系 r，r 中不可能存在两个元组在 X 上的属性值相等而在 Y 上的属性值不等，则 Y 函数依赖于 X，记作 X→Y。
3. 函数依赖用于说明关系中属性之间的相互作用。
4. 函数依赖不是指关系模式 R 的某个或某些关系实例满足的约束条件，而是 R 的所有关系实例均要满足的约束条件。
5. 函数依赖造成的关系缺陷：
   (1) 数据冗余大
   (2) 更新异常
   (3) 插入异常
   (4) 删除异常
6. **非平凡的函数依赖**：X→Y,Y⊆X
7. **平凡的函数依赖**：X→Y,Y⊈X
8. **完全函数依赖**：X→Y,X'↛Y
9. **部分函数依赖**：X→Y,X'→Y
10. **传递函数依赖**：X→Y,Y→Z

2.6.1 基于主键的范式和 BC 范式

1. **范式 (Normal Form)**: 关系需要满足的依赖条件。
2. **1NF**: 关系模式 R 的所有属性都是不可分的基本数据项。
   2NF: 关系模式 R∈1NF，且每个非主属性都完全函数依赖于 R 的码。
   3NF: 关系模式 R∈2NF，且每个非主属性不传递函数依赖于键。
   BCNF: 关系模式 R∈1NF，且每个非平凡的函数依赖 X→Y，都有 X 包含键。

(1) 所有主属性对每个键都是完全函数依赖。

(2) 所有主属性对每个不包含它的键也是完全函数依赖。

(3) 不存在完全函数依赖于非键的属性组。

3. 满足 BCNF 必然满足 3NF,2NF,1NF。

eg:

a. $R(X,Y,Z), XY \rightarrow Z \Rightarrow BCNF$

b. $R(X,Y,Z), Y \rightarrow Z, XZ \rightarrow Y \Rightarrow 3NF$

c. $R(X,Y,Z), Y \rightarrow Z, Y \rightarrow X, X \rightarrow YZ \Rightarrow BCNF$

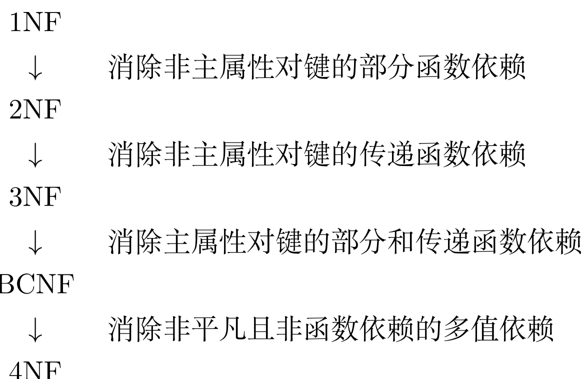
d. $R(X,Y), X \rightarrow Y, X \rightarrow Z \Rightarrow BCNF$

e. $R(W,X,Y,Z), X \rightarrow Z, WX \rightarrow Y \Rightarrow 1NF$

2.6.2 多值依赖和第四范式

1. **多值依赖**：对于给定的 X 值，对应的 Y 是一组数值，且对每个 X 值都成立，则 Y 多值依赖于 X，记作 $X \rightarrow\rightarrow Y$。
2. 关系模式 $R \in 1NF$，且每个非平凡多值依赖 $X \rightarrow\rightarrow Y$, 都有 X 包含键。

2.6.3 关系规范化的过程与原则



3 openGauss 数据库系统

1. openGauss 是关系型数据库，采用客户端/服务器，单进程多线程架构，支持单机和一主多备部署方式，备机可读，支持双机高可用和读扩展。
2. 产品特点：openGauss 相比其他开源数据库主要有复合应用场景、高性能和高可用等产品特点。
3. 软件架构：openGauss 主要包含了 openGauss 服务器，客户端驱动，OM 等模块。

3.1 创建华为云服务器

3.1.1 登录华为云

1. 进入华为云官网`https://www.huaweicloud.com/`，单击登录。
2. 输入账号名和密码，单击登录（如果还没有注册，单击免费注册，按步骤注册后进行登录）。

3.1.2 购买弹性云服务器 ECS（openEuler ARM 操作系统）

1. 在华为云主页点击“产品”，选择 " 精选推荐 " 下的 " 计算 "，再选择 " 弹性云服务器 ECS "。进入弹性云服务器 ECS 购买界面。单击“立即购买”。
2. 自定义购买进行基础配置如下：

   (1) 计费模式：按需计费

(2) 区域：华北-北京四

(3) CPU 架构：鲲鹏计算

(4) 规格：最新系列 2vCPUs|4GiB

(5) 镜像：openEuler 20.03 64bit with ARM(40GB)

其余默认即可，单击 " 下一步网络配置 " 。

3. 步骤 3 自定义购买进行网路配置如下：

   (1) 网络：vpc-default（192.168.0.0/16）

   (2) 弹性公网 IP：现在购买

   (3) 公网带宽：按流量计费

   (4) 带宽大小：5

4. 自定义购买进行高级配置。记住用户名为 root，然后输入自定义密码和确认密码，其余默认即可，单击 " 下一步确认设置 " 。

5. 确认设置信息，尤其是配置费用，然后勾选协议 " 我已经阅读并同意《镜像免责声明》 " ，点击立即购买。

6. 查看云服务器列表。等待数分钟后，状态列显示 " 运行中 " 。

   恭喜您，购买成功！

7. 记下云服务器名称、云服务器密码、公网 IP 和私网 IP。

3.2 修改操作系统配置

3.2.1 连接服务器

运行 putty，在 “Host Name(or IP address)” 一栏输入公网 IP，并单击 “Open”。

3.2.2 设置字符集参数

1. 在/etc/profile 文件中添加 " export LANG= en_US.UTF-8 " 。

```
[root@ecs-c9bf ~]# cat >>/etc/profile<<EOF
> export LANG=en_US.UTF-8
> EOF
```

2. 输入如下命令，使配置修改生效。

```
[root@ecs-c9bf ~]# source /etc/profile
```

3.2.3 修改 python 版本并安装 libaio 包

之后安装过程中 openGauss 用户互信，openEuler 服务器需要用到 Python-3.7.x 命令，但是默认 Python 版本为 Python-2.7.x，所以需要切换 Python 版本。

1. 进入/usr/bin 目录。

```
[root@ecs-c9bf ~]# cd /usr/bin
```

2. 备份 python 文件。

```
[root@ecs-c9bf bin]# mv python python.bak
```

3. 建立 Python3 软连接。

```
[root@ecs-c9bf bin]# ln -s python3 /usr/bin/python
```

4. 验证 Python 版本。

```
[root@ecs-c9bf bin]# python -V
```

显示如下：

```
Python 3.7.4
```

5. Python 版本切换成功，后续安装需要 libaio 包，下载进行安装。

```
[root@ecs-c9bf ~]# yum install libaio* -y
```

3.3 安装 openGauss 数据库

3.3.1 下载数据库安装包

1. 以 root 用户登录待安装 openGauss 的主机后，按规划创建存放安装包的目录。

```
[root@ecs-c9bf bin]# mkdir -p /opt/software/openGauss
[root@ecs-c9bf bin]# chmod 755 -R /opt/software
```

2. 切换到安装目录。

```
[root@ecs-c9bf bin]# cd /opt/software/openGauss
```

3. 使用 wget 下载安装包。

```
[root@ecs-c9bf openGauss]# wget  https://opengauss.obs.cn-south-1.myhuaweicloud.com/2.0.0/arm/
    openGauss-2.0.0-openEuler-64bit-all.tar.gz
```

下载成功后显示如下：

```
...
2021-06-14 13:57:23 (9.33 MB/s) - 'openGauss-2.0.0-openEuler-64bit-all.tar.gz' saved
    [58468915/58468915]
```

3.3.2 创建 XML 配置文件

安装 openGauss 前需要创建 XML 文件。XML 文件包含部署 openGauss 的服务器信息、安装路径、IP 地址以及端口号等。用于告知 openGauss 如何部署。用户需根据不同场合配置对应的 XML 文件。

1. 以 root 用户登录待安装 openGauss 的主机后，切换到存放安装包的目录。

```
[root@ecs-c9bf bin]# cd /opt/software/openGauss
```

2. 创建 XML 配置文件，用于数据库安装。

```
[root@ecs-c9bf openGauss]# vi clusterconfig.xml
```

3. 单击 " i " 进入 INSERT 模式，添加以下文本。

```
<?xml version="1.0" encoding="UTF-8"?>
<ROOT>
    <CLUSTER>
        <PARAM name="clusterName" value="dbCluster" />
        <PARAM name="nodeNames" value="ecs-c9bf" />
        <PARAM name="backIp1s" value="192.168.0.58"/>
        <PARAM name="gaussdbAppPath" value="/opt/gaussdb/app" />
        <PARAM name="gaussdbLogPath" value="/var/log/gaussdb" />
        <PARAM name="gaussdbToolPath" value="/opt/huawei/wisequery" />
        <PARAM name="corePath" value="/opt/opengauss/corefile"/>
        <PARAM name="clusterType" value="single-inst"/>
    </CLUSTER>

    <DEVICELIST>

        <DEVICE sn="1000001">
            <PARAM name="name" value="ecs-c9bf"/>
            <PARAM name="azName" value="AZ1"/>
            <PARAM name="azPriority" value="1"/>
            <PARAM name="backIp1" value="192.168.0.58"/>
```

```
            <PARAM name="sshIp1" value="192.168.0.58"/>

        <!--dbnode-->
        <PARAM name="dataNum" value="1"/>
        <PARAM name="dataPortBase" value="26000"/>
        <PARAM name="dataNode1" value="/gaussdb/data/db1"/>
        </DEVICE>
    </DEVICELIST>
</ROOT>
```

其中，红色部分需自行替换，" ecs-c9bf "是"弹性云服务器的名称"，" 192.168.0.58 "是"私有 IP 地址"，其他 value 值可以不进行修改。

4. 单击" Esc "退出 INSERT 模式，然后输入":wq "后回车退出编辑并保存文本。

3.3.3 初始化安装环境

创建完 openGauss 配置文件后，在执行安装前，为了后续能以最小权限进行安装及 openGauss 管理操作，保证系统安全性，需要运行安装前置脚本 gs_preinstall，准备好安装用户及环境。

1. 修改 performance.sh 文件。

```
[root@ecs-c9bf openGauss]# vi /etc/profile.d/performance.sh
```

单击" i "，进入 INSERT 模式。用 # 注释 sysctl -w vm.min_free_kbytes=112640 &> /dev/null 这行内容。

```
CPUNO=`cat /proc/cpuinfo|grep processor|wc -l`
export GOMP_CPU_AFFINITY=0-$[CPUNO - 1]

#sysctl -w vm.min_free_kbytes=112640 &> /dev/null
sysctl -w vm.dirty_ratio=60 &> /dev/null
sysctl -w kernel.sched_autogroup_enabled=0 &> /dev/null
```

