

数据库基础

データベースの基本

- 数据库基本概念 データベース概要
- PostgreSQL数据库安装 PostgreSQLデータベースのインストール
- SQL语言基础 SQL言語の基本

练习一 时间： 15min

- 创建一个人类，拥有名字，生日（字符串20210101）等属性
- getAge () 方法，根据生日属性计算出现年的年龄并返回（年龄为 int型）。
- 重写equals () 方法，如果两个人的名字和生日均相同，equals返回 true，否则返回false

练习二 时间： 20min

- 知识补充：[java.lang.Comparable](#)<T>是一个函数是接口，用来定义如何比较两个T类的对象。
- 实现Comparable接口的类必须要实现它的[compareTo\(T o\)](#)方法
- [a.compareTo\(b\)](#)：如果a比b大则返回正数，相等返回0，否则返回负数。
- 例：通过重写compareTo方法，定义如何比较两个数字大小

```
1 import java.lang.Comparable;
2 public class Number implements Comparable{
3     int num;
4
5     public Number(int num){
6         this.num = num;
7     }
8
9     public int getNum() {
10        return num;
11    }
12
13    @Override
14    public int compareTo(Object numObject) {
15        int num2 = ((Number)numObject).getNum();
16        if(num2 > this.num) return -1;
17        else if(num2 == this.num) return 0;
18        return 1;
19    }
20
21    Run | Debug
22    public static void main(String args[]) {
23        Number n1 = new Number(2);
24        Number n2 = new Number(3);
25        System.out.println(n1.compareTo(n2)); // output -1
26    }
}
```

练习二 时间： 20min

- 使用Comparable接口来实现人类对象的比较方式
- A和B是两个人类对象， A.compareTo(B):
 - A比B的年龄大， compareTo返回正数
 - A和B的年龄相同， compareTo返回0
 - A比B的年龄小， compareTo返回负数



1

数据库基本概念

データベース概要

2

PostgreSQL数据库安装

PostgreSQLのインストール

3

SQL语言基础

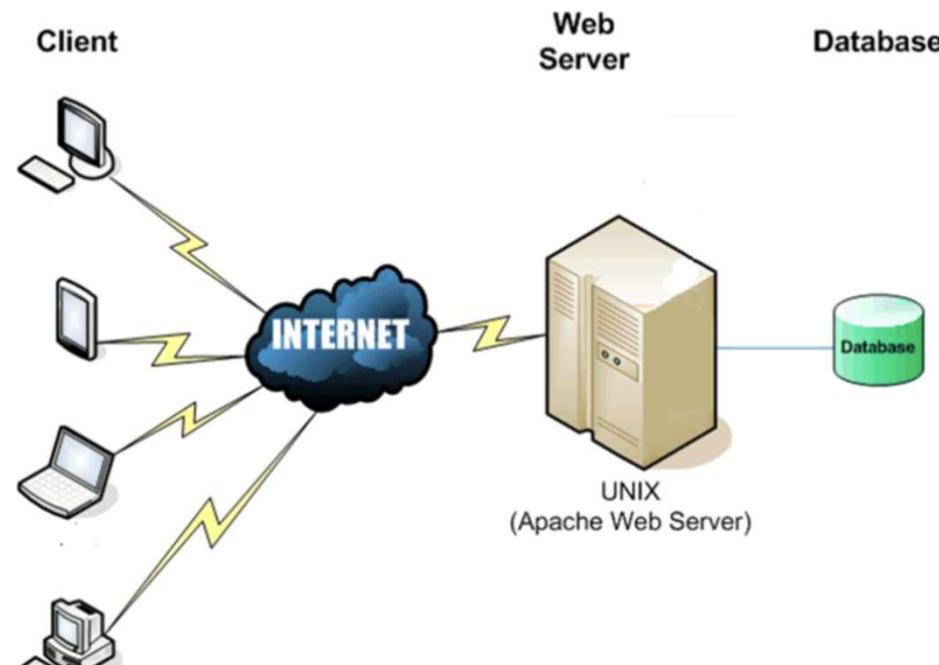
SQL言語の基本

数据库概述

- 数据库(Database, DB, データベース)是按照一定的数据结构来组织、存储、共享和管理数据的仓库。
- 特征
 - 数据按一定的数据模型组织、描述和存储;
 - 可为各种用户共享;
 - 冗余度较小;
 - 数据独立性较高;
 - 易扩展;
 - 是数据库系统DBS的核心，是被管理的对象;

数据库在网站中的地位

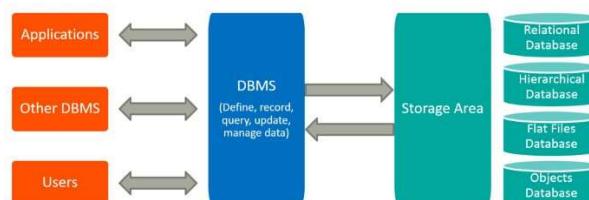
- 要创建一个网站，至少需要以下部分：
 - 使用HTML/CSS/JavaScript创建网页页面（已学习）
 - RDBMS（关系型数据库[関係データベース]管理系统）（如PostgreSQL, SQL Server, MySQL等）
 - 使用SQL语言对RDBMS进行访问和操作，获取所需数据
 - 服务器端API（待学习）



数据库管理系统 (Database Management System)

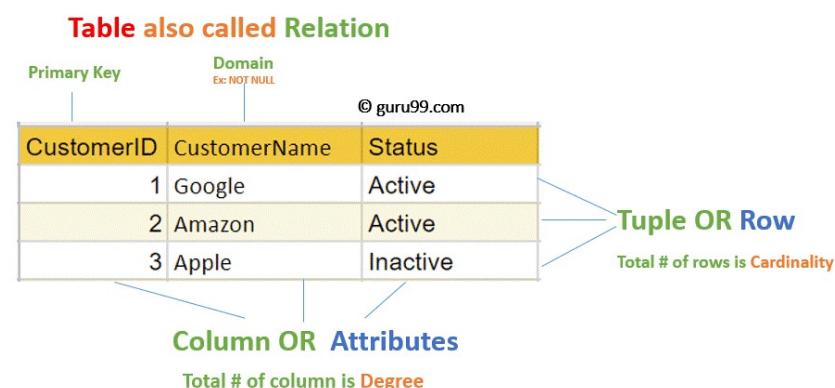
- 数据库管理系统[データベース管理システム](简称DBMS), 是对数据库进行管理的软件
 - 基本功能
 - 数据定义
 - 数据定义语言[データ定義言語](Data Definition Language, 简称DDL), 如create、 alter、 drop、 truncate。
 - 数据操作
 - 数据操纵语言[データ操作言語](Data Manipulation Language, 简称DML), 如数据的插入、 删除、 查询、 修改(增删查改)。
 - 数据控制
 - 数据控制语言[データ制御言語](Data Control Language, 简称DCL), 如分配权限。
 - 数据库的建立与维护

Database Management System



关系型数据库 [関係データベース] (Relational DB)

- 关系模型中常用的概念：
 - 关系：可以理解为一张二维表，每个关系都具有一个关系名，就是通常说的表名。例如：成绩表（学号，姓名，数学，英语）
 - 元组[组] (tuple)：二维表中的一行，在数据库中经常被称为记录
 - 属性：二维表中的一列，在数据库中经常被称为字段
 - 域[定義域] (domain)：属性的取值范围，也就是数据库中某一列的取值限制
 - 主键[主キー]：这个值能够唯一地表示一条记录的字段或者字段的组合(候选主键)。例如，学号就是唯一的，可以作为主键[主キー]。
 - 外键[外部キー]：一个表中的某个字段不是本表的主键[主キー]，而是另外一个表的主键[主キー]或者候选主键，那么这个字段就是外键[外部キー]。
 - 关系模式：指对关系的描述。其格式为：关系名(属性1, 属性2,, 属性N)，在数据库中成为表结构



关系型数据库[関係データベース]特点

- 关系必须规范化，属性不可分割，表中不能包含表；
- 在同一个关系中不能出现相同的属性名；
- 关系中不允许有完全相同的记录(冗余信息)；
- 同一关系中记录的顺序无关紧要；
- 同一关系中属性的顺序无关紧要；

关系运算符

- 定义：用户需要利用查询从关系数据库中找到感兴趣的数据时，需要对多个关系(表)进行运算。
- 分类
 - 关系运算以关系代数为基础，关系的基本运算分为两类
 - 传统集合运算
 - 并、交、差、笛卡尔积
 - 专门集合运算
 - 选择、投影、连接

关系运算符一并 (OR)

- 并(\cup)，设关系R和S有相同的结构，则 $R \cup S$ 由属于R或属于S的元组[組]组成(删去重复的记录)。

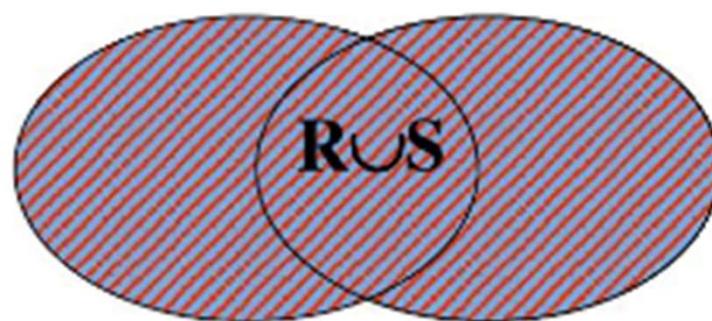
ID	Name	Gender
01	Rose	Female
02	Jack	Male

 \cup

ID	Name	Gender
02	Jack	Male
03	Micheal	Male

 $=$

ID	Name	Gender
01	Rose	Female
02	Jack	Male
03	Micheal	Male



关系运算符—交 (AND)

- 交(\cap)，设关系R和S有相同的结构，则 $R \cap S$ 由既属于R又属于S的元组[組]组成。

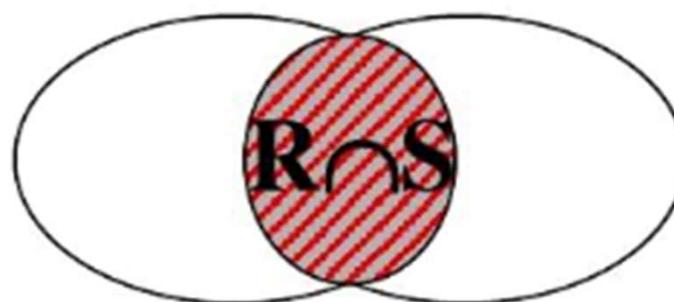
ID	Name	Gender
01	Rose	Female
02	Jack	Male

 \cap

ID	Name	Gender
02	Jack	Male
03	Micheal	Male

 $=$

ID	Name	Gender
02	Jack	Male



关系运算符--差

- 差(-)，设关系R和S有相同的结构，则 $R-S$ 由属于R但不属于S的元组[組]组成。

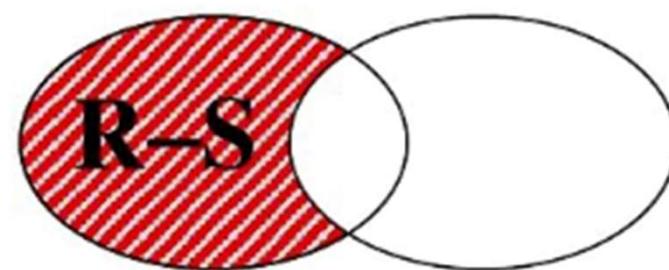
ID	Name	Gender
01	Rose	Female
02	Jack	Male

