

我将为你准备一份详尽的、从零开始的 PostgreSQL 上手指南，特别针对有 MySQL 使用经验的团队。这份文档将涵盖从安装、概念差异、基本管理到使用 Perl 进行编程的完整流程。所有操作都基于 **Ubuntu 22.04** 系统。

PostgreSQL 全方位上手指南 (for MySQL Users)

本文档旨在帮助熟悉 MySQL 的开发和运维团队快速、平稳地在 Ubuntu 22.04 环境下开始使用 PostgreSQL。

目录

1. 引言：为什么是 PostgreSQL?
2. 第一部分：安装与初始化
 - 在 Ubuntu 22.04 上安装 PostgreSQL
 - 安装后的初始状态
3. 第二部分：核心概念对比 (PostgreSQL vs. MySQL)
 - 关键术语对照表
 - 核心差异详解 (认证、用户/角色、Schema)
4. 第三部分：命令行入门 (`psql`)
 - 连接数据库
 - 常用的 `psql` 元命令
5. 第四部分：数据库和用户管理实战
 - 创建新用户 (角色)
 - 创建新数据库
 - 配置远程/本地密码访问 (`pg_hba.conf`)
 - 授权
6. 第五部分：基本 SQL 操作
 - 创建表 (含自增主键)
 - 插入、查询、更新、删除数据
7. 第六部分：使用 Perl 连接和操作 PostgreSQL
 - 安装 Perl 驱动模块
 - Perl 示例代码 (完整的 CRUD 操作)
8. 第七部分：常用工具与后续学习
 - 图形化管理工具
 - 推荐资源

1. 引言：为什么是 PostgreSQL?

PostgreSQL (通常简称为 "Postgres") 是一个功能强大的开源对象关系型数据库系统 (ORDBMS) 。与 MySQL 相比, 它以以下几点著称:

- **高度的可扩展性:** 支持自定义函数、自定义数据类型、自定义索引方法等。
- **严格的 SQL 标准符合性:** 在遵循 SQL 标准方面通常比 MySQL 更严格。
- **更丰富的数据类型:** 原生支持 JSONB (高效的二进制 JSON)、数组、几何类型、网络地址类型等。
- **强大的事务控制:** 提供真正的ACID事务, 并且DDL (如 CREATE TABLE) 也可以在事务块中进行, 这在 MySQL (特别是使用 MyISAM 引擎时) 中是不同的。

对于习惯了 MySQL 的团队来说, 学习曲线主要在于理解其用户管理和网络访问控制的模式。

2. 第一部分: 安装与初始化

在 Ubuntu 22.04 上安装 PostgreSQL

Ubuntu 22.04 的官方仓库中包含了 PostgreSQL。安装过程非常直接。

```
# 1. 更新包列表
sudo apt update

# 2. 安装 PostgreSQL 和其 contrib 包
# contrib 包包含了一些非常有用的扩展和工具
sudo apt install postgresql postgresql-contrib

# 3. 验证安装
# 安装完成后, PostgreSQL 服务会自动启动并运行
systemctl status postgresql.service
```

如果看到 active (running) 的字样, 说明安装成功且服务已在运行。

安装后的初始状态

这部分是与 MySQL 最显著的不同点之一, 请务必理解:

1. **postgres 系统用户:** 安装过程会创建一个名为 postgres 的 Linux 系统用户。这个用户是 PostgreSQL 服务的拥有者。
2. **postgres 数据库用户:** 同时, 在数据库内部也创建了一个名为 postgres 的超级用户 (角色) 。
3. **peer 认证:** 默认情况下, PostgreSQL 使用 peer (对等) 认证方式进行本地连接。这意味着, 如果你的 Linux 用户名是 myuser, PostgreSQL 会尝试让你以数据库用户 myuser 的身份登录, 而不需要密码。因此, 要作为数据库超级用户 postgres 登录, 你必须先切换到 Linux 的 postgres 用户。

这就是为什么你不能直接用 psql -U postgres 登录, 而需要使用 sudo -u postgres 的原因。

3. 第二部分：核心概念对比 (PostgreSQL vs. MySQL)