按" Esc "键，退出 INSERT 模式，然后输入":wq "后回车，保存退出。

2. 为确保 openssl 版本正确，执行预安装前加载安装包中 lib 库。

```
[root@ecs-c9bf openGauss]# vi /etc/profile
```

单击 i，进入 INSERT 模式，在文件的底部添加如下代码，加载安装包中 lib 库。

```
export packagePath=/opt/software/openGauss
export LD_LIBRARY_PATH=$packagePath/script/gspylib/clib:$LD_LIBRARY_PATH
```

按下" Esc "退出 INSERT 模式，输入":wq "后回车，保存后退出。
配置完成后，使设置生效。

```
[root@ecs-c9bf openGauss]# source /etc/profile
```

3. 在安装包所在的目录下，解压安装包。

```
[root@ecs-c9bf openGauss]# cd /opt/software/openGauss
```

解压 openGauss-2.0.0-openEuler-64bit-all.tar.gz 包。

```
[root@ecs-c9bf openGauss]# tar -zxvf  openGauss-2.0.0-openEuler-64bit-all.tar.gz
```

解压 openGauss-2.0.0-openEuler-64bit-om.tar.gz 包。

```
[root@ecs-c9bf openGauss]# tar -zxvf  openGauss-2.0.0-openEuler-64bit-om.tar.gz
```

解压后，执行 ls 命令查看内容如下：

```
[root@ecs-c9bf openGauss]# ls
clusterconfig.xml                            openGauss-Package-bak_392c0438.tar.gz
lib                                          script
openGauss-2.0.0-openEuler-64bit-all.tar.gz   simpleInstall
```

```
openGauss-2.0.0-openEuler-64bit-om.sha256   upgrade_sql.sha256
openGauss-2.0.0-openEuler-64bit-om.tar.gz   upgrade_sql.tar.gz
openGauss-2.0.0-openEuler-64bit.sha256      version.cfg
openGauss-2.0.0-openEuler-64bit.tar.bz2
```

安装包解压后，会在/opt/software/openGauss 路径下自动生成 script 子目录，并且在 script 目录下生成 gs_preinstall 等各种 OM 工具脚本。

4. 使用 gs_preinstall 准备好安装环境，切换到 gs_preinstall 命令所在目录。

```
[root@ecs-c9bf openGauss]# cd /opt/software/openGauss/script/
```

5. 执行 ls 命令查看 script 中内容。

```
[root@ecs-c9bf openGauss]# cd /opt/software/openGauss/script/
[root@ecs-c9bf script]# ls
gs_backup   gs_checkperf  gs_om            gspylib     gs_uninstall   __init__.py
gs_check    gs_collector  gs_postuninstall gs_ssh      gs_upgradectl  killall
gs_checkos  gs_install    gs_preinstall    gs_sshexkey impl           local
```

6. 采用交互模式执行，并在执行过程中会创建 openGauss omm 用户互信。

```
[root@ecs-c9bf script]# python gs_preinstall -U omm -G dbgrp -X /opt/software/openGauss/
    clusterconfig.xml
```

在执行过程中，用户根据提示选择是否创建互信，填写 yes。
此时会创建操作系统 omm 用户，并对 omm 创建 trust 互信，并要求设置密码，设置为 Admin@123（建议用户自定义设置密码）。

```
Are you sure you want to create the user[omm] and create trust for it (yes/no)? yes
Please enter password for cluster user.
Password: 此处输入密码时，屏幕上不会有任何反馈。
Please enter password for cluster user again.
Password: 此处输入密码时，屏幕上不会有任何反馈。
Successfully created [omm] user on all nodes.
```

成功后显示为：

```
...
Setting finish flag.
Successfully set finish flag.
Preinstallation succeeded.
```

3.3.4 执行安装

1. 修改文件权限。

```
[root@ecs-c9bf script]# chmod -R 755 /opt/software/openGauss/script
```

2. 登录到 openGauss 的主机，并切换到 omm 用户。

```
[root@ecs-c9bf script]# su - omm
```

3. 使用 gs_install 安装 openGauss。执行以下命令：

```
[omm@ecs-c9bf ~]$gs_install -X /opt/software/openGauss/clusterconfig.xml --gsinit-parameter="--
    encoding=UTF8"  --dn-guc="max_process_memory=4GB"  --dn-guc="shared_buffers=256MB" --dn-
    guc="bulk_write_ring_size=256MB" --dn-guc="cstore_buffers=16MB"
```

在执行过程中，用户需根据提示输入数据库管理员 omm 用户的密码，密码具有一定的复杂度，为保证用户正常使用该数据库，请记住输入的数据库密码。

```
encrypt cipher and rand files for database.
Please enter password for database: 此处输入密码时，屏幕上不会有任何反馈。
Please repeat for database: 此处输入密码时，屏幕上不会有任何反馈。
begin to create CA cert files
```

成功后显示为：

```
...
Successfully deleted instances from all nodes.
Checking node configuration on all nodes.
Initializing instances on all nodes.
Updating instance configuration on all nodes.
Check consistence of memCheck and coresCheck on database nodes.
Configuring pg_hba on all nodes.
Configuration is completed.
Successfully started cluster.
Successfully installed application.
end deploy.
```

3.4 使用 openGauss 数据库

3.4.1 配置使用环境

1. 在数据库主节点服务器上，切换至 omm 操作系统用户环境。

```
[root@ecs-c9bf script]# su - omm
```

2. 查看服务是否启动。

```
[omm@ecs-9a68 ~]$ gs_om -t status
-----------------------------------------------------------------------
cluster_state   : Normal
redistributing  : No
-----------------------------------------------------------------------
```

3. 启动数据库服务。

```
[omm@ecs-c9bf ~]$ gs_om -t start
Starting cluster.
=========================================
=========================================
Successfully started.
```

4. 连接数据库。

```
[omm@ecs-c9bf ~]$ gsql -d postgres -p 26000 -r
```

成功后显示为：

```
gsql ((openGauss 2.0.0 build 290d125f) compiled at 2021-03-31 02:59:43 commit 2143 last mr 131
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

postgres=#
```

5. （可选）修改数据库 omm 用户密码。

```
postgres=# alter role omm identified by '新密码' replace '原密码';
```

成功后显示如下：

```
ALTER ROLE
```

6. 创建数据库用户。

```
postgres=# CREATE USER wang WITH PASSWORD "A13561973155a";
```

如上创建了一个用户名为 wang，密码为 A13561973155a 的用户。
成功后显示如下：

```
CREATE ROLE
```

7. 创建数据库。

```
postgres=# CREATE DATABASE stu OWNER wang;
```

成功后显示如下:

```
CREATE DATABASE
```

8. 退出 postgres 数据库。

```
postgres=#  \q
```

9. 使用新用户连接到此数据库。

```
[omm@ecs-c9bf ~]$ gsql -d stu -p 26000 -U wang -W A13561973155a -r
```

成功后显示如下:

```
gsql ((openGauss 2.0.0 build 290d125f) compiled at 2021-03-31 02:59:43 commit 2143 last mr 131
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

stu=>
```

10. 创建名为 wang 的 SCHEMA，并设置 wang 为当前的 schema。

```
stu=> CREATE SCHEMA wang AUTHORIZATION wang;
```

成功后显示如下:

```
CREATE SCHEMA
```

11. 将默认搜索路径设为 wang。

```
stu=> SET search_path TO wang;
```

成功后显示如下:

```
SET
```

3.4.2 操作数据库

1. 创建 student 表。

```
stu=> CREATE TABLE student(
number char(10) PRIMARY KEY,
name varchar(20),
gender varchar(10),
age int);
```

2. 插入数据。

```
 INSERT INTO student values('2226704561','张三','男','18');
 INSERT INTO student values('2229153228','李四','女','18');
```

3. 查询数据。

```
SELECT * FROM student;
```

显示如下:

```
stu=> CREATE TABLE student(number char(10) PRIMARY KEY,name varchar(20),gender v
archar(10),age int);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "student_pkey" fo
r table "student"
CREATE TABLE
stu=> INSERT INTO student values('2226704561','张三','男','18');
INSERT 0 1
stu=> INSERT INTO student values('2229153228','李四','女','18');
INSERT 0 1
stu=> SELECT * FROM student;
   number    | name | gender | age
------------+------+--------+-----
 2226704561 | 张三 | 男     |  18
 2229153228 | 李四 | 女     |  18
(2 rows)

stu=>
```

3.5 其他操作

1. 关闭数据库服务。

```
[omm@ecs-c9bf ~]$ gs_om - t stop
```

2. 退出登录。

```
[omm@ecs-c9bf ~]$ exit
```

3. 关机。

```
[omm@ecs-c9bf ~]$ shutdown -h
```

4. openGauss 中的 gsql 命令

```
postgres=#
\? #显示gsql命令
\l #列出所有数据库
\c <database_name> #切换数据库
\d <table_name> #显示表结构
\di #列出索引
\dn #列出当前数据库中的所有模式（schema）
\dt #列出当前模式下的所有数据表
\du #列出当前数据库中的所有用户和角色
\x #切换扩展模式的显示
\i <file_name> #从指定的文件中执行SQL命令。
\o <file_name> #将查询结果输出到指定的文件
\q #退出gsql工具
```

4 SQL 语言

4.1 SQL 语言的功能及特点

各 DBMS 会对 SQL 做修改和扩充，因而在某个 DBMS 中的 SQL 不完全是标准 SQL。

1. 功能：
   (1) 数据定义 DDL
   (2) 数据操纵 DML
   (3) 数据查询 DQL
   (4) 数据控制 DCL
2. 特点：
   (1) 非过程化。
   (2) 以记录的集合为操作对象，集合输入和输出。
   (3) 提供交互式、嵌入式、可视化操作方式。

3. 使用规则：

(1) 以命令关键词开始，后跟操作对象及一条或多条字句，以说明更多操作细节。

(2) 不区分大小写，一般关键字大写。

(3) 关键字不能分开。

(4) 语句末尾可用分号表示结束。

(5) 字符串定界符用单引号，等于用一个等号。

4.2 数据定义

4.2.1 数据库的创建、修改与删除

1. 创建数据库

```
CREATE DATABASE <database_name>
[IF NOT EXISTS tmp]#判断要创建的数据库是否已经存在
[[OWNER[=]owner_name]]#管理员姓名
[TEMPLATE[=]template]#模板
[ENCODING[=]encoding]#编码
[LC_COLLATE[=]lc_collate]]#字符串比较和排序的规则
[LC_CTYPE[=]lc_ctype]]#字符集的处理规则，包括字符分类、大小写转换、编码和解码等
[TABLESPACE[=]tablespace_name]]#逻辑存储结构
[CONNECTION LIMIT[=]connlimit]#一个数据库用户可以同时打开的连接数量
[IS_TEMPLATE[=]istemplate]];#指示一个数据库是否是模板数据库
```