-

ID	Name	Gender
02	Jack	Male
03	Micheal	Male

=

ID	Name	Gender
01	Rose	Female



关系运算符—笛卡尔积

- 笛卡尔积(\times)，设n元关系R和m元关系S，则 $R \times S$ 是一个 $n \times m$ 元组的集合。
- 注意：R和S关系的结构不必相同。

Subject
Python
Java

X

ID	Name	Gender
01	Rose	Female
02	Jack	Male

=

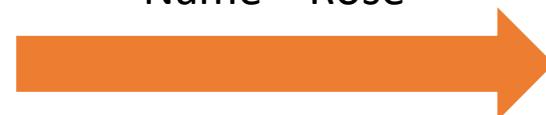
Subject	ID	Name	Gender
Python	01	Rose	Female
Python	02	Jack	Male
Java	01	Rose	Female
Java	02	Jack	Male

关系运算符—选择 (Select)

- 选择，从关系中选择满足一定条件的元组[組]。
- 选择条件中常常包含着大于小于，与或非的运算。

ID	Name	Gender
01	Rose	Female
02	Jack	Male

Name="Rose"



ID	Name	Gender
01	Rose	Female

ID	Name	Gender
01	Rose	Female
02	Jack	Male

Name, Gender



Name	Gender
Rose	Female
Jack	Male

Q & A

You Have Questions
We Have Answers



1

数据库基本概念 データベース概要

2

PostgreSQL数据库安装 PostgreSQLのインストール

3

SQL语言基础 SQL言語の基本

PostgreSQL简介

- PostgreSQL 是一个免费开源的对象-关系数据库管理系统[データベース管理システム](ORDBMS)，具有大型商业RDBMS中所具有的特性，包括事务、子选择、触发器、视图、外键[外部キー]引用完整性和复杂锁定等功能。



PostgreSQL下载（请大家严格按照步骤进行）

- 下载地址：

<https://www.enterprisedb.com/downloads/postgres-postgresql-downloads>

Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
13.2	N/A	N/A	Download	Download	N/A
12.6	N/A	N/A	Download	Download	N/A
11.11	N/A	N/A	Download	Download	N/A
10.16	Download				
9.6.21	Download				
9.5.25 (Not Supported)	Download				
9.4.26 (Not Supported)	Download				

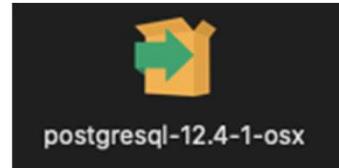
安装过的同学不用再次安装

PostgreSQL安装步骤1 (Mac)

- 双击下载好的dmg文件



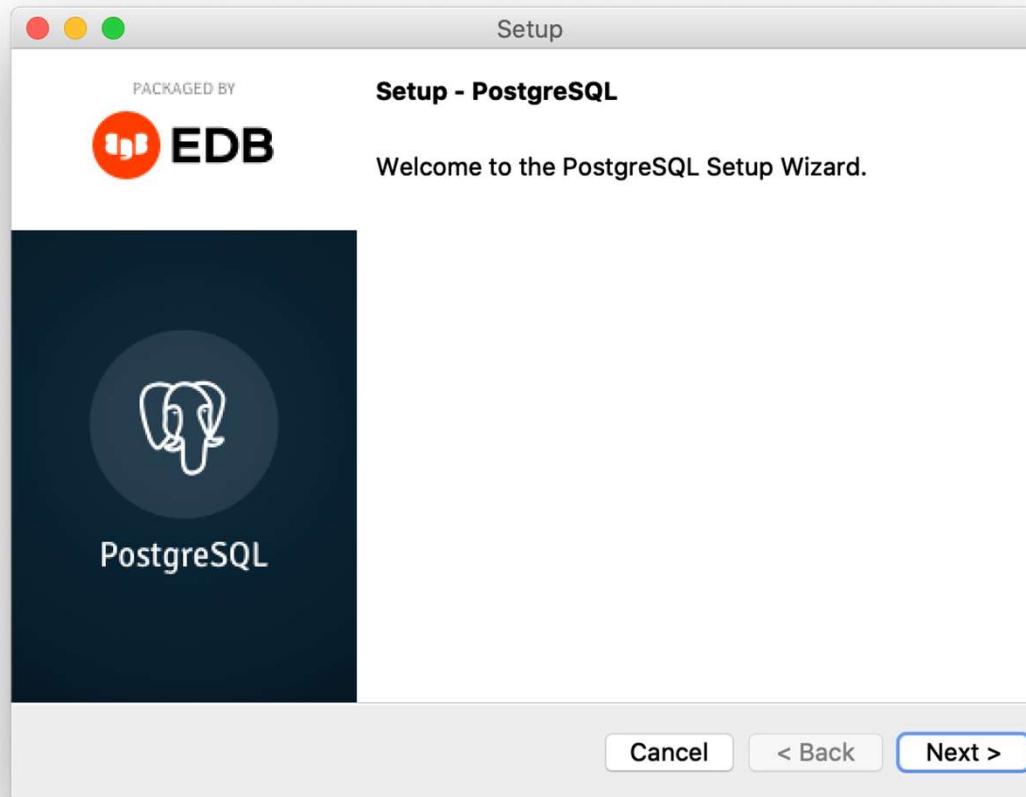
- 双击-->



- 可能会有权限警告，选择确定，并输入开机密码

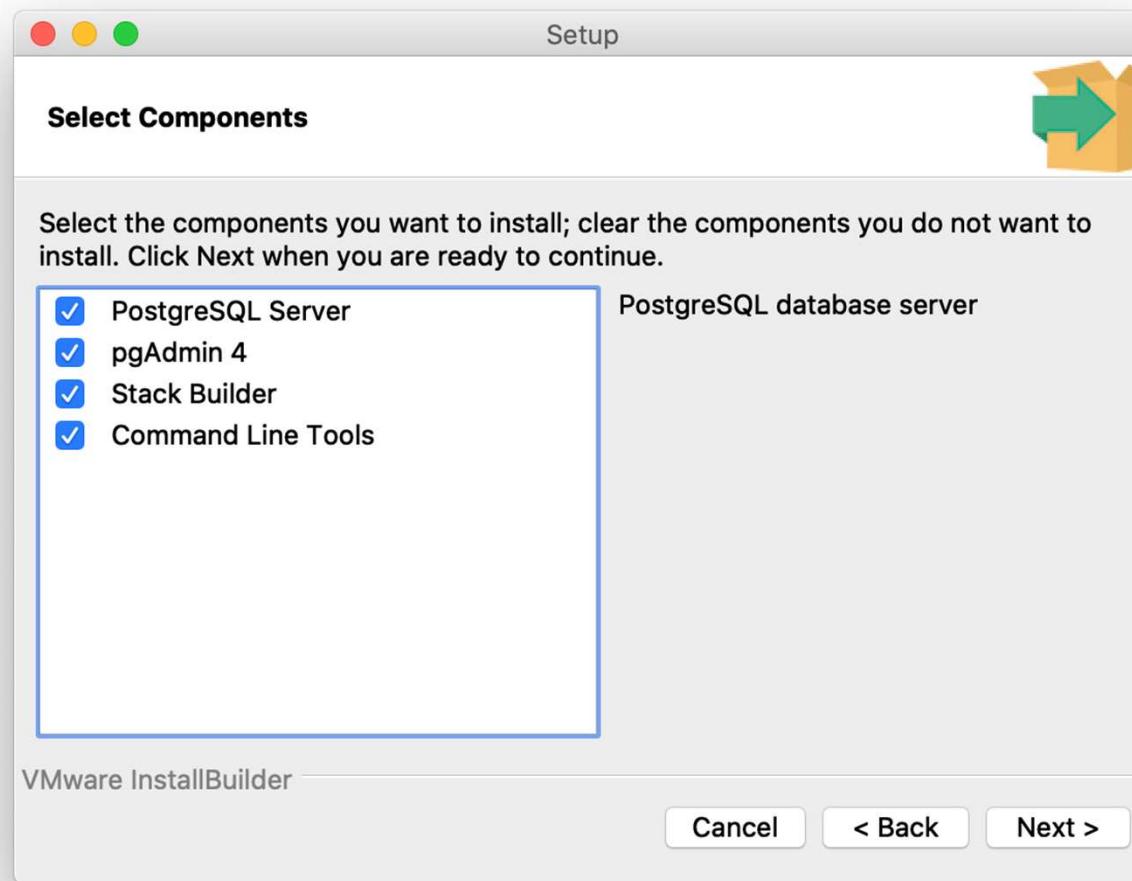
PostgreSQL安装步骤2 (Mac)

- Next, 直到步骤3
- 保存路径可以自定义



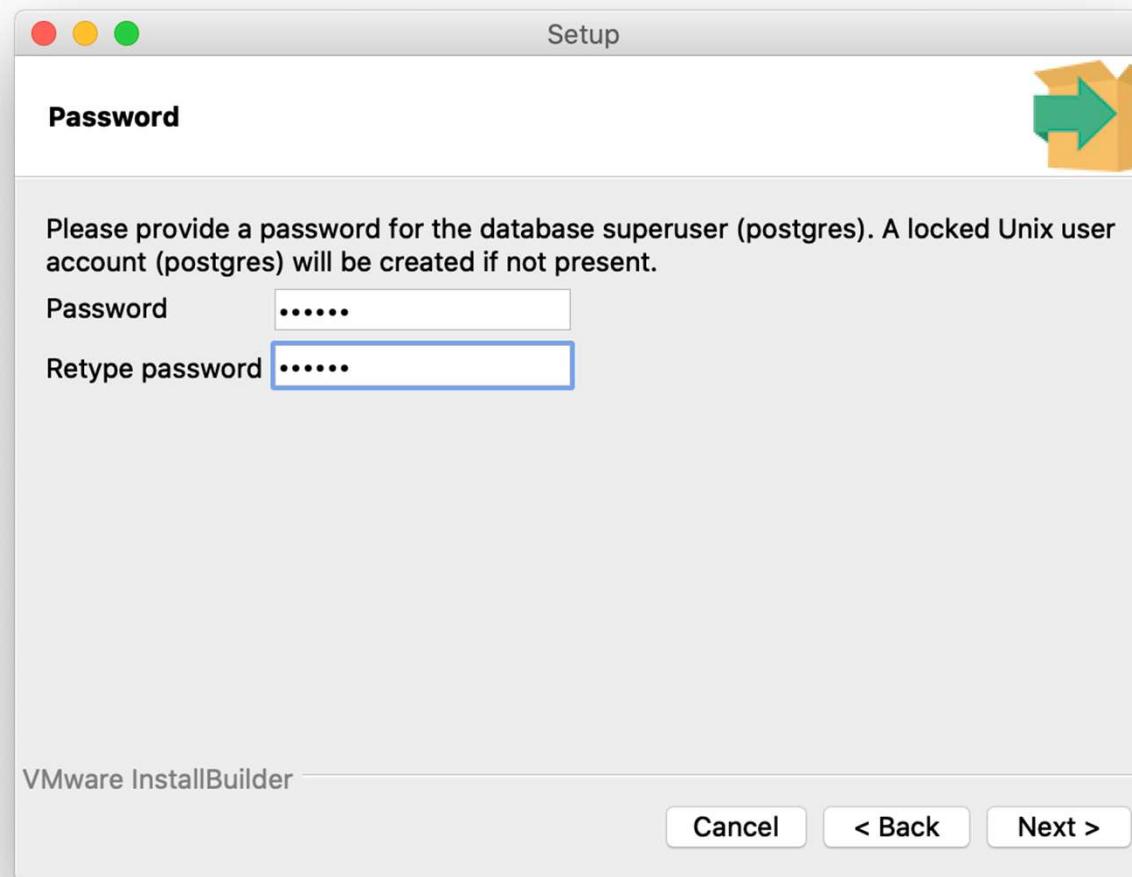
PostgreSQL安装步骤3 (Mac)