理解这些术语和概念的差异是平稳过渡的关键。

关键术语对照表

MySQL 概念	PostgreSQL 对应概念	说明
(无直接对应)	Cluster (集群)	一个 PostgreSQL 服务实例管理的数据集合。在文件系统中表现为一个目录。一个集群可以包含多个数据库。
Database	Database	基本相同，都是表的集合。主要区别在于 PostgreSQL 的数据库隔离性更强。
User	Role (角色)	这是最大的区别！在 PostgreSQL 中，没有“用户”和“组”的区分，只有“角色”。一个角色可以登录（此时可称之为用户），也可以不登录（此时类似组）。角色可以继承其他角色的权限。CREATE USER 只是 CREATE ROLE ... LOGIN 的别名。
(无直接对应)	Schema (模式)	在一个数据库内部的命名空间。一个数据库可以有多个 Schema。CREATE TABLE my_table 默认会创建在 public 这个 Schema 下，完整名为 public.my_table 。这类似于在 C++ 中使用 namespace 。MySQL 中，数据库本身就充当了这一层命名空间。
AUTO_INCREMENT	SERIAL 或 IDENTITY	用于创建自增列。SERIAL 是一个语法糖，它会自动创建一个序列（Sequence）并与列关联。IDENTITY 是 SQL 标准方式，更推荐在新版本中使用。

核心差异详解

- 认证 (pg_hba.conf):
 - **MySQL:** 认证信息存储在 mysql.user 表中，主要基于用户名、来源主机和密码。
 - **PostgreSQL:** 认证由一个名为 pg_hba.conf 的配置文件控制（HBA = Host-Based Authentication）。这个文件定义了谁(USER)、可以从哪里(ADDRESS)、访问哪个数据库(DATABASE)、以及使用什么方法(METHOD)进行认证。常见方法有 peer , md5 (密码), trust (信任)。这是所有连接问题的首要排查点。
- 用户 vs. 角色:
 - **MySQL:** 用户和权限是分开实体。你创建一个用户，然后 GRANT 权限给他。
 - **PostgreSQL:** 一切皆角色。CREATE ROLE my_user LOGIN PASSWORD '...'; 创建了一个可以登录的角色（即用户）。CREATE ROLE db_admins; 创建了一个不能登录的角色（即

组)。然后你可以 `GRANT db_admins TO my_user;` , 让 `my_user` 继承 `db_admins` 的所有权限。

- **Schema:**
 - **MySQL:** 你通常这样写 `SELECT * FROM my_database.my_table;` 。
 - **PostgreSQL:** 连接到一个数据库后, 你通常这样写 `SELECT * FROM my_schema.my_table;` 。默认的 Schema 是 `public` , 所以 `SELECT * FROM my_table;` 等价于 `SELECT * FROM public.my_table;` 。使用 Schema 可以更好地组织数据库对象, 避免命名冲突, 并进行更细粒度的权限控制。

4. 第三部分：命令行入门 (psql)

`psql` 是 PostgreSQL 的交互式命令行工具, 功能极其强大, 类似于 MySQL 的 `mysql` 客户端。

连接数据库

如前所述, 要作为超级用户连接, 需要使用 `sudo` :

```
# 切换到 postgres 系统用户, 并打开 psql 客户端
# 这会连接到默认的 postgres 数据库
sudo -u postgres psql
```

你将看到类似 `postgres=#` 的提示符。

常用的 psql 元命令

这些命令在 `psql` 内部执行, 以反斜杠 `\` 开头, 不以分号 `;` 结尾。

psql 命令	MySQL 等效命令/操作	描述
<code>\l</code> 或 <code>\list</code>	<code>SHOW DATABASES;</code>	列出所有数据库
<code>\c db_name</code>	<code>USE db_name;</code>	连接到另一个数据库
<code>\dt</code>	<code>SHOW TABLES;</code>	列出当前数据库 (和Schema) 中的表
<code>\d table_name</code>	<code>DESCRIBE table_name;</code>	显示表的结构 (列、类型、索引等)
<code>\du</code>	<code>SELECT user, host FROM mysql.user;</code>	列出所有角色 (用户)
<code>\dn</code>	(无直接对应)	列出所有 Schema
<code>\q</code>	<code>exit</code> 或 <code>quit</code>	退出 <code>psql</code>