2. 修改数据库

```
ALTER DATABASE <database_name> RENAME TO new_name;#重命名数据库
ALTER DATABASE <database_name> OWNER TO {new_owner|CURRENT_USER|SESSION_USER};#修改管理员
```

3. 删除数据库

```
DROP DATABASE <database_name>[,<database_name>][,...];
```

4. 注意事项：

(1) 有权限的用户才能创建、修改、删除数据库。

(2) 不能重命名当前使用的数据库。

(3) 不能删除正在连接的数据库。

(4) 不能删除默认安装的三个数据库：postgres、template0、template1。

4.2.2 数据表的创建、修改与删除

1. 创建表

```
CREATE TABLE [IF NOT EXISTS tmp] <table_name>
(<column_name> <data_type> [<column_constraint>],#数据项及列约束
[table_constraint],#表约束
[LIKE source_table],#形式
[,...]);
```

2. 数据类型：

(1) 逻辑型：bool, boolean

(2) 字符：char(n)（定长字符），varchar(n)（变长字符），TEXT（变长文本）

(3) 整数：SMALLINT, int, SERIAL, smallserial, serial, bigserial

(4) 实数：float(n)（n 位精度），real（8 字节实数），numeric(p,s)（精确数，共 p 位，小数 s 位）

(5) 时间：DATE, TIME, TIMESTAMP

(6) JSON：JSON, JSONB

(7) UUID：唯一值类型

(8) 空间数据：box, line, lseg, polygon, inet, macaddr

3. 约束：

```
NOT NULL|NULL#非空/空
CHECK expression#检验条件
DEFAULT default_expr#指定默认值
UNIQUE index_parameter#唯一
SERIAL#自增序列
CONSTRAINT constraint_name#约束命名
PRIMARY KEY index_parameter#主键
PRIMARY KEY (index_parameter1,index_parameter2)#联合主键
FOREIGN KEY index_parameter REFERENCES main_table#外键
```

4. 日期时间：current_date, current_time, current_timestamp, now()

5. 修改表

```
ALTER TABLE <table_name>
{ADD COLUMN <column_name><data_type>[<column_constraint>]|#添加列
DROP COLUMN <column_name>|#删除列
ADD CONSTRAINT <constraint_name> <constraint_type> (<column_name>)|#添加约束
DROP CONSTRAINT <constraint_name> <constraint_type> (<column_name>)|#删除约束
<table_name> RENAME TO <new_table_name>|#重命名表
RENAME <column_name> TO <new_column_name>};#重命名列
```

6. 删除表

```
DROP TABLE <table_name>;
```

4.3 数据更新

1. 插入数据

```
INSERT INTO <table_name> VALUES (<index_parameters>);
```

2. 修改数据

```
UPDATE <table_name> SET <column1>=<expression1> [,<column2>=<expression2>, ...]
[WHERE <condition>];
```

3. 删除数据

```
DELETE FROM <table_name>[WHERE <condition>];
TRUNCATE <table_name>;#删除表中所有数据
```

4.4 数据查询

```
SELECT [ALL|DISTINCT] <aim expression>
FROM<table_name>
[WHERE <condition1>]
[GROUP BY <column_name1>][,...] [HAVING <condition2>]]
[ORDER BY <column_name2>[ASC|DESC]]
```

4.4.1 单表查询

1. 查询数据

```
SELECT * FROM student;#最简查询
SELECT number FROM student;#投影查询
SELECT DISTINCT sclass FROM student;#去掉重复项
SELECT number, name FROM student
ORDER BY name ASC;#结果排序
SELECT name FROM student
WHERE age between 19 and 20;#条件查询
```

2. 运算符:

   (1) +, - , * , /, %

   (2) >, <, >=, <=, <>, = , !=, !<, !>

   (3) AND, OR, NOT

   (4) IN, NOT IN

   (5) BETWEEN a AND b, NOT BETWEEN a AND b

   (6) LIKE, NOT LIKE

3. select 表达式:

```
select 2+3;
select concat(numb,name) from stu;
select concat(rtrim(numb),rtrim(name),100) from stu;#concat连接字符串
select 'abc'||'123';#||连接字符串
```

```
dbzhao=> select numb,name from stu;
    numb    |          name
------------+----------------------
 1001       | zhang
 1002       | 王霞
 1001       | zhang
 1002       | 王霞
 1003       | 李明
 1004       | 张倩
 1005       | 马雯
(7 rows)
```

```
dbzhao=> SELECT 2+3;
 ?column?
----------
        5
(1 row)
```

```
dbzhao=> select concat(numb,name) from stu;
            concat
----------------------------------
 1001       zhang
 1002       王霞
 1001       zhang
 1002       王霞
 1003       李明
 1004       张倩
 1005       马雯
(7 rows)
```

```
dbzhao=> select concat(rtrim(numb),
    concat
---------------
 1001zhang100
 1002王霞100
 1001zhang100
 1002王霞100
 1003李明100
 1004张倩100
 1005马雯100
(7 rows)
```

```
dbzhao=> SELECT 'abc'||'123';
 ?column?
----------
 abc123
(1 row)
```

4.4.2 聚合函数与分组查询

1. 聚合函数的使用:在组内进行

```
SELECT avg(age) as avg_age from student;#查询平均年龄
SELECT max(age) as max_age from student;#查询最大年龄
SELECT min(age) as min_age from student;#查询最小年龄
SELECT sum(age) as sum_age from student;#查询年龄总和
SELECT concat(name, age) as name_age from student;#查询名字和年龄的集合
SELECT class,avg(extract(year from age(birth_date))) as avg_age
from students GROUP BY class;#使用日期函数查询年龄
SELECT count(*) as count_n from student;#查询记录数
SELECT age, count(*) as count_age from student GROUP BY age;#查询各年龄人数
SELECT setval('student_id_seq', 10);#设置序列的值从10开始递增
```

2. 条件查询

```
SELECT count(*) as 男生人数 from student WHERE gender='男';
SELECT avg(age) as 平均年龄
FROM student WHERE sclass='电气71';
```

3. 分组查询

```
SELECT sclass,count(*) from student GROUP BY sclass;#查询各班人数
SELECT sclass,avg(age) from student GROUP BY sclass;#查询各班平均年龄
SELECT sclass,count(*) as pnumber FROM student GROUP BY sclass
HAVING count(*)>10;#查询人数大于10的班级人数
```

4. 除聚合函数外,select 后的列名应该在 group by 中出现。

4.4.3 连接查询

1. 使用 WHERE 子句连接。

```
SELECT a.sno,sname,cname,grade FROM student a, sc b, course c
where a.sno=b.sno and b.cno=c.cno;#查询选课的人的学号、姓名，课程名和分数
SELECT distinct sclass FROM student a, sc b
where a.sno=b.sno;#查询选课的人的班级
SELECT a.sno,sname,cname,grade FROM student a, sc b, course c
where a.sno=b.sno and b.cno=c.cno and sclass='电气41';#查询电气41班的人的课程考试信息
```

2. 内连接

```
select distinct sclass FROM student a
INNER JOIN sc b ON a.sno=b.sno;#查询选课的人的班级
select a.sno, sname, cname, b.grade FROM student a
INNER JOIN sc b ON a.sno=b.sno
INNER JOIN course ON b.cno=course.cno
WHERE sclass ='电气41';#查询电气41班的人的课程考试信息
```

3. 外连接

```
SELECT stu.number, cno, score
FROM stu LEFT OUTER JOIN sc on stu.number=sc.number
WHERE score is NULL;#左外连接查询没有选课的学生信息
SELECT b.cno,b.cname
FROM sc a RIGHT OUTER JOIN course b ON a.cno=b.cno
WHERE a.cno IS NULL;#右外连接查询没有被选过的课程
SELECT *
FROM t1 FULL OUTER JOIN t2 ON t1.number=t2.number;#查询不对应的数据
SELECT *
FROM t1 RIGHT OUTER JOIN t2 on t1.number=t2.number
UNION SELECT *
FROM t1 LEFT OUTER JOIN t2 on t1.number=t2.number;#查询不对应的数据
```

openGauss 不支持全外连接，但支持并 (UNION)、交 (INTERSECT)、差 (EXCEPT) 运算。

4. 自连接

```
SELECT a.cname, b.cname AS 先修
FROM course a INNER JOIN course b ON a.cpno=b.cno;#查询课程的先修课程
```

4.4.4 子查询

1. 带有 IN 或比较算符的子查询

```
SELECT sno, sname, sclass FROM student
WHERE sno in (SELECT sno FROM sc
WHERE cno in (SELECT cno FROM course
WHERE credit=2));#查询选修了2学分课程的学生信息
SELECT sclass, sno, sname FROM student
WHERE sno in (SELECT sno FROM sc WHERE cno in (SELECT cno FROM course
WHERE cname ='大学英语'));#查询选修了课程名为“大学英语”的学生信息
```

2. 带有 ANY 或 ALL 谓词的子查询

```
SELECT sno,sname,sclass,age FROM student
WHERE age<ALL(SELECT age FROM student
WHERE school='电信学院');#查询比电信学院的所有学生年龄都小的人
SELECT sno,sname,sclass,age FROM student
WHERE age<ANY(SELECT age from student
WHERE school='电信学院' ) AND school !='电信学院';#查询年龄总小于电信学院某个学生的其他学院学生
```

4.4.5 查询结果操作

1. 将查询结果插入到已有表中

```
INSERT INTO <table_name> [column1,...] <SELECT sentences>;
```

2. 将查询结果插入到新表中

```
SELECT * INTO <new_table_name>
FROM <table_name>;
```

3. 将查询结果合并

```
<SELECT sentences1> UNION <SELECT sentences2>;
```

4. 交 INTERSECT，差 EXCEPT

4.5 SQL 语言的视图

1. 视图是由存储在数据库中的查询所定义的虚拟表。
2. 视图所对应的查询称为视图定义。
3. 数据库只存放视图的定义，而数据仍放在导出视图的基表中。基表变化，视图的查询结果也会改变。
4. 视图使查询像表一样使用。
5. 视图使用连接、分组和聚合函数操作时，不允许更新。
6. 视图对单表操作，可以更新，实际上是对基表的更新。
7. 创建视图

```
CREATE [OR RAPLACE][TEMP|TEMPORARY][RECUSIVE] VIEW <view_name> [(column_name[1,...n])]
AS <select sentence> [WITH [CASCADED|LOCAL] CHECK OPTION]
```