- 四个组件全选， Next



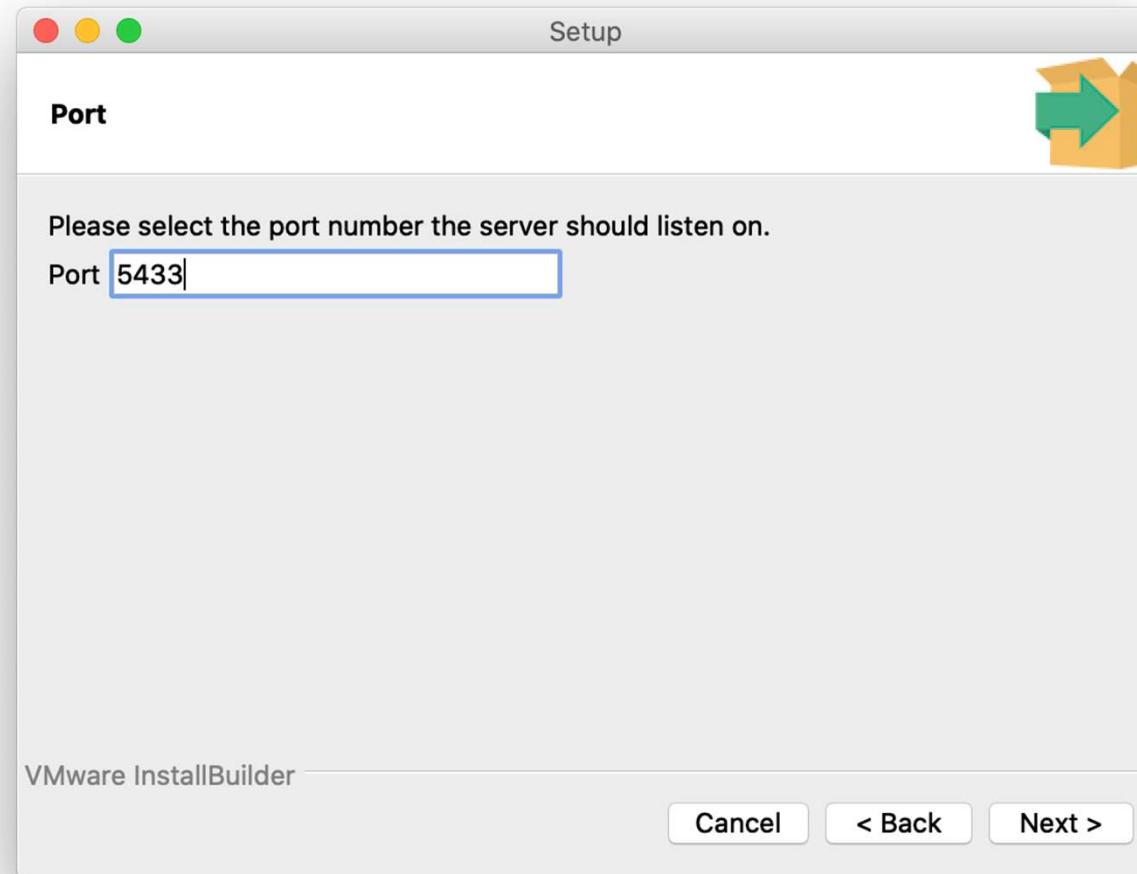
PostgreSQL安装步骤4 (Mac)

- 这里设置数据库密码 (这里为了学习, 建议大家都设置为123456)



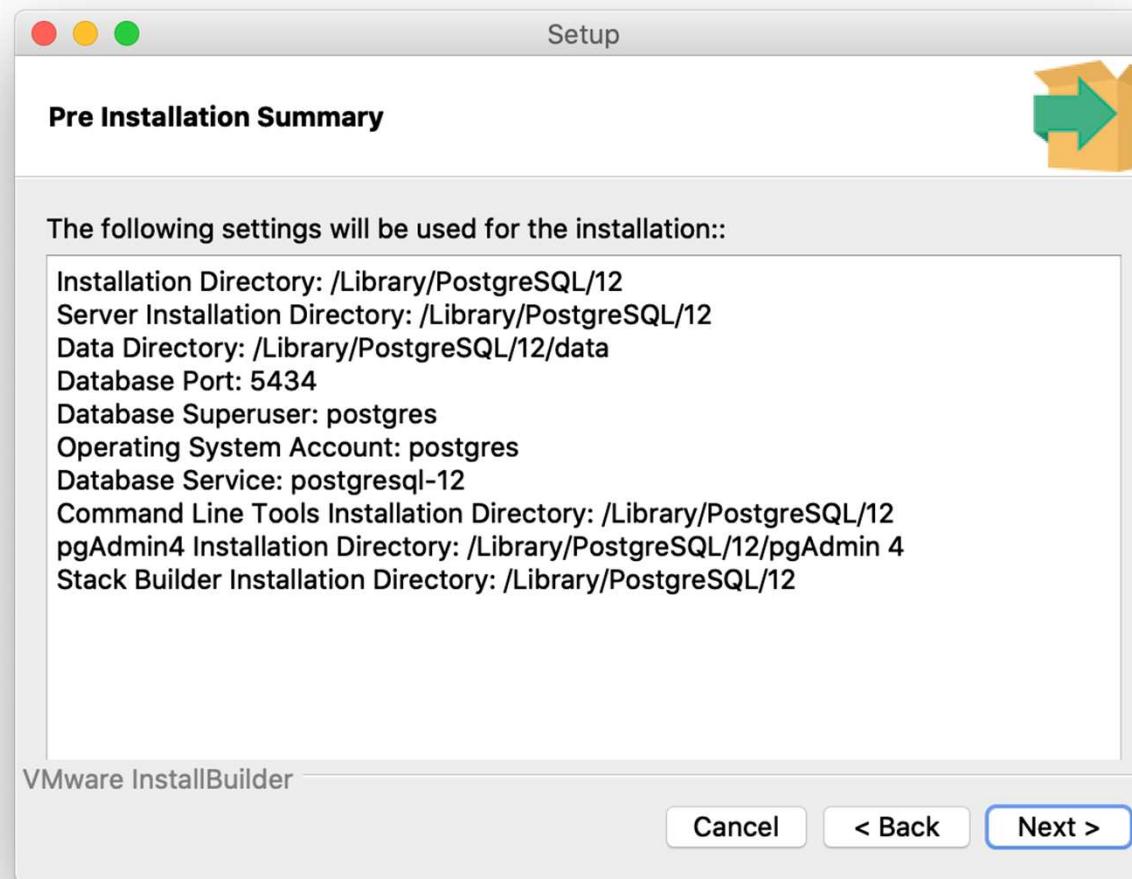
PostgreSQL安装步骤5 (Mac)

- 这里使用默认的端口号 (Port) 就可以 (5433)
- 如果不行，则使用5434



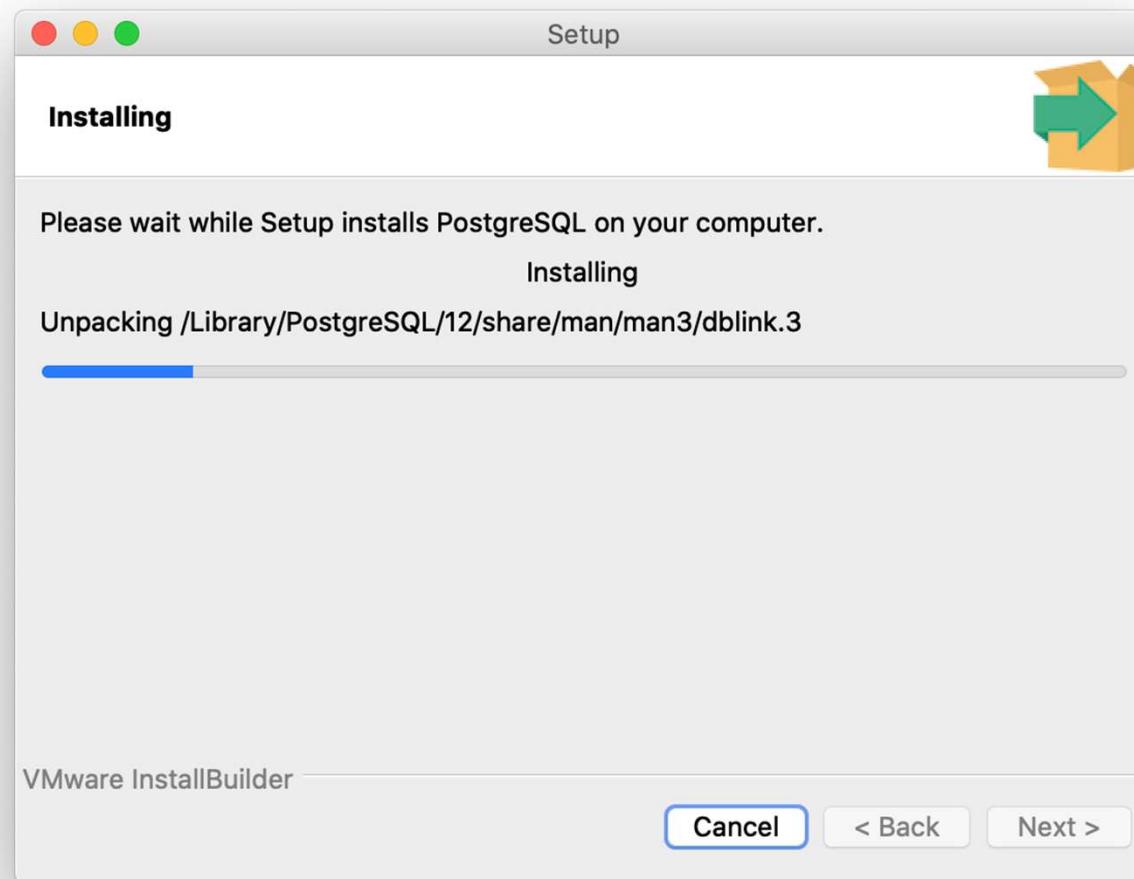
PostgreSQL安装步骤6 (Mac)