psql 命令	MySQL 等效命令/操作	描述
\?	(无直接对应)	显示所有 \ 命令的帮助

花几分钟时间熟悉这些命令，会对你的管理工作大有裨益。

5. 第四部分：数据库和用户管理实战

让我们来完成一个标准流程：创建一个新用户和新数据库，并配置其可以通过密码从本地登录。

场景: 我们要为项目 `my_project` 创建一个数据库 `my_project_db` 和一个用户 `my_project_user`。

第 1 步：登录 psql

```
sudo -u postgres psql
```

第 2 步：创建新用户（角色）

我们创建一个名为 `my_project_user` 的角色，并赋予它登录权限和密码。

```
-- 在 psql 提示符下执行
CREATE ROLE my_project_user WITH LOGIN PASSWORD 'a_very_secure_password';
```

- `WITH LOGIN`：表示这个角色可以用来登录，是 `CREATE USER` 的等效写法。
- `PASSWORD`：设置密码。请使用强密码。

第 3 步：创建新数据库

我们创建数据库，并指定所有者为我们刚刚创建的用户。

```
-- 在 psql 提示符下执行
CREATE DATABASE my_project_db OWNER my_project_user;
```

- `OWNER`：指定数据库的所有者。这是一种好的实践，意味着 `my_project_user` 对这个数据库有完全的控制权。

现在可以输入 `\q` 退出 `psql`。

第 4 步：配置密码访问 (`pg_hba.conf`)

这是最关键的一步。我们需要允许 `my_project_user` 使用密码从本地（`localhost`）连接到 `my_project_db`。

1. 找到 `pg_hba.conf` 文件。在 `psql` 中，你可以用 `SHOW hba_file;` 命令找到它的确切位置。通常在 `/etc/postgresql/14/main/pg_hba.conf` (版本号 14 可能不同)。

2. 编辑该文件。

```
# 版本号 14 可能会根据你的安装而变化
sudo nano /etc/postgresql/14/main/pg_hba.conf
```

3. 在文件末尾添加一行。找到类似下面这几行：

```
# "local" is for Unix domain socket connections only
local    all                postgres                                peer
# IPv4 local connections:
host     all                all                127.0.0.1/32                scram-sha-256
# IPv6 local connections:
host     all                all                ::1/128                    scram-sha-256
```

在这些行的上方（为了优先匹配）或下方添加我们自己的规则。推荐加在 `local` 规则的下面。

```
# TYPE DATABASE USER ADDRESS METHOD

# 允许 my_project_user 通过 Unix 套接字使用密码登录到任何数据库
local    all                my_project_user                                md5

# 如果你需要通过 TCP/IP（例如从 Perl 脚本）连接，也需要下面这行
host     all                my_project_user    127.0.0.1/32                md5
```

- `local`：表示通过 Unix domain socket 的本地连接。
- `host`：表示通过 TCP/IP 的连接。
- `all`：匹配所有数据库。你也可以指定为 `my_project_db`。
- `my_project_user`：指定用户名。
- `md5`：表示使用 MD5 加密的密码进行认证。这是 Perl DBI 模块连接时最常用的方式。
`scram-sha-256` 是更现代、更安全的方式，但 `md5` 的兼容性最好。

4. 重启 PostgreSQL 服务以应用更改。

```
sudo systemctl restart postgresql
```

第 5 步：测试连接

现在，你可以作为一个普通用户，尝试连接你新创建的数据库了。

```
psql -h localhost -U my_project_user -d my_project_db
```

系统会提示你输入密码。输入 `a_very_secure_password`，如果成功进入 `my_project_db=>` 提示符，恭喜你，配置成功！

6. 第五部分：基本 SQL 操作

连接到你的新数据库后 (`psql -h localhost -U my_project_user -d my_project_db`), 我们来执行一些基本的 SQL。