   (1) 子查询可以任意复杂，**不能包含 DISTINCT**。
   (2) OR REPLACE：如果视图存在，则替换。
   (3) TEMP|TEMPORARY：创建临时视图。会话结束后删除。不保存到数据库中。
   (4) RECURSIVE：创建递归视图，可以引用视图自身。
   (5) WITH CHECK OPTION 表示在对视图进行更新、插入或删除时应满足子查询中的约束条件（openGauss 中不可用）。

```
CREATE VIEW eie AS
SELECT * FROM student WHERE school='电信学院'#建立查询电信学院学生的视图
SELECT sno,sname FROM eie WHERE province='陕西'#查询电信学院的陕西籍学生的学号和姓名
SELECT sno,sname FROM eie, sc
WHERE eie.sno=sc.sno AND grade<60 #查询电信学院的不及格学生的学号和姓名
SELECT sclass,avg(sc.grade) FROM eie,sc
WHERE eie.sno=sc.sno GROUP BY eie.sclass#查询电信学院的各班平均成绩
```

8. 修改视图

```
ALERT VIEW [IF EXISTS] <view_name> RENAME TO <new_view_name>;#重命名
```

9. 删除视图

```
DROP VIEW <view_name>;
```

4.6 SQL 语言的索引

1. 索引是加速搜索引擎检索数据的一种特殊表查询。简单地说，是指向表中数据的指针。
2. 索引有助于加快 SELECT 查询和 WHERE 子句，但它会减慢使用 UPDATE 和 INSERT 语句时的数据输入。
3. 一个索引被创建后，系统必须保持它与表同步。这增加了数据操作的负担。因此以下情况需要避免使用索引：
   (1) 较小的表。
   (2) 有频繁的大批量的更新或插入操作的表。
   (3) 含有大量的 NULL 值的列。
   (4) 频繁操作的列。
4. 索引可以创建或删除，但不会影响数据。
5. 创建索引

```
CRREAT [UNIQUE] INDEX [IF NOT EXISTS] <index_name> ON
<table_name>(<column_name1>[ASC|DESC], [<column_name2>,...]);
```

6. **单列索引**：是只基于表的一个列上创建的索引。

```
CREATE INDEX salary_index ON company(salary);
```

7. **组合索引**：是基于表的多列上创建的索引。

```
CREATE INDEX stu_index ON student(name ASC,tel DESC);
```

8. **唯一索引**：不允许任何重复的值插入到表中。

```
 CREATE UNIQUE INDEX stu_index ON student(id);
```

9. **局部索引**：是在表的子集上构建的索引，子集由一个条件表达式上定义。

```
CREATE INDEX access_log_client_ip_ix ON access_log (client_ip)
WHERE NOT (client_ip > inet '192.168.100.0' AND
client_ip < inet '192.168.100.255');
```

10. **隐式索引**：是在创建对象时，由数据库服务器自动创建的索引。
    openGauss 会自动为定义了一个唯一约束或主键的表创建一个唯一索引。
11. **表达式索引**：从表的一列或多列计算而来的一个函数或者标量表达式作为索引列。

```
CREATE INDEX test1_lower_col1_idx ON test1(lower(col1));#lower使大小写不敏感
CREATE INDEX people_names ON people((first_name||' '||last_name));#连接字符串
```

12. 其他操作
    MySQL、MariaDB 或 Oracle：

```
show tables;#显示所有表
show create table <table_name>;#显示创建表的SQL语句
show create view <view_name>;#显示创建视图的SQL语句
describe <table_name>;#显示表结构
```

PostgreSQL：

```
\q#退出数据库
\c <database_name>#切换数据库
\l#显示所有数据库
\d#显示所有表
\d <table_name>#显示表结构和索引
\di#显示所有索引
\dn#显示schema
```

4.7 openGauss 实现

1. 启动 DBserver 并创建数据表 teaching。

```
log in: root
[root@ecs-c9bf ~]$ su - omm
[omm@ecs-c9bf ~]$ gs_om -t start#启动服务
[omm@ecs-c9bf ~]$ gsql -d postgres -p 26000 -U omm -r#进入数据库postgres
postgres=# CREATE DATABASE teaching IF NOT EXISTS tmp OWNER omm;创建数据库teaching
```

2. 创建下列三个数据表，并批量导入 student，course 和 sc 三个表的数据。

列名	说明	数据类型	约束
sclass	班级	字符串，20	
sno	学号	字符串，20	主键
sname	姓名	字符串，40	非空
gender	性别	字符串，10	男或女
school	学院	字符串，40	
birthday	生日	date	
age	年龄	int	
province	籍贯	字符串，40	

表 1: student

列名	说明	数据类型	约束
cno	课程号	字符串，20	主键
cname	课程名	字符串，40	
credit	学分	float	
cpno	先修课程	字符串，20	

表 2: course

列名	说明	数据类型	约束
sno	学号	字符串，20	联合主键，外键，引用 student 表 sno
cno	课程号	字符串，20	联合主键，外键，引用 course 表 cno
grade	成绩	整数	

表 3: sc

```
postgres=# \q
[omm@ecs -c9bf ~]$ gsql -d teaching -p 26000 -U omm -r#进入teaching数据库
teaching=# CREATE TABLE student (
sclass varchar(20), sno varchar(20) primary key,
sname varchar(40) NOT NULL,
gender varchar(10), CHECK (gender = '男' OR gender = '女'),
school varchar(40),
birthday date,
age int,
province varchar(40));
teaching=# CREATE TABLE course (
cno varchar(20) primary key,
cname varchar(40),
credit float,
cpno varchar(20));

teaching=# INSERT INTO student
values ('材料41', '2014100001', 'wang', '男', '材料', '1999-01-01', 18, '河南');
teaching=# INSERT INTO course values
('CS01', '大学计算机基础',3, NULL);
teaching=#\q

[omm@ecs -c9bf ~]$ mkdir mydata    #创建mydata文件夹
[omm@ecs -c9bf ~]$ cd mydata       #打开mydata文件夹

[omm@ecs -c9bf -mydata]$ gs_dump -f student.txt -t student teaching -p 26000 -U omm
#生成student的脚本文件student.txt
[omm@ecs -c9bf -mydata]$ vi student.txt
[omm@ecs -c9bf -mydata]$ gs_dump -f course.txt -t course teaching -p 26000 -U omm
#生成course的脚本文件course.txt
[omm@ecs -c9bf -mydata]$ vi course.txt
[omm@ecs -c9bf -mydata]$ gsql -d teaching -p 26000 -U omm -r
teaching=# DROP TABLE student  #删除原student表
teaching=# DROP TABLE course   #删除原course表
teaching=# \q
[omm@ecs -c9bf ~]$ cd mydata
[omm@ecs -c9bf -mydata]$ gsql -p 26000 -f student.txt -d teaching
#执行student.txt脚本文件，生成student表
[omm@ecs -c9bf -mydata]$ gsql -p 26000 -f course.txt -d teaching
#执行course.txt脚本文件，生成course表

[omm@ecs -c9bf -mydata]$ gsql -d teaching -p 26000 -U omm -r
teaching=# CREATE TABLE sc (
```

```
sno varchar(20),
cno varchar(20),
grade int,
primary key(sno, cno),
foreign key (sno) references student(sno),
foreign key (cno) references course(cno));

teaching=# INSERT INTO sc values(2130500002, CS04, 81);
teaching=# \q

[omm@ecs -c9bf -mydata]$ gs_dump -f sc.txt -t sc teaching -p 26000 -U omm
#生成sc的脚本文件sc.txt
[omm@ecs -c9bf -mydata]$ vi sc.txt
[omm@ecs -c9bf -mydata]$ gsql -d teaching -p 26000 -U omm -r
teaching=# DROP TABLE sc;    #删除原sc表
teaching=# \q
[omm@ecs -c9bf -mydata]$ gsql -p 26000 -f sc.txt -d teaching
```

3. 查询。

(1) 按学号查询学生信息。

```
select * from student where sno= '2014100001';
```

(2) 按年龄查询学生信息。

```
select * from student where age between 19 and 20;
```

(3) 按年龄和性别查询学生信息。

```
select * from student where age>20 and gender= '女';
```

(4) 按出生日期查询学生生信息。

```
select * from student where birthday>'1998-01-01'
```

(5) 按姓氏查询学生信息。

```
select * from student where sname like '张%';
select * from student where sname like '%张%';
select * from student where sname like '张_';
```

(6) 按籍贯查询某几个省份的学生信息。

```
select * from student where province in ('北京', '上海', '广东');
```

(7) 查询省份为空的学生信息。如果没有，请先插入记录。

```
select * from student where province is null;
insert into student
values ('物试2201', '2223711803', '汪洋', '男', ' 物理学院', '2004-09-05', 18, null);
select * from student where province is null;
```

(8) 按学号统计最高、最低、平均成绩，使用字段别名。

```
select sno, max(grade) as max_grade, min(grade) as min_grade, avg(grade) as avg_grade
from sc group by sno;
```

(9) 按课程号统计最高、最低、平均成绩，使用字段别名。

```
select cno, max(grade) as max_grade, min(grade) as min_grade, avg(grade) as avg_grade
from sc group by cno;
```

(10) 查询各班学生的平均年龄（使用日期函数计算年龄）。

```
select sclass, avg(extract(year from age(birthday))) as avg_age from student group by
    sclass;
```

(11) 查询选课的人的有哪些班级。

```sql
select distinct sclass
from student a inner join sc b on a.sno=b.sno;
```

(12) 查询没有被选过的课程。

```sql
select a.cno, a.cname from course a left outer join sc b on a.cno=b.cno
where b.cno is null;
```

(13) 查询课程的先修课程。

```sql
select b.cno, b.cname from course a inner join course b on a.cpno=b.cno;
```

(14) 查询选修 CS02 号课程的学生名单。

```sql
select a.sname from student a inner join sc b on a.sno=b.sno where cno='CS02';
```

(15) 查询和"刘帅"一个班的学生名单。

```sql
select a.sname from student a inner join student b on a.sclass=b.sclass
where b.sname= '刘帅' and a.sname != '刘帅';
```

(16) 查询选修了 2 学分课程的学生信息。

```sql
select a.sno, a.sname, a.sclass, a.gender, a.age
from student a inner join sc b on a.sno=b.sno inner join course c on b.cno=c.cno
where c.credit=2;
```

(17) 查询选了 2 门以上课程的学生的班级、学号和姓名。

```sql
select sclass, sno, sname from student where sno in
(select sno from sc group by sno having count(*)>2);
```

(18) 查询各班各课程的平均成绩（显示班级、课程名称、平均成绩、考试人数）。

```sql
select a.sclass, b.cname, avg(c.grade) as avg_grade, count(*) as sn
from student a inner join sc c on a.sno=c.sno inner join course b on b.cno=c.cno
group by a.sclass, b.cname;
```