- 到这一步请截图保留， Next



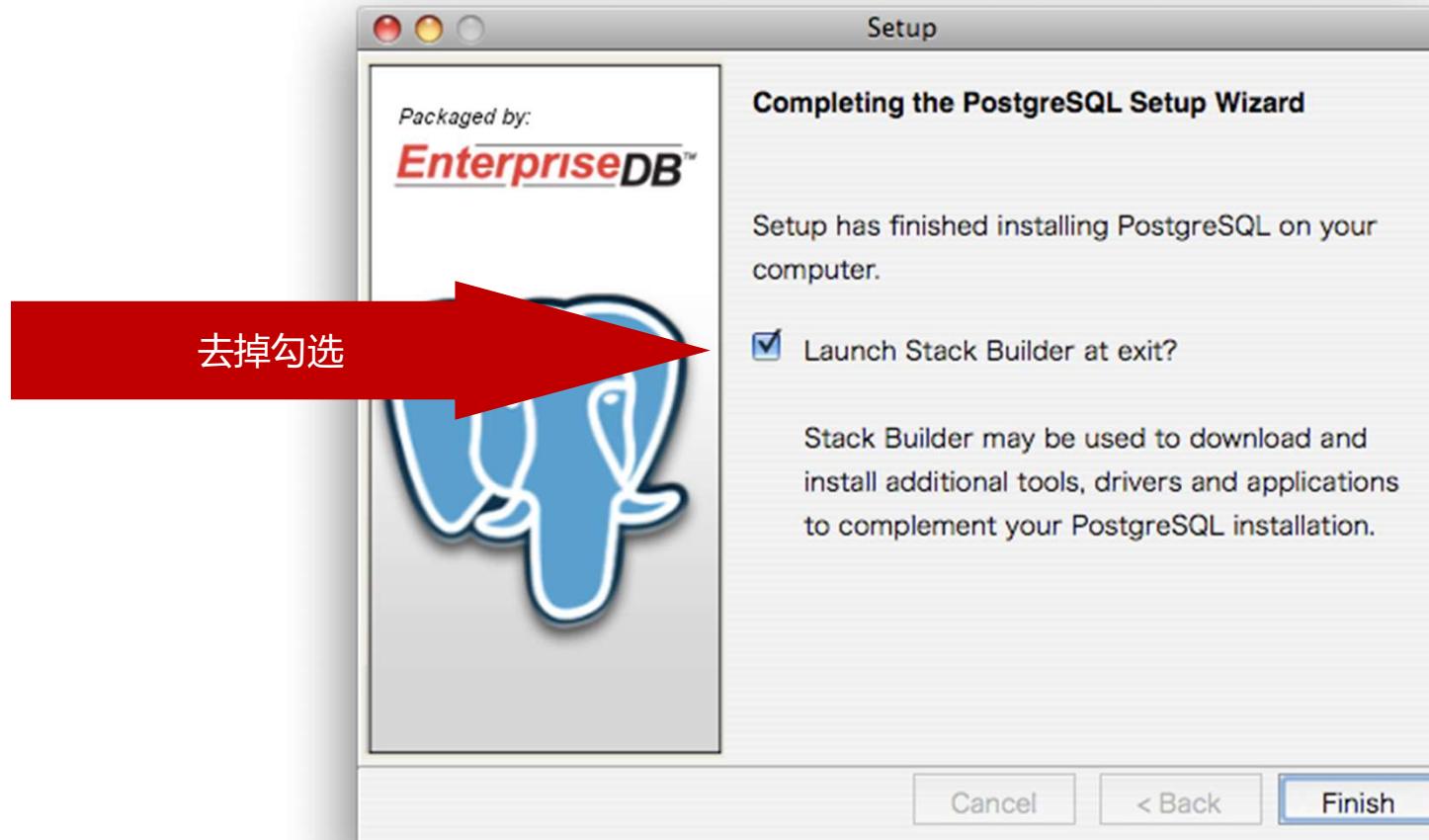
PostgreSQL安装步骤6 (Mac)

- 一直Next到下面界面，等待安装结束



PostgreSQL安装步骤7 (Mac)

- 去掉勾选, Finish,



PostgreSQL安装步骤8 (Mac)

- 安装完成后，应用里面将会多三个图标



PostgreSQL可视化交互界面



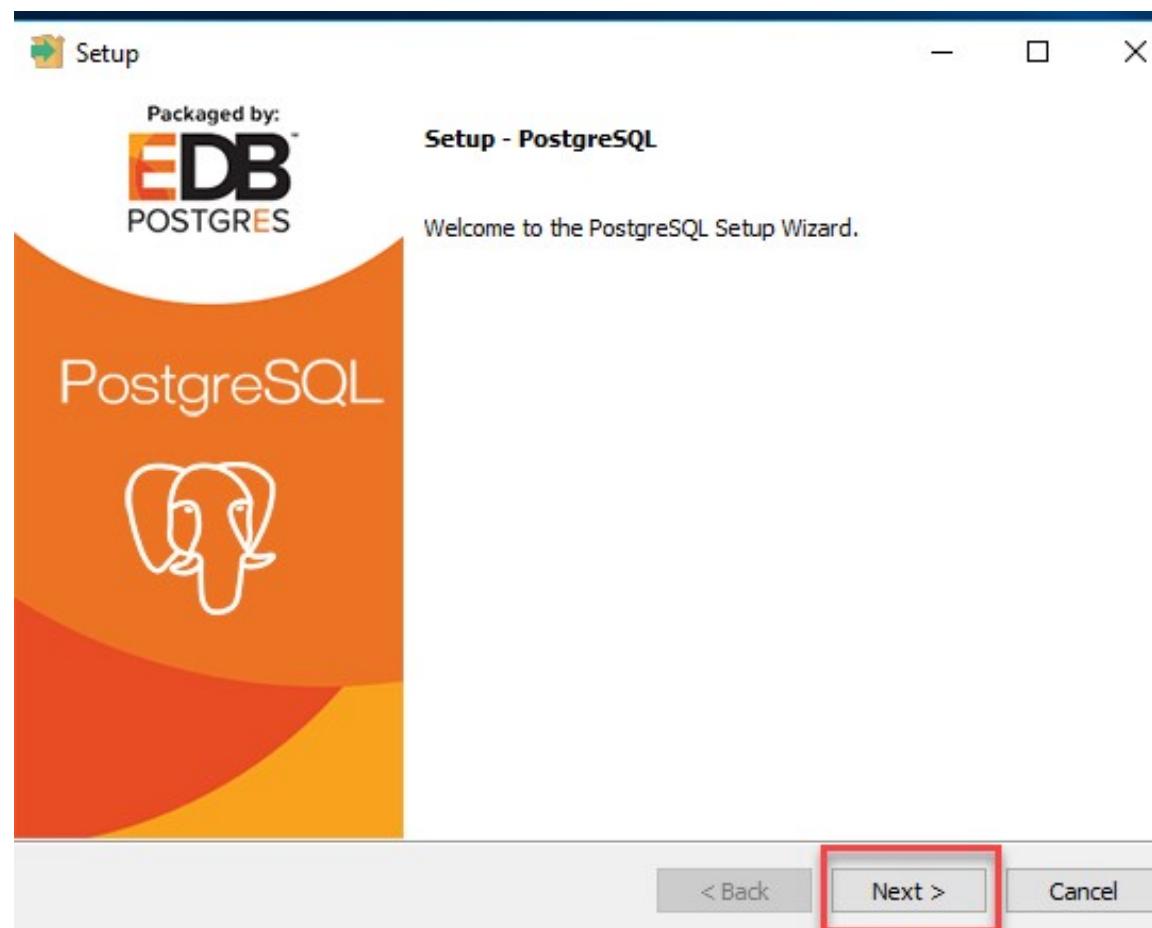
PostgreSQL说明文档



PostgreSQL 命令行

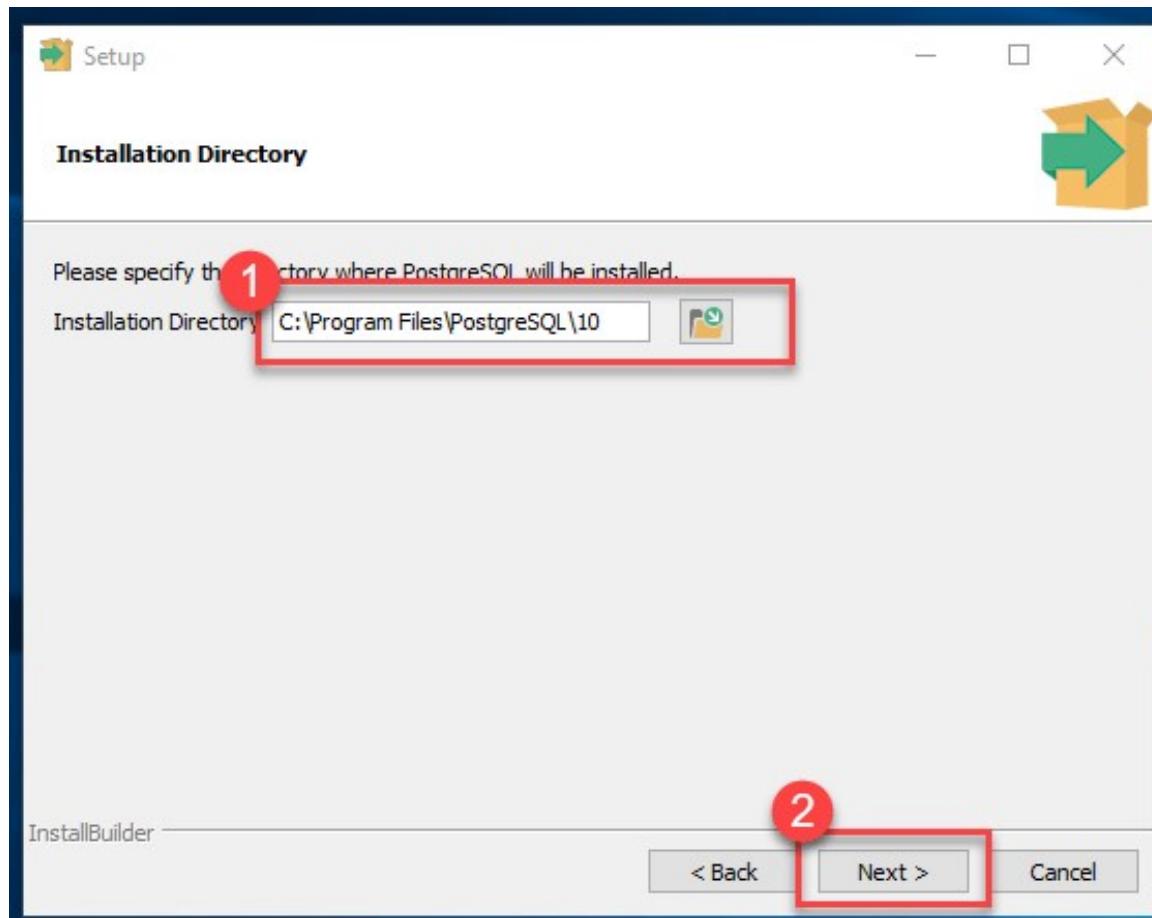
PostgreSQL安装步骤1 (Win)