创建表

我们创建一个 `products` 表, 其中包含一个自增 ID。

```
-- 使用 SERIAL 来创建自增主键
CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    price NUMERIC(10, 2) NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);
```

```
-- 或者使用更现代的 IDENTITY (推荐)
CREATE TABLE products (
    id INT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    name VARCHAR(100) NOT NULL,
    price NUMERIC(10, 2) NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);
```

- `SERIAL`: 是 `integer NOT NULL DEFAULT nextval('products_id_seq'::regclass)` 的简写。
- `NUMERIC(10, 2)`: 用于精确存储货币值, 优于 `FLOAT`。
- `TIMESTAMP WITH TIME ZONE (timestampz)`: 存储时间戳并包含时区信息, 是处理时间的最佳实践。

插入数据 (INSERT)

```
INSERT INTO products (name, price) VALUES ('Laptop', 1200.50);
INSERT INTO products (name, price) VALUES ('Mouse', 25.00);
INSERT INTO products (name, price) VALUES ('Keyboard', 75.99);
```

ID 会自动生成。

查询数据 (SELECT)

```
SELECT * FROM products;
SELECT * FROM products WHERE price > 100;
```

更新数据 (UPDATE)

```
UPDATE products SET price = 1150.00 WHERE name = 'Laptop';
```

删除数据 (DELETE)

```
DELETE FROM products WHERE name = 'Mouse';
```

这些 SQL 语法与 MySQL 基本一致，你应该会感到非常熟悉。

7. 第六部分：使用 Perl 连接和操作 PostgreSQL

现在，让我们用 Perl 代码来操作我们刚刚创建的数据库和表。

安装 Perl 驱动模块

你需要 DBI (数据库通用接口) 和 DBD::Pg (PostgreSQL 的驱动)。

```
# 使用 apt 安装，这是最简单的方式，可以处理好系统依赖
sudo apt install libdbd-pg-perl
```

```
# 如果你习惯使用 cpan，也可以用 cpanm
# sudo cpanm DBI
# sudo cpanm DBD::Pg
```

Perl 示例代码 (完整的 CRUD 操作)

下面是一个完整的 Perl 脚本，演示了如何连接数据库，并执行创建、读取、更新和删除操作。

```
#!/usr/bin/perl

use strict;
use warnings;
use DBI;

# --- 数据库连接配置 ---
my $db_name = 'my_project_db';
my $host     = 'localhost';
my $port     = '5432'; # 默认端口
my $user     = 'my_project_user';
my $pass     = 'a_very_secure_password';

# --- 构建 DSN (Data Source Name) ---
# PostgreSQL 的 DSN 格式是 "dbi:Pg:dbname=...;host=...;port=..."
my $dsn = "dbi:Pg:dbname=$db_name;host=$host;port=$port";

# --- 声明数据库句柄 ---
my $dbh;
```

```

# --- 1. 连接数据库 ---
# 使用 RaiseError => 1 可以在发生错误时自动 die
# 使用 PrintError => 0 禁用自动打印错误（因为 RaiseError 会处理）
# 使用 AutoCommit => 1（默认）或 0（用于事务）
eval {
    $dbh = DBI->connect($dsn, $user, $pass, {
        RaiseError => 1,
        PrintError => 0,
        AutoCommit => 1,
    });
    print "Successfully connected to PostgreSQL database '$db_name'!\n\n";
};
if ($?) {
    die "Database connection failed: $@";
}

# --- 2. 准备和清理表 ---
print "--- Preparing table 'employees' ---\n";
# 使用 IF EXISTS 避免在表不存在时报错
$dbh->do("DROP TABLE IF EXISTS employees");
$dbh->do(q{
    CREATE TABLE employees (
        id INT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
        first_name VARCHAR(50) NOT NULL,
        last_name VARCHAR(50) NOT NULL,
        department VARCHAR(50),
        salary NUMERIC(12, 2)
    )
});
print "Table 'employees' created successfully.\n\n";

# --- 3. 插入数据（CREATE） ---
print "--- Inserting new employees ---\n";
my $insert_sql = "INSERT INTO employees (first_name, last_name, department, salary) VA
my $sth = $dbh->prepare($insert_sql);

# 使用占位符 (?) 是防止 SQL 注入的最佳实践
$sth->execute('John', 'Doe', 'Engineering', 80000);
$sth->execute('Jane', 'Smith', 'Marketing', 75000);
$sth->execute('Peter', 'Jones', 'Engineering', 92000);

# 使用 execute_for_fetch 可以一次性插入多行，但这里分开展示更清晰
print "3 employees inserted.\n\n";

# --- 4. 查询数据（READ） ---
print "--- Querying all employees ---\n";
my $select_sql = "SELECT id, first_name, last_name, department, salary FROM employees
$sth = $dbh->prepare($select_sql);
$sth->execute();