(19) 查询某课程各班的平均成绩、考试人数（显示课程名、班级、考试人数，平均成绩）。

```sql
select b.cname, a.sclass, count(*) as sn, avg(c.grade) as avg_grade
from student a inner join sc c on a.sno=c.sno inner join course b on b.cno=c.cno
where b.cname='数据库基础及应用' group by a.sclass, b.cname;
```

(20) 查询某课程平均成绩在 80 以上的班级、平均成绩和考试人数（显示课程名、班级、考试人数，平均成绩）。

```sql
select b.cname, a.sclass, count(*) as sn, avg(c.grade) as avg_grade
from student a inner join sc c on a.sno=c.sno inner join course b on b.cno=c.cno
where b.cname='数据库基础及应用' group by a.sclass, b.cname having avg(c.grade)>80;
```

5 数据库完整性

5.1 数据库完整性的概念

1. **数据库完整性**：是指数据的正确性（数据是否合法）、有效性（数据是否属于所定义的有效范围）和相容性（描述同一客观事物的数据应该相同）。
2. **数据库安全性**：是指保护数据库，以防止不合法使用所造成的数据泄密、更改或破坏。
3. **完整性约束条件**：附着在数据库中的数据之上的语义约束内容。它们作为模式的一部分存入数据库中。
4. **完整性规则**：数据库中数据应满足的条件。
5. **完整性检查**：检查数据库中数据是否满足规定的条件。
6. 满足基本的数据完整性需求的数据库具有的特点：
   (1) 数据取值准确无误。

(2) 数据与数据之间的关系都是和谐的。

7. 数据库完整性的几种情况：
   (1) 要求的数据：字段不能为空。
   (2) 有效性检查：数据应符合取值范围。
   (3) 实体完整性：主键是唯一的。
   (4) 引用完整性：外键用于连接包含相应字段的另一个表。
   (5) 其他数据库关系：现实中的特殊限制。如规定学生各门课程都及格才能选修某门课程。
   (6) 业务规则：如规定只有某些班级才能选修某门课程。
   (7) 数据一致性
8. 完整性约束条件：
   (1) 列约束：列的数据类型、取值范围、精度和排序等。
   (2) 元组约束：元组中各个字段的约束。
   (3) 关系约束：若干元组之间、关系集合上以及关系之间联系的约束。
9. 完整性控制机制：
   (1) 定义功能：允许用户定义各类完整性约束条件。
   (2) 检查功能：检查用户发出的请求操作是否违背完整性约束条件。
   (3) 违约反应：采取措施来保证数据的完整性。
10. 完整性规则的组成：
   (1) 触发条件：什么时候使用规则进行检查。
   (2) 约束条件：要检查什么样的错误。
   (3) 如果检查出错误，应该怎样处理。
11. 完整性规则的分类：
   (1) 域完整性规则：保障列输入有效。
   (2) 实体完整性规则：所有主属性不能取空值。
   (3) 引用完整性规则：用于约束具有引用关系的两个表中的主键和外键的数据要保持一致。

5.2 DBMS 中的数据完整性

1. 完整性约束条件作为数据库模式定义的有机组成部分，由 DBMS 在用户输入或更新时进行完整性检查。
2. 违反实体完整性和用户定义完整性，拒绝执行；违反参照完整性，可能会附加执行其他操作。
3. 实体完整性控制

```
CREATE TABLE <table_name>(<column_name1> <data_type> PRYMARY KEY,...);#列级定义主键
CREATE TABLE <table_name>(<column_name1> <data_type> ,..., PRYMARY KEY(<column_name1>,...));
#表级定义主键
ALTER TABLE <table_name>
ADD CONSTRAINT <constraint_name> primary key (<column_name>);#添加主键约束
ALTER TABLE <table_name>
ADD CONSTRAINT <constraint_name> UNIQUE (<column_name>);#添加唯一值约束
```

4. 引用完整性控制

```
CREATE TABLE <table_name>
(<column_name> <data_type>[FOREIGN KEY] REFERENCES <main_table_name>
[ON DELETE {RESTRICT|CASCADE|SET NULL|NO ACTION|SET DEFAULT}]
[ON UPDATE {RESTRICT|CASCADE|SET NULL|NO ACTION|SET DEFAULT}]);
#指定当删除或更新主表中的关联记录时如何处理外键表中的记录
ALTER TABLE <table_name>
ADD CONSTRAINT <constraint_name>
[FOREIGN KEY](<column_name>,...) REFERENCES <main_table_name>(<column_name>,...)
[ON DELETE {RESTRICT|CASCADE|SET NULL|NO ACTION|SET DEFAULT}]
[ON UPDATE {RESTRICT|CASCADE|SET NULL|NO ACTION|SET DEFAULT}]);
```

5. 用户定义完整性控制

```
CREATE TABLE <table_name>
(<column_name1> <data_type1> NOT NULL,
<column_name2> <data_type2> UNIQUE,
<column_name3> <data_type3> CHECK(<condition>);#属性上的约束条件定义
CREATE TABLE <table_name>
(<column_name> <data_type>,...,CHECK(gender='女' OR name NOT LIKE '女-%');
#元组上的约束条件定义
```

eg:

```
CREATE TABLE score(sno char(10) primary key,
c1 int CHECK(c1>0), c2 int CHECK(c2>0), c3 int CHECK(c3>0),
CONSTRAINT 3 pk_c3 CHECK(c3=c1+c2));
```

6. 修改完整性约束

```
ALTER TABLE <table_name> DROP CONSTRAINT <constraint_name>;
ALTER TABLE <table_name> ADD CONSTRAINT <constraint_name> CHECK(<condition>);
```

5.3 触发器

1. **触发器**：是强制执行特定操作的语句或程序。
2. 每当发生了施加于触发器所保护的数据之上的修改操作时，相应的操作就会自动执行。
3. 触发器的组成：事件、条件和动作。
   (1) 事件：施加于数据之上的操作，如插入、删除或修改。
   (2) 动作：一系列的语句。
   (3) 条件：动作执行的条件。
4. 常规触发器：依附于表、视图。
5. 事件触发器：定义在某个特定数据库上，在该数据库范围内是全局可见的，可以捕获 DDL 事件。
6. 约束和触发器是两种强制业务逻辑和数据完整性的机制。
7. **触发器函数**：是触发器触发时执行的一段程序。
   (1) 在定义触发器之前定义。
   (2) 不带参数，返回类型为 TRIGGER。
   (3) 一个触发器函数可以用在多个触发器中。
   (4) tg_op 是一个特殊的变量，用于触发器函数内部。表示触发器的当前操作类型。返回以下值之一：INSERT、UPDATE、DELETE、TRUNCATE（注意一定要大写）。
8. 创建触发器函数

```
CREATE OR REPLACE FUNCTION <function_name>()
RETURNS TRIGGRE
LANGUAGE PLPGSQL
AS $$
BEGIN
<SQL sentence>;
RETURN NEW[OLD];
END;
$$;
```

9. 删除触发器函数

```
DROP FUNCTION <function_name>();
```

10. 创建触发器

```
CREATE TIGGER <trigger_name>
{BEFORE|AFTER|INSTEAD OF} {INSERT|UPDATE[of <column_name>[,...]]|DELETE|TRUNCATE[OR...]}
ON <table_name>[<view_name>]
[FOR[EACH]{ROW|STATEMENT}] [WHEN (<condition>)]
EXECUTE PROCEDURE <function_name>(arguments);
```

11. 修改触发器

```
ALTER TRIGGER <trigger_name> ON <table_name>[<view_name>]...
ALTER TRIGGER <trigger_name> ON <table_name>[<view_name>] RENAME TO <new_trigger_name>;#重命名
    触发器
```

12. 删除触发器

```
DROP TRIGGER [IF EXISTS] <trigger_name> ON <table_name>;
```

13. 禁用/启用触发器

```
ALTER TABLE <table_name> {DISABLE|ENABLE} TRIGGER <trigger_name>;
```

14. 其他操作

```
SELECT tgrelid,tgname from pg_trigger;#显示所有触发器
SELECT pc.relname AS table_name, pt.tgname AS trigger_name
FROM pg_trigger pt JOIN pg_class pc ON pt.tgrelid = pc.oid;#显示所有关联表和触发器
\df #显示所有函数
```

因为 PostgreSQL 中的函数有四种分类：aggregates/normal/trigger/ window functions，所以 psql 中提供了附加参数，分别用来显示对应的函数：\dfa、\dfn、dft、dfw。

15. 触发器说明：
    (1) old.<column_name> 表示更新前的表，new.<column_name> 表示更新后的表。
    (2) 触发器和表关联。
    (3) CHECK 约束在触发器之前检查。
    (4) 可以在一个表上创建多个触发器，创建多个同一事件的触发器。
    (5) 如果为同一事件定义了多个相同类型的触发器，则按触发器名称的首字母顺序触发它们。
    (6) 触发体不能返回结果集合，如 select 等。
    (7) INSTEAD OF 触发器不支持 WHEN。
    (8) 如果定义了级联触发器，要防止循环触发。
    (9) 要在一个表上创建一个触发器，用户必须具有该表上的 TRIGGER 特权。用户还必须具有在触发器函数上的 EXECUTE 特权。

5.4 两个实例

1. 创建触发器，记录操作日志。

创建职员表 emp 和审计日志表 emp_audit，在职员表上创建触发器以在审计日志表中记录用户所做的操作。在 emp 上做相应操作，验证触发器的功能，最后删除相应的对象。在实验报告中展示操作过程和结果。审计日志表应包括操作类型（INSERT、DELETE、UPDATE）、时间、操作员和职员姓名。

```
drop table if exists emp;
create table emp(id serial primary key, name varchar(20));
create table emp_audit(action char(1), action_time timestamp, action_user varchar(20), emp_name
     varchar(20));

create or replace function emp_audit_funcion()
returns trigger
language plpgsql
as $$
begin
if (tg_op='INSERT') then
insert into emp_audit values('I', now(), user, new.name);
return new;
elsif (tg_op='UPDATE') then
insert into emp_audit values('u', now(), user, new.name);
return new;
elsif (tg_op='DELETE') then
insert into emp_audit values('d', now(), user, old.name);
```

```
return old;
end if;
return null;
end;
$$;

create trigger emp_audit_trigger after insert or update or delete on emp for each row
execute procedure emp_audit_funcion();

insert into emp values('123456', 'Andy');
update emp set name = 'Ash' where name = 'Andy';
delete from emp where name='Ash';
select * from emp_audit;

drop trigger emp_audit_trigger on emp;
drop function emp_audit_function();
drop table emp;
drop table emp_audit;
```