- 双击下载安装包，开始安装



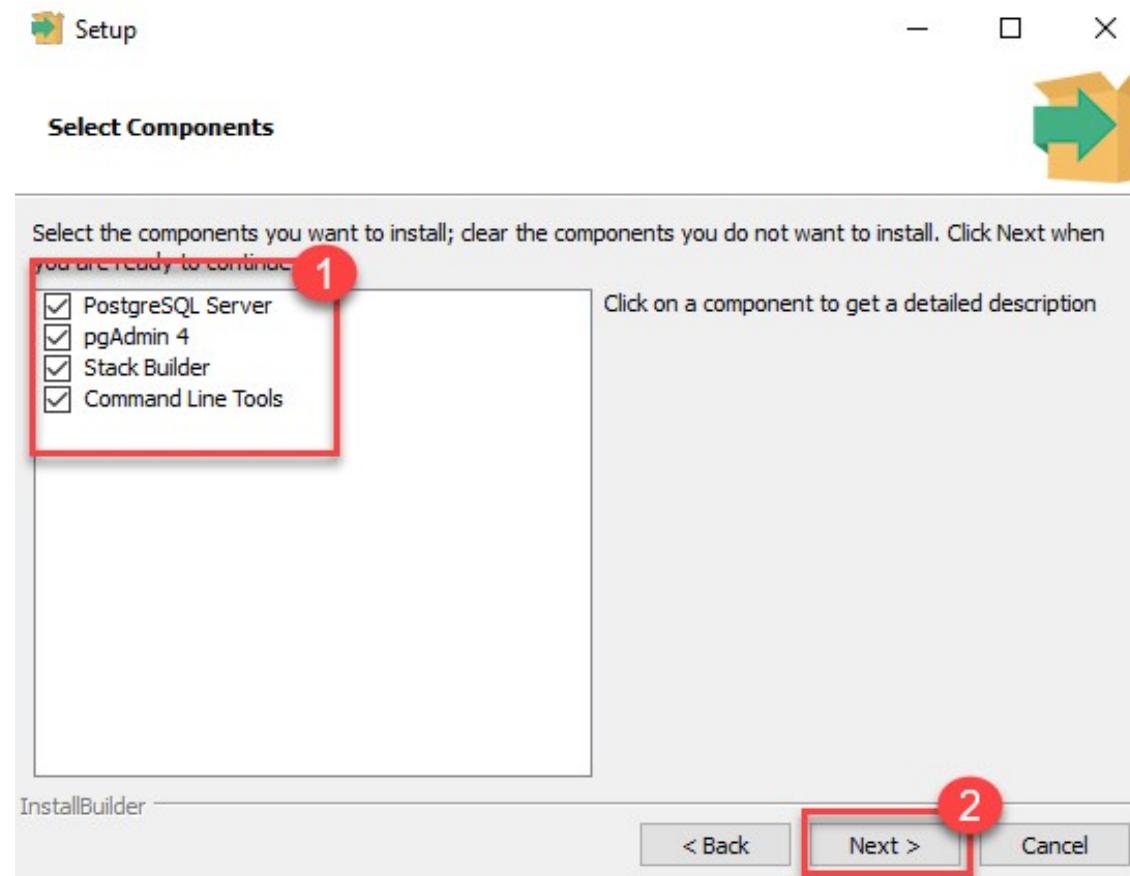
PostgreSQL安装步骤1 (Win)

- 建议使用默认安装路径



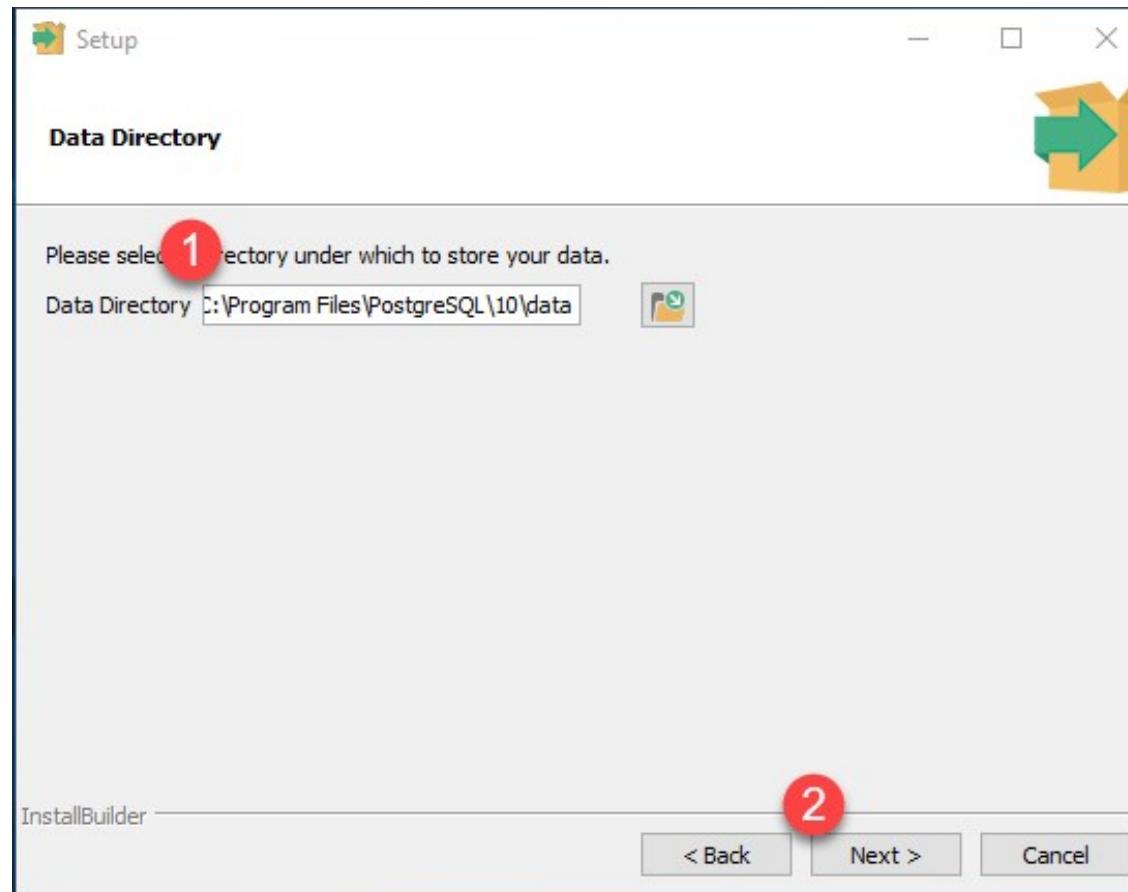
PostgreSQL安装步骤2 (Win)

- 全部勾选



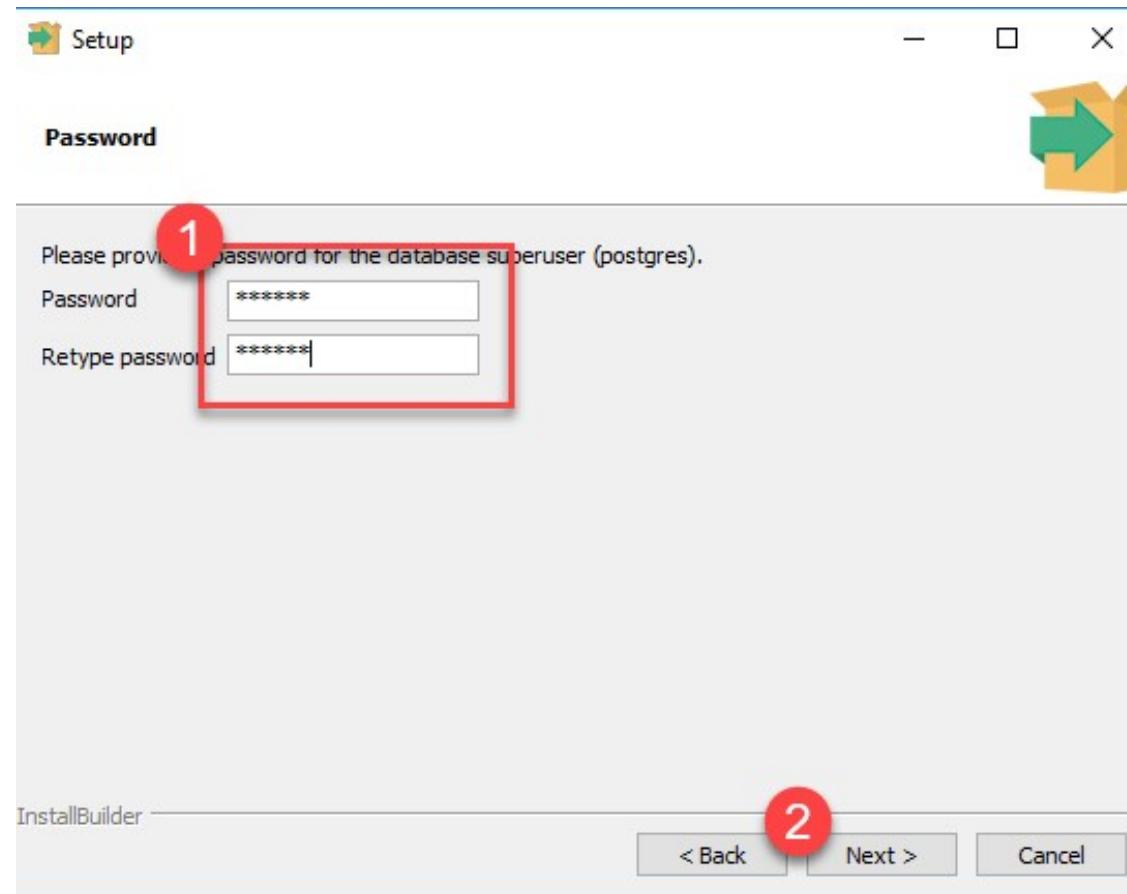
PostgreSQL安装步骤3 (Win)

- 建议使用默认数据库路径



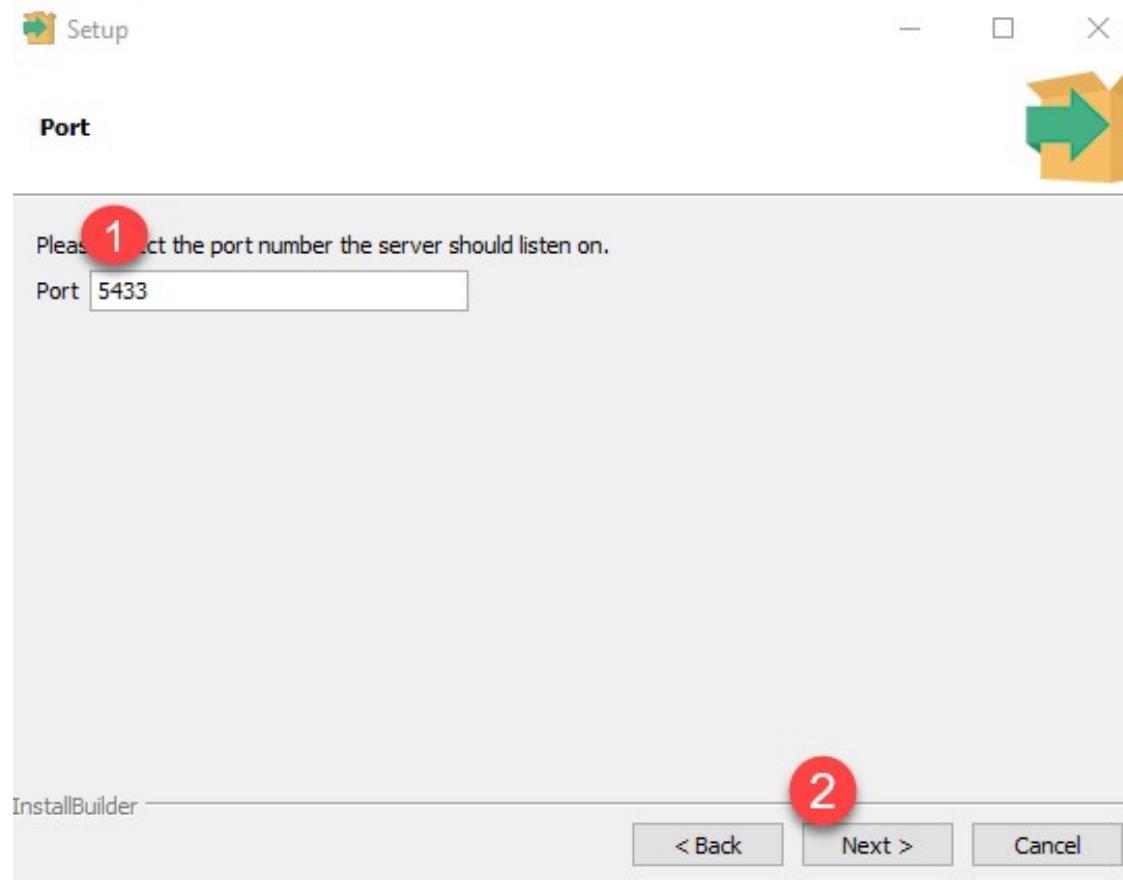
PostgreSQL安装步骤4 (Win)