```

```

# 使用 fetchrow_hashref 循环获取每一行，结果是一个哈希引用
while (my $row = $sth->fetchrow_hashref) {
    printf "ID: %d, Name: %s %s, Department: %s, Salary: %.2f\n",
        $row->{id},
        $row->{first_name},
        $row->{last_name},
        $row->{department},
        $row->{salary};
}
print "\n";

# --- 5. 更新数据 (UPDATE) ---
print "--- Updating John Doe's salary ---\n";
my $update_sql = "UPDATE employees SET salary = ? WHERE first_name = ? AND last_name = ?";
$sth = $dbh->prepare($update_sql);
my $rows_affected = $sth->execute(85000, 'John', 'Doe');
print "$rows_affected row(s) updated.\n\n";

# 验证更新
print "--- Verifying update ---\n";
$sth = $dbh->prepare("SELECT salary FROM employees WHERE first_name = 'John'");
$sth->execute();
my $new_salary = $sth->fetchrow_array();
print "John Doe's new salary is: $new_salary\n\n";

# --- 6. 删除数据 (DELETE) ---
print "--- Deleting Jane Smith ---\n";
my $delete_sql = "DELETE FROM employees WHERE first_name = ? AND last_name = ?";
$sth = $dbh->prepare($delete_sql);
$rows_affected = $sth->execute('Jane', 'Smith');
print "$rows_affected row(s) deleted.\n\n";

# --- 7. 断开连接 ---
$dbh->disconnect();
print "Successfully disconnected from the database.\n";

exit 0;

```

如何运行这个脚本：

1. 将代码保存为 `pg_test.pl` 。
2. 确保你已经完成了第四部分的所有步骤（创建了用户和数据库，并配置了 `pg_hba.conf`）。
3. 在终端中运行：`perl pg_test.pl` 。

你将看到脚本逐步执行并打印出每一步的结果。

8. 第七部分：常用工具与后续学习

图形化管理工具

虽然 `psql` 很强大，但 GUI 工具在日常开发和数据探查中也非常方便。

- **pgAdmin:** PostgreSQL 官方的开源管理工具。功能非常全面，从查询、设计到服务器监控无所不包。
- **DBeaver:** 一款优秀的开源通用数据库工具，支持 PostgreSQL、MySQL、SQLite 等多种数据库。如果你的团队同时使用多种数据库，这是一个极佳的选择。
- **DataGrip:** JetBrains 出品的商业数据库 IDE，功能强大，如果你已经在使用 IntelliJ IDEA 或 PhpStorm，会感到非常亲切。

推荐资源

- **官方文档:** [PostgreSQL Documentation](#) 是最权威、最准确的信息来源。当你遇到问题时，首先应该查阅这里。
- **PostgreSQL Tutorial:** 一个对初学者非常友好的网站 [postgresql-tutorial.com](#)。

总结

从 MySQL 过渡到 PostgreSQL，你需要关注的主要是**概念上的差异**，特别是围绕**角色（Role）**和**认证配置（pg_hba.conf）**。一旦你掌握了这些，你会发现 PostgreSQL 在 SQL 层面与 MySQL 非常相似，同时它提供了更多高级和强大的功能。

希望这份详尽的指南能帮助你的团队顺利开启 PostgreSQL 之旅！如果在实践中遇到任何问题，随时可以继续提问。