2. 创建触发器，保证数据一致性。

创建学生表和班级表，班级表中记录班级名称和人数，在学生表上创建触发器，实现学生表的添加、修改和删除时相应的班级表中的班级人数随之改变。在学生表上进行相应操作，验证触发器的功能，最后删除相应的对象。

```
create table stu(sID varchar(10) primary key, name varchar(20), class varchar(20), age int);
create table class(name varchar(20), count int);

create or replace function change_class_count()
returns trigger
language plpgsql
as $$
begin
if (tg_op='INSERT') then
  if (exists(select * from class where name=new.class)) then
update class set count=count+1 where name=new.class;
return new;
else
insert into class values(new.class, 1);
return new;
end if;
elsif (tg_op='UPDATE') then
  if (new.class!=old.class) then
update class set count=count-1 where name=old.class;
if (exists(select * from class where name=new.class)) then
update class set count=count+1 where name=new.class;
return new;
else
insert into class values(new.class,1);
return new;
end if;
end if;
elsif (tg_op='DELETE') then
update class set count=count-1 where name=old.class;
return old;
end if;
return null;
end;
$$;

create trigger change_class_count_trigger after insert or update or delete on stu
for each row
execute procedure change_class_count();

```

```sql
insert into stu values('001', '张三', '计算机', 18);
insert into stu values('002', '李四', '计算机', 18);
insert into stu values('003', '王五', '数学', 18);
insert into stu values('004', '赵六', '物理', 18);
select * from class;

update stu set class='物理' where class='数学';
select * from class;

update stu set class='机械' where class='物理';
select * from class;

delete from stu where class='机械' ;
select * from class;

drop trigger change_class_count_trigger on stu;
drop function change_class_count();
drop table stu;
drop table class;
```

6 数据库应用程序

6.1 存储过程和函数

1. **存储过程**：是一组 SQL 语句和逻辑控制的集合。
2. 存储过程的优点：
   (1) 提高执行效率。
   (2) 减低数据传输量。
   (3) 增强安全性。
3. **PL/pgSQL 过程语言**：是一种用于 PostgreSQL 数据库系统可载入的过程语言。
4. 函数的参数表示：<function_name>.<column_name>。
   当函数的参数名与表的列名重复时，列名优先。
   在更旧的数字方法中可以用 $n 表示第 n 个参数。
5. 创建存储过程

```sql
CREATE PROCEDURE <procedure_name>[(parameter,...)]
AS
BEGIN
<SQL sentence>;
END
/
```

6. 创建函数

```sql
CREATE FUNCTION <function_name>([parameter,...])
RETURNS <>
LANGUAGE plpgsql
AS $$
[DECLARE <column_name> <datd_type> [DEFAULT]]
BEGIN
<SQL sentence>;
RETURN <>;
END;
$$;
```

7. 条件和循环

```sql
IF...THEN...END IF;
IF...THEN...ELSE...END IF;
```

```
IF...THEN...ELSIF...THEN...ELSE...END IF;#IF条件

CASE <expression>
WHEN {<value1>|<condition1>}[,...] THEN result1
[WHEN {<value2>|<condition2>}[,...] THEN result2
...]
[ELSE resultn]
END CASE;#CASE条件

FOR i IN [REVERSE] <start>..<end> [BY 1] LOOP...END LOOP;#FOR循环

WHILE <condition> LOOP...END LOOP;#WHILE循环
```

8. 调用存储过程/函数

```
SELECT <procedure_name>[<function_name>](parameter,...);
SELECT * FROM <procedure_name>[<function_name>](parameter,...);
CALL <procedure_name>[<function_name>](parameter,...);
```

9. 删除存储过程/函数

```
DROP PROCEDURE[FUNCTION] <procedure_name>[<function_name>];
```

10. 插入数据

```
CREATE OR REPLACE FUNCTION <function_name>(<column_name> <data_type>,...)
RETURNS VOID
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO <table_name> VALUES(parameter,...);
END;
$$;
```

11. 返回一行

```
CREATE OR REPLACE FUNCTION <fuction_name>(<column_name> <data_type>,...)
RETURNS SETOF <table_name>
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY SELECT * FROM <table_name> <condition>;
END;
$$;
```

12. 返回多行

```
CREATE OR REPLACE FUNCTION <function_name>(<column_name> <data_type>,...)
RETURNS SETOF <table_name>
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY SELECT * FROM <table_name> <condition>;
END;
$$;
```

13. 返回一列

```
CREATE OR REPLACE FUNCTION <function_name>(<column_name> <data_type>,...)
RETURNS SETOF <data_type>
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY SELECT <column_name> FROM <table_name> <condition>;
END;
$$;
```

14. 返回多列

```
CREATE OR REPLACE FUNCTION <function_name>(<column_name> <data_type>,...)
RETURNS TABLE(<column_name1> <data_type1>, <column_name2> <data_type2>,...)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY SELECT <column_name1>, <column_name2>,... FROM <table_name> <condition>;
END;
$$;
```

15. 其他操作

```
\sf <function_name>#查看函数定义
SELECT proname, proowner, pronamespace FROM pg_proc WHERE proname LIKE 'my%';#查看函数列表
```

6.2 函数实例

1. 计算两数之和

```
CREATE OR REPLACE FUNCTION add_numbers(a INTEGER, b INTEGER)
RETURNS INTEGER
LANGUAGE plpgsql
AS $$
DECLARE
t INTEGER;
BEGIN
    t := a+b;
    RETURN t;
END;
$$;
```

2. 计算 n!

```
CREATE OR REPLACE FUNCTION factorial(n INTEGER)
RETURNS INTEGER
LANGUAGE plpgsql
AS $$
DECLARE
    result INTEGER := 1;
    i INTEGER;
BEGIN
    IF n < 0 THEN
        RAISE EXCEPTION 'Input must be a non-negative integer.';
    END IF;
    FOR i IN 1..n
    LOOP
        result := result * i;
    END LOOP;
    RETURN result;
END;
$$;
#使用FOR循环
CREATE OR REPLACE FUNCTION factorial(n INTEGER)
RETURNS INTEGER
LANGUAGE plpgsql
AS $$
DECLARE
    result INTEGER := 1;
    i INTEGER := 1;
BEGIN
    IF n < 0 THEN
        RAISE EXCEPTION 'Input must be a non-negative integer.';
    END IF;
```

```
    WHILE i <= n
    LOOP
        result := result * i;
        i := i + 1;
    END LOOP;
    RETURN result;
END;
$$;
#使用WHILE循环
```

3. 计算斐波那契数列第 n 项的函数

```
CREATE OR REPLACE FUNCTION fibonacci(n INTEGER)
RETURNS INTEGER
LANGUAGE plpgsql
AS $$
DECLARE
    a INTEGER := 0;
    b INTEGER := 1;
    i INTEGER;
    temp INTEGER;
BEGIN
    IF n <= 1 THEN
        RETURN n;
    END IF;
    FOR i IN 2..n LOOP
        temp := a + b;
        a := b;
        b := temp;
    END LOOP;
    RETURN b;
END;
$$;
#循环解法
CREATE OR REPLACE FUNCTION fibonacci(n INTEGER)
RETURNS INTEGER
LANGUAGE plpgsql
AS $$
BEGIN
    IF n <= 1 THEN
        RETURN n;
    ELSE
        RETURN fibonacci(n - 1) + fibonacci(n - 2);
    END IF;
END;
$$;
#递归解法
```

4. 计算 10 以内的奇数和

```
CREATE OR REPLACE FUNCTION sum_odds(n TNTEGER)
RETURNS INTEGER
LANGUAGE plpgsql
AS $$
DECLARE
    sum_odds INTEGER :=0 ;
    i INTEGER;
BEGIN
    FOR i IN 1..10
    LOOP
        IF i % 2 != 0 THEN
            sum_odds := sum_odds + i;
        END IF;
    END LOOP;
    RETURN sum_odds;
```

```
END;
$$;
#使用FOR循环
CREATE OR REPLACE FUNCTION sum_odds(n TNTEGER)
RETURNS INTEGER
LANGUAGE plpgsql
AS $$
DECLARE
    sum_odds INTEGER :=0 ;
    i INTEGER := 1;
BEGIN
    WHILE 1 LOOP  #此处1被视为真，创建了一个无限循环
        IF i >= 10 THEN
            EXIT;
        END IF;
        IF i % 2 != 0 THEN
            sum_odds := sum_odds + i;
            i = i + 1;
            CONTINUE;
        END IF;
        i = i + 1;
    END LOOP;
    RETURN sum_odds;
END;
$$;
#使用WHILE循环
```

6.3 Python 可视化数据库应用系统

6.3.1 设置防火墙安全规则

6.3.2 设置 DB 加密方式和监听 IP

1. 修改密码加密方式

```
[omm@ecs -c9bf ~]$ gs_guc set -I all -c "password_encryption_type=1"
```

2. 修改 DB 监听地址

```
[omm@ecs -c9bf ~]$ vi /gaussdb/data/db1/postgresql.conf
listen_addresses='10.0.0.15'
local_bind_address ='10.0.0.15'
修改为
listen_addresses='*'
local_bind_address='*'
```

3. 修改登录主机信任方式（允许远程访问，在 IPV4 下面）

```
vi /gaussdb/data/db1/pg_hba.conf
增加
host all all 192.168.56.1/24 md5 #虚拟机使用
host all all 0.0.0.0/0 md5 #ECS使用
```

6.3.3 创建数据库用户并设密码

```
[omm@ecs-5138 ~]$ gs_om -t start
[omm@ecs-5138 ~]$ gsql -p 26000 -d postgres -U omm -r
postgres=# create user xjtudba with password 'xjtu@123';
postgres=# create database dbapp owner xjtudba;
postgres=# \c dbapp
dbapp=# create schema xjtudba authorization xjtudba;
dbapp=# set search_path to xjtudba;
```

6.3.4 Python 编程

```python
import tkinter as tk
from tkinter import messagebox
import psycopg2

def create_database():
    yes=messagebox.askokcancel('确认初始化','初始化会删除所有数据！')
    if yes == True:
        con = psycopg2.connect(
            host='123.249.97.41',
            port=26000,
            user='xjtudba',
            password='xjtu@123',
            database='dbapp')
        cur = con.cursor();#创建游标对象
        sql="drop table if exists addrbook"
        cur.execute(sql)
        sql="create table if not exists addrbook(name char(20) primary key,"
        sql+="tel char(15))"
        cur.execute(sql)
        con.commit()#提交事务
        cur.close()#关闭游标
        con.close()#断开连接
        print("初始化完成")
    else:
        print("初始化失败")

root = tk.Tk()
root.title("OpenGauss Connection")#创建Tkinter窗口

button = tk.Button(root, text="Create Database", command=create_opengauss)
button.pack()#创建按钮

root.mainloop()#启动主事件循环
```