- 设置超级用户的密码，建议设置为123456



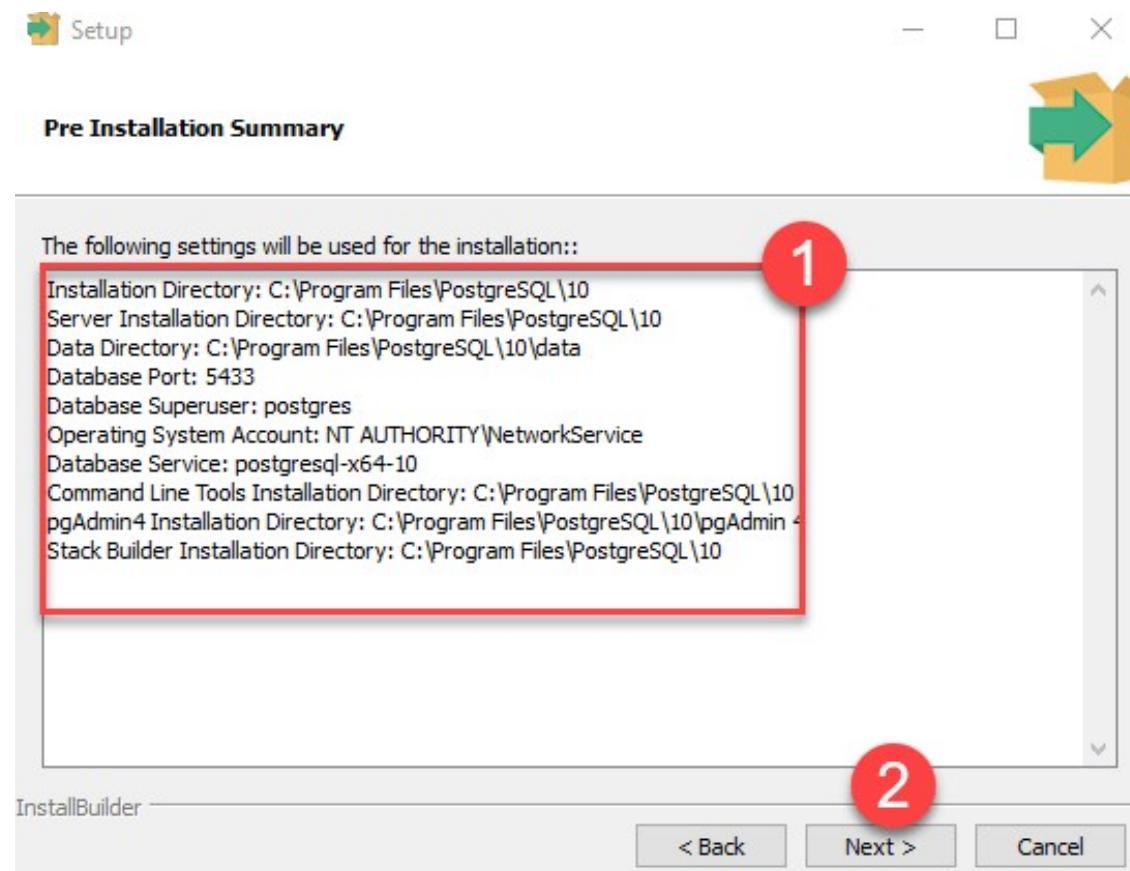
PostgreSQL安装步骤5 (Win)

- 设置端口号，可以直接用默认就行



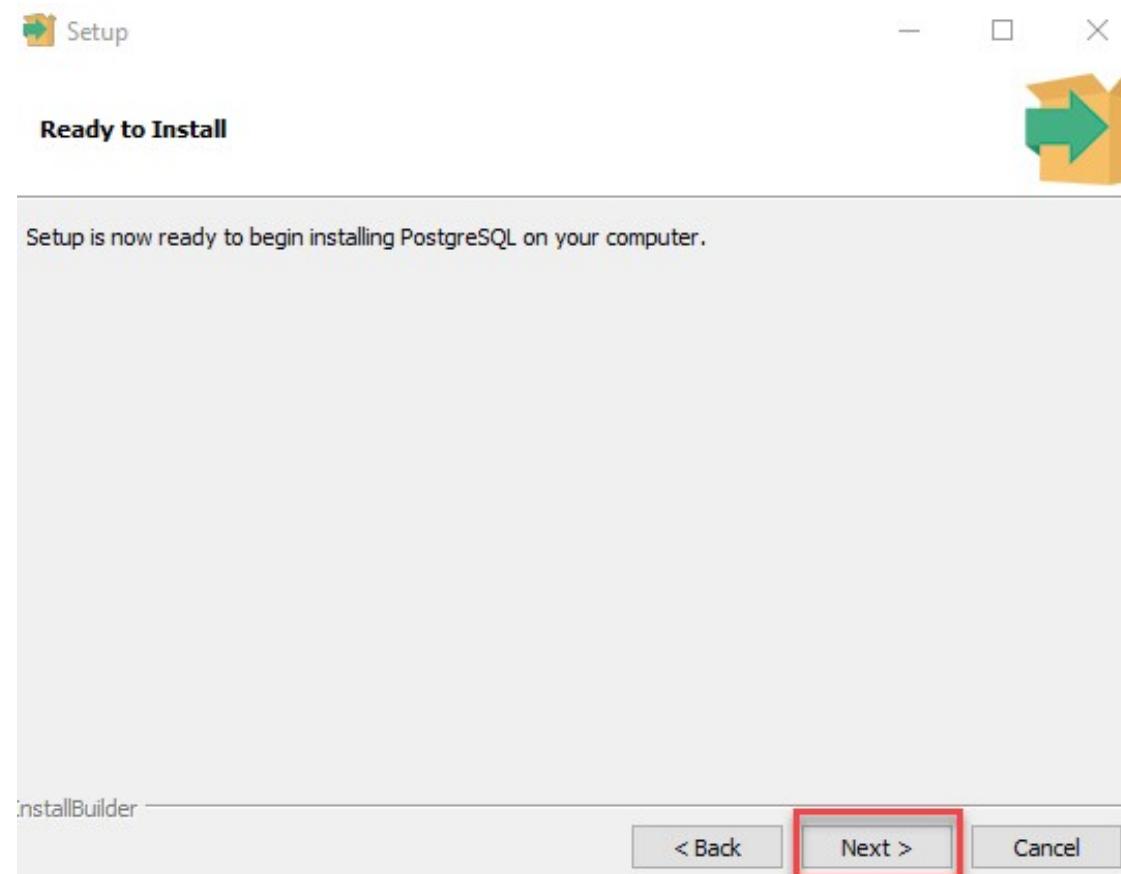
PostgreSQL安装步骤6 (Win)

- 这个界面最好截图或者拍一下



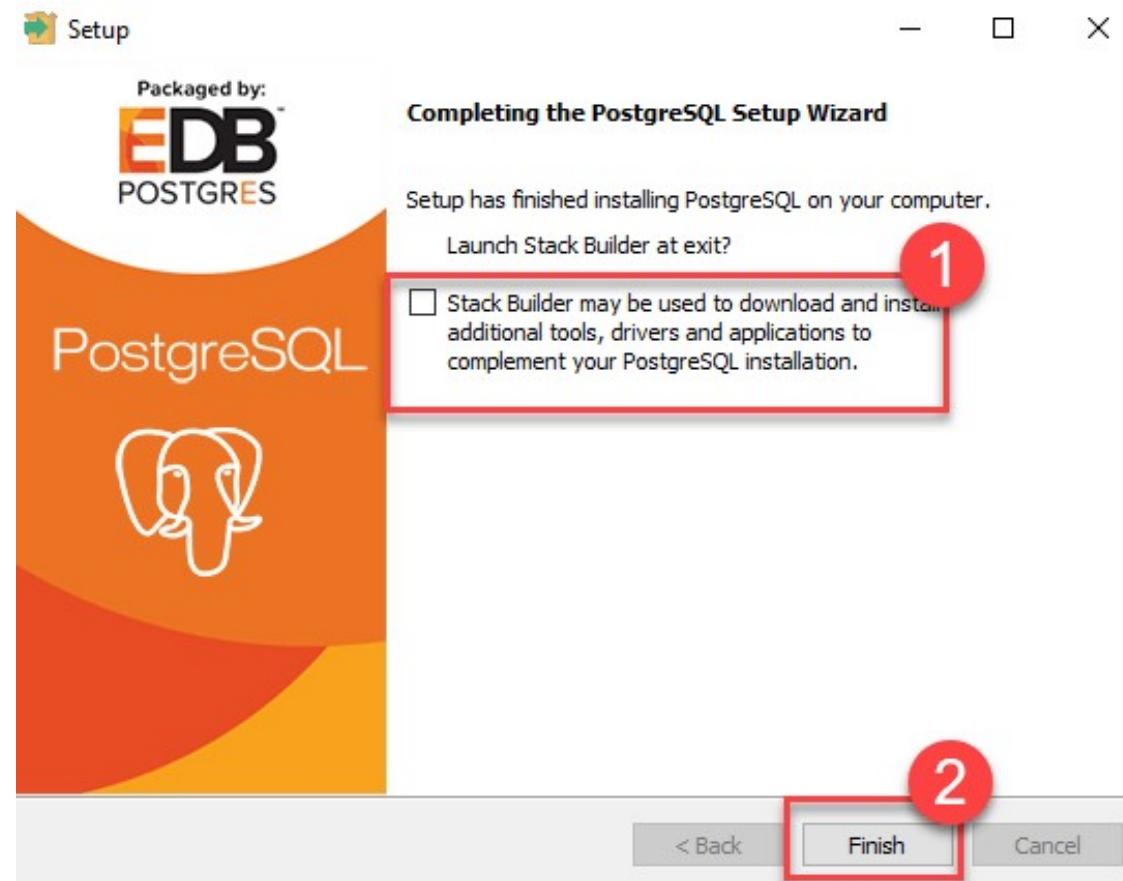
PostgreSQL安装步骤7 (Win)

- 点Next，开始安装



PostgreSQL安装步骤8 (Win)

- 安装完毕，这里去掉勾选



请打开下载好的SQL shell (psql)

- 本节课使用postgreSQL的SQL shell进行sql语言的学习



打开psql之后

- 进入到命令行窗口

```
Server [localhost]:          ↪回车  
Database [postgres]:        ↪回车  
Port [5432]:                ↪回车  
Username [postgres]:         ↪回车  
[Password for user postgres: ↪回车  
  psql (12.4)  
  Type "help" for help.  
  
postgres=#  
↑  
在这里写SQL代码
```

输入密码 (123456) (密码可能不会显示), 回车

win和mac显示稍有不同，但操作一样

psql的一些用法命令（之后会用到）

- 切换数据库： `\c dbname`
- 查看当前系统中有哪些数据库，可以使用\l或\list
- 查看当前数据库中所有的表，使用\d
- 查看指定表中的字段，使用\d + table_name： `\d mytable`
- 退出登录，使用\q
- 注意：反斜杠在一些电脑中会显示为¥

Q & A

You Have
Questions
We Have
Answers



1

数据库基本概念 データベース概要

2

PostgreSQL数据库安装 PostgreSQLのインストール

3

SQL语言基础 SQL言語の基本

SQL 简介

- SQL是用于访问和操作数据库的标准语言。
- SQL于1986年成为美国国家标准协会（ANSI）和1987年国际标准化组织（ISO）的标准
- SQL主要功能：
 - 对数据库执行查询
 - 从数据库检索数据
 - 在数据库中插入记录
 - 更新数据库中的记录
 - 从数据库中删除记录
 - 创建新的数据库
 - 在数据库中创建新表
 - 在数据库中创建存储过程
 - 在数据库中创建视图
 - 对表，过程和视图设置权限

SQL 简介

- 本节课我们讲学习如何用SQL语言访问和操作PostgreSQL数据库管理系统[データベース管理システム]。
- 尽管SQL是ANSI / ISO标准，但是不同的数据库管理系统[データベース管理システム]所使用的SQL语言的语法稍有不同。
- 但为了符合ANSI标准，它们都至少支持一些主要命令（例如SELECT, UPDATE, DELETE, INSERT, WHERE）。



Database Tables

- 数据库通常包含一个或多个表。每个表均由名称标识（例如“客户”或“订单”）。
- 表包含具有数据的记录（行）。

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

- 上表包含五个记录（每个客户一个）和七个列（CustomerID, CustomerName, ContactName, Address, City, PostalCode和Country）。

一些最重要的SQL命令

- CREATE DATABASE - 创建一个新的数据库
- CREATE TABLE - 创建一个新表
- SELECT - 从数据库中提取数据
- UPDATE - 更新数据库中的数据
- DELETE - 从数据库中删除数据
- INSERT INTO - 将新数据插入数据库
- ALTER DATABASE - 修改数据库
- ALTER TABLE - 修改表格
- DROP TABLE - 删除表
- CREATE INDEX - 创建索引 (搜索关键字)
- DROP INDEX - 删除索引
- 注意：
- SQL关键字不区分大小写：select与SELECT相同
- 每个SQL语句的末尾都需要使用分号；。

SQL CREATE DATABASE语句

- CREATE DATABASE语句用于创建新的SQL数据库。
- 句法：

```
CREATE DATABASE databasename;
```

- 以下SQL语句创建一个名为“MYDB”的数据库：

```
[postgres=# CREATE DATABASE MYDB;  
CREATE DATABASE
```

- 反斜杠\ + MYDB 进入此数据库

SQL DROP DATABASE语句

- DROP DATABASE语句用于删除现有的SQL数据库。
- 以下SQL语句删除现有数据库“testDB”：

```
DROP DATABASE testDB;
```

- 注意：删除数据库之前请小心。删除数据库将导致丢失存储在数据库中的完整信息！

SQL CREATE TABLE语句

- CREATE TABLE语句用于在数据库中创建新表。
- 句法：

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
    ....
);
```

- column1 指定表的列名称。
- datatype 指定列保存的数据类型（例如char, varchar, integer等）。
- [postgreSQL完整数据类型](#)

SQL CREATE TABLE语句

- 下面的示例创建一个名为“Persons”的表，该表包含五列：PersonID, LastName, FirstName, Address和City：

```
postgres=# CREATE TABLE Persons (
postgres(#     PersonID int,
postgres(#     LastName varchar(255),
postgres(#     FirstName varchar(255),
postgres(#     Address varchar(255),
postgres(#     City varchar(255)
[postgres(# );
CREATE TABLE
```

- PersonID列的类型为int，容纳一个整数。
- LastName, FirstName, Address和City列的类型为varchar，表示字段，并且这些字段的最大长度为255个字符。

查看表格信息

- \d 可以查看已存在的表信息（注意要用反斜杠）

```
[postgres=# \d
          List of relations
 Schema |   Name    | Type  | Owner
-----+-----+-----+
 public | persons | table | postgres
(1 row)
```

- \d + 表名查看表具体信息

```
[postgres=# \d persons
              Table "public.persons"
 Column |           Type           | Collation | Nullable | Default
-----+-----+-----+-----+-----+
 personid | integer
 lastname  | character varying(255)
 firstname | character varying(255)
 address   | character varying(255)
 city      | character varying(255)
```

SQL DROP TABLE语句

- DROP TABLE语句用于删除数据库中的现有表。
- 以下SQL语句删除现有表“ Shippers”：

```
DROP TABLE Shippers;
```

- TRUNCATE TABLE语句用于删除表中的数据，但不删除表本身。

```
TRUNCATE TABLE table_name;
```

SQL 约束[制約] (constraint)

- SQL约束[制約]用于指定表中数据的规则。
- 以在使用CREATE TABLE语句创建表或使用ALTER TABLE语句创建表后指定约束[制約]。
- 句法：

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
    ...
);
```

- 示例：

```
postgres=# CREATE TABLE COMPANY(
postgres(#     ID INT PRIMARY KEY      NOT NULL,
postgres(#     NAME        TEXT    NOT NULL,
postgres(#     AGE         INT     NOT NULL,
postgres(#     ADDRESS     CHAR(50),
postgres(#     SALARY      REAL
postgres(# );
CREATE TABLE
```

```
postgres=# \d company
Table "public.company"
 Column | Type | Collation | Nullable | Default
-----+-----+-----+-----+-----+
 id    | integer |          | not null |
 name  | text   |          | not null |
 age   | integer |          | not null |
 address | character(50) |          |
 salary | real   |          |
Indexes:
"company_pkey" PRIMARY KEY, btree (id)
```

SQL 约束[制約] (constraint)

- 约束[制約]用于限制可以进入表的数据类型。这样可以确保表格中数据的准确性和可靠性。如果约束[制約]和数据操作之间存在任何冲突，则将中止。
- 约束[制約]可以是列级别或表级别。列级约束[制約]适用于列，而表级约束[制約]适用于整个表。
- SQL中通常使用以下约束[制約]：
 - NOT NULL -确保列不能具有NULL值
 - UNIQUE -确保列中的所有值都不同
 - PRIMARY KEY -NOT NULL和UNIQUE的组合。唯一标识表中的每一行
 - FOREIGN KEY -唯一标识另一个表中的行/记录
 - CHECK -确保列中的所有值都满足特定条件
 - DEFAULT -在未指定值的情况下为列设置默认值
 - INDEX -用于非常快速地从数据库创建和检索数据

SQL ALTER TABLE语句

- ALTER TABLE语句用于添加，删除或修改现有表中的列。
- ALTER TABLE语句还用于在现有表上添加和删除各种约束[制約]。
- 语法：
 - 在一张已存在的表上添加列

```
ALTER TABLE table_name ADD column_name datatype;
```

- 在一张已存在的表上 DROP COLUMN (删除列)

```
ALTER TABLE table_name DROP COLUMN column_name;
```

- 修改表中某列的 DATA TYPE (数据类型)

```
ALTER TABLE table_name ALTER COLUMN column_name TYPE datatype;
```

- 给表中某列添加 NOT NULL 约束[制約]

```
ALTER TABLE table_name ALTER COLUMN column_name SET not null;
```

SQL ALTER TABLE语句

- 给表 ADD PRIMARY KEY (添加主键[主キー])

```
ALTER TABLE table_name  
ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1, column2...);
```

- DROP CONSTRAINT (删除约束[制約])

```
ALTER TABLE table_name ALTER COLUMN column_name DROP not null;
```

- DROP PRIMARY KEY (删除主键[主キー])

```
ALTER TABLE table_name  
DROP CONSTRAINT MyPrimaryKey;
```

练习时间

- 在之前创建好的Persons表中进行练习
- 为personid列增加主键[主キー]和not null约束[制約]
- 为lastname和firstname添加not null约束[制約]
- 删除lastname的not null约束[制約]

Q & A

You Have Questions
We Have Answers

pgAdmin4

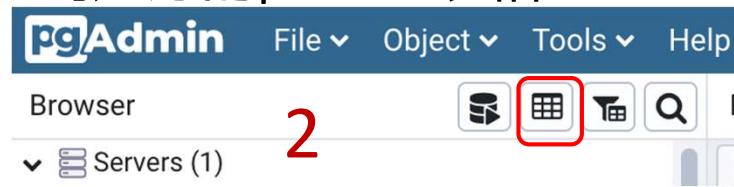
- 点击pgAdmin4， 浏览器会弹出可视化界面，并要求输入密码（123456）



- pgAdmin4是数据库的用户可视化操作界面，也就是说，可以取代psql里面用代码的形式对数据库进行访问与更改。
- 我们在psql用代码对数据库进行操作，然后刷新pgAdmin4的页面，结果则会显示在可视化页面上。

pgAdmin4

- 页面左侧点击我们自己创建好的persons表格
- 然后点击上面图标



- 页面的右边会出现下图所示表格

	personid	lastname	firstname	address
1	0	Tony	Stark	NewYork
2	0	Tony	Stark	NewYork
3	0	Tony	Stark	[null]
4	0	Tony	Stark	[null]
5	1	Bruce	Banner	[null]

1

Browser

Servers (1)

PostgreSQL 12

Databases (1)

postgres

Casts

Catalogs

Event Triggers

Extensions

Foreign Data Wrappers

Languages

Schemas (1)

public

Collations

Domains

FTS Configurations

FTS Dictionaries

FTS Parsers

FTS Templates

Foreign Tables

Functions

Materialized Views

Procedures

Sequences

Tables (2)

company

persons

Trigger Functions

Types

SQL INSERT INTO语句

- INSERT INTO语句用于在表中插入新记录。
- 可以通过两种方式编写INSERT INTO语句。
- 第一种方法同时指定列名和要插入的值：

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

- 示例：

```
postgres=# insert into persons (personid, lastname,firstname)
values (0, 'Tony','Stark'),(1,'Bruce','Banner')
;
INSERT 0 2
```

- 逗号隔开小括号可以实现插入多行
- 字符串要使用单引号

SQL INSERT INTO语句

- 如果要为表的所有列添加值，则无需在SQL查询中指定列名。但是要确保值的顺序与表中各列的顺序相同：

```
INSERT INTO table_name  
VALUES (value1, value2, value3, ...);
```

- 示例：

```
[postgres=# insert into persons  
values (0, 'Tony', 'Stark', 'NewYork')  
;  
INSERT 0 1
```

练习时间

- 将下列信息插入Persons表格
 - (2, 'Peter', 'Parker', 'Manhattan Queen Block')
 - (3, 'Harry', 'Pottor', 'London')
 - (4, 'Barry', 'Allen', 'Star City')

SQL SELECT语句

- SELECT语句用于从数据库中选择数据。
- 返回的数据存储在结果表中，称为结果集。
- SELECT语法

```
SELECT column1, column2, ...
FROM table_name;
```