7 数据库安全性

7.1 数据库安全性

1. 数据库安全保护的目标：确保只有授权用户才能访问数据库，未授权用户不能查看或使用数据库中的数据。
2. 数据库安全涉及数据库技术之外的防盗、防灾、防破坏、人员分级与审查等。就数据库技术来说，主要是针对存储的数据，包括独立性、安全性、完整性、并发控制、故障恢复等。
3. 数据库安全性的控制方式：
   (1) 物理处理方式：对口令泄露、窃听通信线路、盗窃物理设备等加密、加强保护、强化警戒等。
   (2) 系统处理方式：数据库系统的处理方式。
4. 安全性级别：
   (1) 环境级：对机房、设备的保护，防止物理破坏。
   (2) 用户级：对职员的管理，对人员的限制。
   (3) 操作系统级：限制进入操作系统，防止通过操作系统进入数据库。
   (4) 网络级：限制人员、数据库、操作通过网络访问。
   (5) 数据库级：数据库用户的身份、权限。
5. 措施原则：最小权限，最短时间。
6. 数据库安全控制的方法：

(1) 用户标识与鉴别。
    (a) 身份认证。
    (b) 口令认证。
    (c) 随机数运算认证。
(2) 存取控制。
    (a) 构成：定义用户权限、进行权限检查。
    (b) 类别：自主存取控制、强制存取控制。
(3) 视图机制。
(4) 数据加密。
(5) 审计方法。

7.2 OpenGauss 的用户和权限管理

7.2.1 用户

1. **管理员用户**：授予管理员权限的用户。
   (1) 系统管理员、安全管理员和审计管理员三权分立。但初始用户 (omm) 权限不受三权分立影响。
   (2) 常用管理员权限：SYSADMIN、CREATEDB、CREATEROLE、AUDITADMIN、LOGIN。
   (3) SYSADMIN 权限可以通过 GRANT/REVOKE ALL PRIVILEGES 授予或撤销，但无法通过 ROLE 和 USER 的权限被继承，也无法授予 PUBLIC。
2. **普通用户**：由初始用户、系统管理员、有 CREATEROLE 属性的安全管理员创建。
3. 创建用户

```
CREATE USER <user_name> [[WITH]
[CREATEDB|NOCREATEDB]#允许或禁止用户创建数据库，默认为NOCREATEDB
[CREATEROLE|NOCREATEROLE]#允许或禁止用户创建角色，默认为NOCREATEROLE
[INHERIT|NOINHERIT]#允许或禁止用户继承角色，默认为INHERIT
[LOGIN|NOLOGIN]#允许或禁止用户登录，默认为LOGIN
[CONNECTION LIMIT connlimit]#限制用户最大并发连接数，默认为-1（无限制）
[[ENCRYPTED] PASSWORD '<password>']#设置密码
[[IN] ROLE <role_name>[,...]]];#将用户添加到一个或多个现有角色中
```

4. 修改用户

```
ALTER USER <old_user_name> RENAME TO <new_user_name>;#重命名用户
ALTER USER <user_name> <privileges>;#修改权限
ALTER USER PASSWORD '<password>';#添加密码
ALTER USER <user_name> IDENTIFIED BY '<old_password>' REPLACE
'<new_password>';#修改密码
```

5. 删除用户

```
DROP USER <user_name>;
```

6. 其他操作

```
SELECT USER;#查看当前用户
\d pg_user#查看所有用户
SELECT usename,usesysid,usesuper FROM pg_user;#查看所有用户
\d pg_authid#查看所有角色
SELECT rolname,oid,rolsuper FROM pg_authid;#查看所有角色
```

7.2.2 角色

1. **角色**：是权限的集合，可以被看成是一个数据库用户或者是一个数据库用户组。
2. 一个用户属于某个角色，这个用户就具有该角色的权限。

3. OpenGauss 不分用户和角色。
   "CREATE USER" 是"CREATE ROLE" 的别名,这两个命令几乎相同,唯一的区别是"CREATE USER" 命令创建的用户默认带有 LOGIN 属性,而"CREATE ROLE" 命令创建的用户默认不带 LOGIN 属性。
4. 创建角色

```
CREATE ROLE <role_name> WITH LOGIN PASSWORD '<password>'
[VALID BEGIN '<begin_date>' VALID UNTIL '<end_date>'];#指定生效期和失效期
```

5. 修改角色

```
ALTER ROLE <old_role_name> RENAME TO <new_role_name>;#重命名角色
ALTER ROLE <role_name> <privileges>;#修改角色权限
```

6. 删除角色

```
DROP ROLE <role_name>;
```

7.2.3 PUBLIC

1. **PUBLIC**：是一个角色组，包含所有角色。
2. PUBLIC 默认拥有的权限包括：数据库 CONNECT 权限、SQL 查询权限、函数 EXECUTE 权限、语言和数据类型（包括域）USAGE 权限。
3. 默认情况下,对表、表字段、序列、外部数据源、外部服务器、模式或表空间对象的权限不会授予 PUBLIC。
4. 授予 PUBLIC 角色组权限，相当于授予所有的用户。

7.2.4 Schema

1. **模式 (Schema)**：类似于操作系统的目录，是一个命名空间，包含表、视图、索引、数据类型、函数、操作符等。
2. 模式不能嵌套。
3. 创建用户时会自动创建一个同名模式。创建角色时不会。
4. 一个用户可以有多个模式，一个模式只有一个用户。
5. 创建模式

```
CREATE schema <schema_name>;
```

6. 修改模式

```
ALTER SCHEMA <old_shema_name> RENAME TO <new_schema_name>;#重命名模式
ALTER SCHEMA <schema_name> OWNER TO <user_name>;#修改模式用户
```

7. 删除模式

```
DROP SCHEMA <schema_name>;
```

8. 其他操作

```
\dn 显示所有模式
\d pg_namespace#显示所有模式
SELECT nspname,nspowner FROM pg_namespace;#显示所有模式
\dt 显示s所有表及其模式
SET search_path to <schema_name>;#设置默认模式
CREATE TABLE <schema_name>.<table_name>(<column_name> <data_type>,...);#在模式下创建表
DROP TABLE <schema_name>.<table_name>;#删除模式下的表
```

7.2.5 用户权限的设置与回收

1. 授权

```
GRANT {{SELECT|INSERT|UPDATE|DELETE|TRUNCATE|REFERENCES|TRIGGER}[,...]|ALL[PRIVILEGES]}
ON {TABLE <table_name>[,...]|ALL TABLES IN SCHEMA <schema_name>[,...]|
DATABASE <database_name>[,...]|SCHEMA <schema_name>[,...]}
TO <user_name>[,...] [WITH GRANT[ADMIN] OPTION];
GRANT <user_name1> TO <user_name2> [WITH GRANT[ADMIN] OPTION];#将用户的权限授权给其他用户
```

2. 回收

```
REVOKE {{SELECT|INSERT|UPDATE|DELETE|TRUNCATE|REFERENCES|TRIGGER}[,...]|ALL[PRIVILEGES]}
ON {TABLE <table_name>[,...]|ALL TABLES IN SCHEMA <schema_name>[,...]
|DATABASE <database_name>[,...]|SCHEMA <schema_name>[,...]}
FROM <user_name>[,...] [WITH REVOKE[ADMIN] OPTION];
REVOKE <user_name1> FROM <user_name2>;#将用户或角色的权限回收
```

3. 辅助操作

```
\c <database_name>#切换数据库
\c <user_name>#切换用户
\c <database_name> <user_name>#切换数据库和用户
SELECT USER;#查看当前用户
SELECT * FROM CURRENT_DATABASE();#查看当前数据库
```

7.3 事务

1. **事务**：是整体完成或撤销的一系列数据库操作（如查询、插入、修改或删除）的序列。
2. 事务的类型：显式事务、隐式事务、自定义事务。
3. 事务的特点：ACID 属性
   (1) 原子性（atomicity）：一个事务是一个不可分割的工作单位。
   (2) 一致性（consistency）：事务使数据库从一个一致性状态变到另一个一致性状态。
   (3) 隔离性（isolation）：一个事务的执行不能被其他事务干扰。
   (4) 持久性（durability）：一个事务一旦提交，它对数据库中数据的改变就是永久性的。
4. 事务的 4 种隔离级别：
   (1) 读未提交：READ UNCOMMITTED
   (2) 读已提交：READ COMMITTED
   (3) 可重复读：REPEATABLE READ
   (4) 可串行化：SERIALIZABLE
5. 创建事务

```
BEGIN[BEGIN TRANSACTION|START TRANSACTION];
[SET TRANSACTION [ISOLATION LEVEL <isolation_level>] [READ WRITE|READ ONLY];]
#设置隔离级别和访问模式
<SQL sentences>
COMMIT[ROLLBACK];
```

6. 实际上 OpenGauss 将每一个 SQL 语句都作为一个事务来执行。如果没有发出 BEGIN 命令，则每个独立的语句都会被加上一个隐式的 BEGIN 以及 COMMIT（如果成功）来包围它。
7. 保存点可以多次回滚。

7.4 备份和恢复

7.4.1 SQL 转储（备份数据库）

创建一个由 SQL 命令组成的文件，当把这个文件回馈给服务器时，服务器将利用其中的 SQL 命令重建与转储时状态一样的数据库。

利用 openGauss 提供的程序 gs_dump。

1. 数据库导出到文件

```
[omm@ecs-c9bf ~]$ gs_dump <database_name> -p <port> -U <username> -W <password> -f <file_name>
#数据库导出到文件
[omm@ecs-c9bf ~]$ gs_dump <database_name> > <file_name> -p <port> -U <username> -W <password>
#也可以用重定向符>
```

2. 文件恢复为数据库

```
[omm@ecs-c9bf ~]$ createdb -p 26000 -U <username> -W <password> <database_name>
#创建新数据表
[omm@ecs-c9bf ~]$ gsql -d <database_name> -p 26000 -U <username> -W <password> -f <file_name>
#文件恢复为数据库
[omm@ecs-c9bf ~]$ gsql <database_name> < <file_name> -p <port> -U <username> -W <password>
#也可以用重定向符<
```

3. 备份所有数据库

```
[omm@ecs-c9bf ~]$ gs_dumpall -f gsdumpall.sql -p <port>
#所有数据库导出到文件
[omm@ecs-c9bf ~]$ gsql -f gsdumpall.sql -d <database_name> -p <port>
#文件恢复为数据库
```

4. gs_dump 可选参数：
   (1) -n schema 或-t table 选项备份该数据库中能够访问的模式和表。
   (2) -h host 和 -p port，声明连接哪个数据库服务器
   (3) 默认主机是本地主机或 PGHOST 环境变量指定的主机。
   (4) 默认端口是环境变量 PGPORT 或（如果 PGPORT 不存在）内建默认值。
   (5) gs_dump 默认与当前操作系统用户名同名的数据库用户名连接。使用其他名字要声明-U 选项。
   (6) gs_dumpall 备份给定集簇中的每一个数据库，并且保留集簇范围的数据，如角色和表空间定义。

7.4.2 文件系统级备份

通过对物理文件拷贝的方式对数据进行备份，以磁盘块为基本单位将数据从主机上复制到备机上，通过备份的数据文件及归档日志等文件，可以做到完全恢复。物理备份速度快，通常用于数据的备份与恢复，适用于全量备份。

1. 数据库文件备份

   方法 1 归档数据库文件

```
[omm@ecs-c9bf ~]$ tar -cf backup.tar/gaussdb/data/<database_name>
```

   方法 2 新建目录创建备份

```
[omm@ecs-c9bf ~]$ cd ~#将当前工作目录更改为主目录
[omm@ecs-c9bf ~]$ mkdir mydata#在主目录中创建一个新目录
[omm@ecs-c9bf ~]$ cd mydata#将当前工作目录更改为新目录
[omm@ecs-c9bf ~]$ mkdir back#创建一个新子目录
[omm@ecs-c9bf ~]$ gs_basebackup -D back -p 26000#执行数据库备份并将其存储在子目录中
```

2. gs_basebackup 可选参数：
   (1) -D：备份文件输出的目录。
   (2) -X：备份 xlog 日志的方式（fetch|stream）。
   (3) -F：指定备份文件的输出格式（plain|tar）。
   (4) -Z：指定备份归档级别（只有输出格式为 tar 时才生效）。
   (5) -U：指定使用哪个用户进行备份操作。
   (6) -p：指定数据库服务监听的端口号。
   (7) -W：指定用户连接密码。

7.4.3 恢复

1. 停止数据库服务

```
[omm@ecs-c9bf ~]$ gs_om -t stop
```

2. 恢复数据库
   方法 1 拷贝替换原文件后启动数据库

```
[omm@ecs-c9bf ~]$ cp -f backup /gaussdb/data/<database_name>
```

   方法 2 直接在备份库上启动数据库

```
[omm@ecs-c9bf ~]$ gs_om -t start -D /mydata/backup
```

7.5 安全策略

1. **安全策略**：指在某个安全区域内，用于所有与安全相关的一套安全规则和行动策略。
2. openGauss 中包括：账户安全策略、账号有效期、密码安全策略等。

7.5.1 账户安全策略

1. 用户输入密码次数超过一定次数，系统将自动锁定该帐户，默认值为 10。
2. 帐户被锁定时间超过设定值，则自动解锁，默认值为 1 天。
3. 查看尝试次数（默认值为 10）

```
postgres=# SHOW password_encryption_type;
```

4. 修改尝试次数

```
 [omm@ecs-c9bf ~]$ gs_guc reload -D /gaussdb/data/db1 -c "failed_login_attempts=<count_time>"
```

5. 查看锁定时间（默认值为 1）

```
postgres=# SHOW password_lock_time;
```

6. 修改锁定时间

```
[omm@ecs-c9bf ~]$ gs_guc reload -N all -I all -c "password_lock_time=<count_day>"
```

7.5.2 账号有效期

1. 创建新用户时，需要限制用户的操作期限（有效开始时间和有效结束时间）。
2. 不在有效操作期内的用户需要重新设定帐号的有效操作期。
3. 创建用户并制定用户的有效开始时间和有效结束时间。

```
postgres=# CREATE USER <user_name> WITH PASSWORD '<password>' VALID BEGIN '<begin_date>' VALID
    UNTIL '<end_date>';
```

4. 重新设定帐号的有效期

```
postgres=# ALTER USER <user_name> WITH VALID BEGIN '<begin_date>' VALID UNTIL '<end_date>';
```

7.5.3 密码安全策略

1. 用户密码存储在系统表 pg_authid 中，openGauss 对用户密码可进行加密存储、密码复杂度设置、密码重用天数设置、密码有效期限设置等。
2. 查看已配置的加密算法（默认值为 2）

```
postgres=# SHOW password_encryption_type;
```

3. 修改加密算法

```
[omm@ecs-c9bf ~]$ gs_guc reload -N all -I all -c "password_encryption_type=2"
```

(1) 当参数 password_encryption_type 设置为 0 时，表示采用 md5 方式对密码加密。md5 为不安全的加密算法，不建议使用。

(2) 当参数 password_encryption_type 设置为 1 时，表示采用 sha256 和 md5 方式对密码加密。其中包含 md5 为不安全的加密算法，不建议使用。

(3) 当参数 password_encryption_type 设置为 2 时，表示采用 sha256 方式对密码加密。

4. 查看已配置的密码安全参数（默认值为 1）

```
postgres=# SHOW password_policy;
```

5. 修改密码安全参数

```
[omm@ecs-c9bf ~]$ gs_guc reload -N all -I all -c "password_policy=1"
```

(1) 当参数 password_policy 设置为 1 时表示采用密码复杂度校验。

(2) 当参数 password_policy 设置为 0 时表示不采用任何密码复杂度校验，会存在安全风险，不建议设置为 0。

6. 查看密码不可重用天数（默认值为 60）

```
postgres=# SHOW password_reuse_time;
```

7. 修改密码不可重用天数

```
[omm@ecs-c9bf ~]$ gs_guc reload -N all -I all -c "password_reuse_time=<count_day>"
```

8. 查看密码不可重用次数（默认值为 0）

```
postgres=# SHOW password_reuse_max;
```

9. 修改密码不可重用次数

```
[omm@ecs-c9bf ~]$ gs_guc reload -N all -I all -c "password_reuse_max=<count_time>"
```

10. 查看密码有效期（默认值为 90）

```
postgres=# SHOW password_effect_time;
```

11. 修改密码有效期

```
[omm@ecs-c9bf ~]$ gs_guc reload -N all -I all -c "password_effect_time=<count_day>"
```

12. 查看到期提醒天数

```
postgres=# SHOW password_notify_time;
```

13. 修改到期提醒天数

```
[omm@ecs-c9bf ~]$ gs_guc reload -N all -I all -c "password_notify_time = 7"
```

7.6 审计

1. 查看审计总开关参数（默认值 on）

```
postgres=# show audit_enabled;
```

2. 修改审计总开关参数

```
gs_guc reload -N all -I all -c "audit_enabled=on"
```

3. 其他操作：

(1) 用户登录、注销审计：audit_login_logout。默认值为 7。不推荐设置除 0 和 7 之外的值。

(2) 数据库启动、停止、恢复和切换审计：audit_database_process。默认值为 1。

(3) 用户锁定和解锁审计：audit_user_locked。默认值为 1。

(4) 用户访问越权审计：audit_user_violation。默认值为 0。

(5) 权和回收权限审计：audit_grant_revoke。默认值为 1。

(6) 数据库对象的 CREATE，ALTER，DROP 操作审计：audit_system_object。默认值为 12295。
(7) 表的 INSERT、UPDATE 和 DELETE 操作审计：audit_dml_state。默认值为 0。
(8) SELECT 操作审计：audit_dml_state_select。默认值为 0。
(9) COPY 审计：audit_copy_exec。默认值为 0。
(10) 存储过程和自定义函数的执行审计：audit_function_exec。默认值为 0。
(11) SET 审计：audit_set_parameter。默认值为 1。

4. 查看审计记录

```
postgres=# SELECT time,type,result,username,object_name  FROM pg_query_audit('<begin_date>','<end_date>');
```

8 数据库设计与创建

8.1 数据库设计方法

1. **数据库设计**：根据用户的数据管理需求、数据操纵需求和数据库支撑环境设计出满足用户需求的数据库模式以及典型的应用程序。
2. 数据库设计的目的：对一个给定的环境，构造最优的数据库模式，建立数据库及应用系统，使之能够有效地存储和操纵数据，满足不同用户的应用需求。
3. 数据库设计方法：
   (a) 面向过程的方法：以处理需求为主，兼顾信息需求。
   (b) 面向数据的方法：以信息需求为主，兼顾处理需求。（主流）
4. 传统数据库设计方法：目前一般采用生命周期法。
   (1) 需求分析阶段；形成用户的需求规格说明（使用数据流图）。
   (2) 概念结构设计阶段：形成 E-R 模型或面向对象模型。
   (3) 逻辑结构设计阶段：形成具体 DBMS 所支持的数据模型并优化。
   (4) 物理结构设计阶段：包括数据库的文件组织格式、内部存储结构、建立索引和表的聚集等。
   (5) 数据库实施阶段：建立数据库，编写应用程序，插入数据，功能和性能测试。
   (6) 数据库运行和维护阶段：运行、评价、调整与修改。
5. 数据库设计的特征
   (1) 反复性：设计方案需要和用户反复沟通。
   (2) 试探性：需要对不同方案进行试验、尝试。
   (3) 逐步进行：分阶段进行，后一阶段依赖前一阶段的结果。
6. 面向对象数据库设计：需求分析阶段：问题陈述 + 对象模型 + 动态模型 + 功能模型

8.2 数据库设计过程

8.2.1 需求分析

1. 调查
   (1) 信息要求：需要哪些数据、产生、得出哪些数据。
   (2) 处理要求：工作、统计、计算方法、数据量、速度。
   (3) 安全性和完整性要求：权限、数据之间的关系、限制。
2. 形成数据流图
   (1) 是现有系统逻辑模型描述工具。
   (2) 以图形方式，描述数据处理和流动的过程。
   (3) 圆表示加工、箭头表示数据流、双杠表示存储文件、方框表示数据的源点和终点。
3. 形成数据字典

(1) 数据项：数据的最小单位。
(2) 数据结构：若干个数据项的有意义的集合。
(3) 数据流：表示数据结构在系统内的传输路径。
(4) 数据存储：处理过程中存取的数据。
(5) 处理过程

8.2.2 概念设计

1. 概念设计的目的：建立数据的抽象模型。
2. 概念设计的方法：
   (1) 集中式模式设计法
   (2) 视图集成设计法
3. 概念设计的一般步骤：
   (1) 选择局部应用
   (2) 视图设计：自顶向下、由底向上、由内向外。
   (3) 视图集成：命名冲突、概念冲突、域冲突、约束冲突
4. 整体视图的要求：
   (1) 内部结构一致，不存在矛盾表达。
   (2) 准确反映原有视图结构，包括实体、属性、联系。
   (3) 满足需求分析的要求。
   (4) 应征求用户的意见，得到用户认可。

8.2.3 逻辑设计

1. **逻辑设计**：将 E-R 图转换成 DBMS 中的关系模式。
2. 转换时的问题：
   (1) 命名及属性域的处理。
   (2) 可再分的非原子属性的处理。
   (3) 联系的转换。
3. 逻辑模式规范化及其调整
   (1) 关系规范化
   (2) 模式优化
      (a) 减少连接运算。
      (b) 调整关系大小。
      (c) 使用快照（建立临时表，周期性更新，避免每次重新计算）。
4. 关系视图设计的作用：
   (1) 提供数据的逻辑独立性。
   (2) 适应用户对数据的不同需求。
   (3) 有一定的数据保密功能。