- 上面代码译为从 (From) table_name里选择column1,2....。
- 如果要选择表中所有可用字段，请使用以下语法：
- 示例：

```
SELECT * FROM table_name;
```

```
postgres=# select personid, lastname
from persons
;
+-----+-----+
| personid | lastname |
+-----+-----+
|       0 |    Tony |
|       1 |   Bruce |
|       2 |    Tony |
+-----+-----+
(6 rows)
```

SQL SELECT DISTINCT语句

- 在表内部，一列通常包含许多重复值。
- SELECT DISTINCT语句仅用于返回不同的（不同的）值：

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

- 示例：personid不重复

```
[postgres=# select distinct personid, lastname from persons;
 personid | lastname
-----+-----
      2 | Tony
      1 | Bruce
      0 | Tony
(3 rows)
```

SQL WHERE子句

- WHERE子句用于选择满足指定条件的记录：

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

- 注意： WHERE子句不仅在SELECT语句中使用，还在UPDATE， DELETE语句等中使用
- 示例：

```
postgres=# select personid, lastname
[postgres-# from persons
[postgres-# where lastname='Tony';
 personid | lastname
-----+-----
      0 | Tony
      0 | Tony
      0 | Tony
      0 | Tony
      2 | Tony
(5 rows)
```

SQL 比较运算符

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. Note: In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

SQL 与或非 逻辑运算符

- WHERE子句可以与AND, OR和NOT运算符结合使用。
- AND语法：

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

- OR语法：

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

- NOT语法：

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

SQL 与或非示例

- 获得personid是0并且address是NewYork的记录：

```
[postgres=# select personid, lastname, firstname, address
[postgres-# from persons
[postgres-# where personid=0 and address='NewYork';
      personid | lastname | firstname | address
-----+-----+-----+-----
          0 | Tony     | Stark    | NewYork
          0 | Tony     | Stark    | NewYork
(2 rows)
```

SQL ORDER BY关键字

- ORDER BY关键字用于按升序或降序对结果集进行排序。
- 默认情况下， ORDER BY关键字对记录进行升序排序。要按降序对记录进行排序，请使用DESC关键字：

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

- 示例：从persons中选择所有记录，并按personid进行降序：

```
[postgres=# select * from persons order by personid desc;
 personid | lastname | firstname | address
 -----+-----+-----+-----
      2 | Tony     |           |
      1 | Bruce   | Banner    |
      0 | Tony     | Stark     | NewYork
      0 | Tony     | Stark     | NewYork
      0 | Tony     | Stark     |
      0 | Tony     | Stark     |
(6 rows)
```

SQL NULL值

- NULL值不同于零值或包含空格的字段。具有NULL值的字段是在记录创建过程中留为空白的字段
- 无法使用比较运算符（例如=, <或>）测试NULL值。
- 只能使用IS NULL和IS NOT NULL运算符。

```
SELECT column_names  
FROM table_name  
WHERE column_name IS NULL;
```

```
SELECT column_names  
FROM table_name  
WHERE column_name IS NOT NULL;
```

SQL UPDATE语句

- UPDATE语句用于修改表中的现有记录：

```
UPDATE table_name  
SET column1 = value1, column2 = value2, ...  
WHERE condition;
```

- 注意： UPDATE语句中的WHERE指定需要更新的记录。如果省略WHERE子句， 表中的所有记录都被更新
- 示例：

```
[postgres=# update persons  
[postgres-# set personid=3, lastname='John', firstname='Wick'  
[postgres-# where lastname='Bruce'  
[postgres-# ;  
UPDATE 1  
[postgres=# select * from persons  
[postgres-# ;  
 personid | lastname | firstname | address  
-----+-----+-----+-----  
       0 | Tony     | Stark    | NewYork  
       0 | Tony     | Stark    | NewYork  
       0 | Tony     | Stark    |  
       0 | Tony     | Stark    |  
       2 | Tony     | Stark    |  
       3 | John     | Wick    |  
(6 rows)
```

SQL DELETE语句

- DELETE语句用于删除表中的现有记录：

```
DELETE FROM table_name WHERE condition;
```

- 可以删除表中的所有行而不删除表。这意味着表结构，属性和索引将保持不变：

```
DELETE FROM table_name;
```

SQL常用函数

- MIN () , MAX () 函数返回所选列的最小/大值。

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

- COUNT () 函数返回与指定条件匹配的行的数量。

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

- AVG () 函数返回数字列的平均值。

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

- SUM () 函数返回数字列的总和。

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

SQL LIKE运算子

- 在WHERE子句中使用LIKE运算符在列中搜索指定的pattern：

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

- 通常将两个通配符与LIKE运算符结合使用（还有很多其他通配符）：

- % 百分号代表零个，一个或多个字符
- _ 下划线表示单个字符

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%oo'	Finds any values that start with "a" and ends with "o"

SQL IN和BETWEEN运算符

- IN运算符在WHERE子句中指定多个值。
- IN运算符是多个OR条件的简写。

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

- 或者这样写：

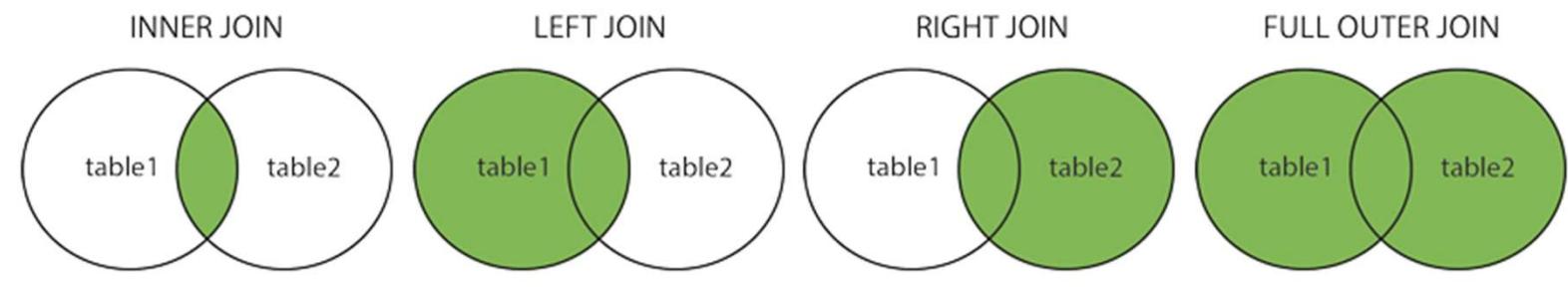
```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

- BETWEEN运算符选择给定范围内的值。值可以是数字，文本或日期。

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

不同类型的SQL JOIN (连接)

- JOIN子句用于根据两个或多个表之间的相关列来组合它们。
- SQL中JOIN的五种类型：
 - (INNER) JOIN[内部結合]：返回两个表中都具有的匹配值记录
 - LEFT (OUTER) JOIN[左外部結合]：从左表返回所有记录，并从右表返回匹配的记录
 - RIGHT (OUTER) JOIN[右外部結合]：从右表返回所有记录，并从左表返回匹配的记录
 - FULL (OUTER) JOIN[完全外部結合]：当左表或右表中存在匹配项时，返回所有记录
 - CROSS JOIN[交差結合]：交叉连接 (笛卡尔积)



SQL连接

- 接下来我们创建两张表 COMPANY 和 DEPARTMENT。
- 打开代码文件[company.sql](#)
- 将文件内容全部复制到psql里， 创建一个新的company表
- 内容如下：

	<code>id</code>	<code>name</code>	<code>age</code>	<code>address</code>	<code>salary</code>
1	1	Paul	32	California	20000
2	2	Allen	25	Texas	15000
3	3	Teddy	23	Norway	20000
4	4	Mark	25	Rich-Mond	65000
5	5	David	27	Texas	85000
6	6	Kim	22	South-Hall	45000
7	7	James	24	Houston	10000
8	8	Paul	24	Houston	20000
9	9	James	44	Norway	5000
10	10	James	45	Texas	5000

SQL连接

- 打开代码文件department.sql
- 将内容复制粘贴到psql， 创建department表
- 内容如下：

id	dept	emp_id
1	IT Billing	1
2	Engineering	2
3	Finance	7

交叉连接[交差結合] (CROSS JOIN)

- 笛卡尔积的SQL实现
- CROSS JOIN 的基础语法：

```
SELECT ... FROM table1 CROSS JOIN table2 ...
```

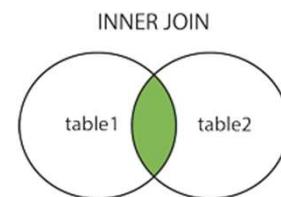
- 示例：
`SELECT EMP_ID, NAME, DEPT FROM COMPANY CROSS JOIN DEPARTMENT;`
- 得到的结果集特别大，共三十行

emp_id	name	dept
1	Paul	IT Billing
1	Allen	IT Billing
1	Teddy	IT Billing
1	Mark	IT Billing
1	David	IT Billing
1	Kim	IT Billing
1	James	IT Billing
1	Paul	IT Billing
1	James	IT Billing
1	James	IT Billing
2	Paul	Engineering
2	Allen	Engineering
2	Teddy	Engineering
2	Mark	Engineering
2	David	Engineering
2	Kim	Engineering
2	James	Engineering
2	Paul	Engineering
2	James	Engineering
2	James	Engineering
7	Paul	Finance
7	Allen	Finance
7	Teddy	Finance
7	Mark	Finance
7	David	Finance
7	Kim	Finance
7	James	Finance
7	Paul	Finance
7	James	Finance
7	James	Finance

内连接 (INNER JOIN)

- 内连接 (INNER JOIN) 的语法:

```
SELECT table1.column1, table2.column2...
FROM table1
INNER JOIN table2
ON table1.common_field = table2.common_field;
```

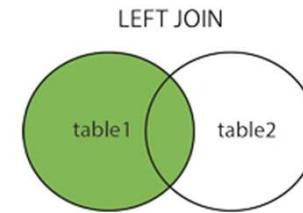


- 示例:

```
postgres=# SELECT EMP_ID, NAME, DEPT FROM COMPANY INNER JOIN DEPARTMENT ON COMPANY.ID = DEPARTMENT.EMP_ID;
   emp_id | name      | dept
-----+-----+
        1 | Paul      | IT Billing
        2 | Allen     | Engineering
        7 | James     | Finance
(3 rows)
```

左外连接（LEFT OUTER JOIN）

- 左外连接（LEFT OUTER JOIN）的基础语法：

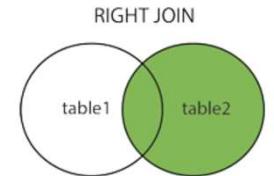


```
SELECT ... FROM table1 LEFT OUTER JOIN table2 ON conditional_expression ...
```

- 示例：

```
postgres=# SELECT EMP_ID, NAME, DEPT FROM COMPANY LEFT OUTER JOIN DEPARTMENT ON COMPANY.ID = DEPARTMENT.EMP_ID;
   emp_id | name      | dept
-----+-----+
        1 | Paul      | IT Billing
        2 | Allen     | Engineering
        7 | James     | Finance
       | James    |
       | David    |
       | Paul     |
       | Kim      |
       | Mark     |
       | Teddy    |
       | James    |
(10 rows)
```

右外连接（RIGHT OUT JOIN）



- 右外连接（RIGHT OUT JOIN）的基本语法：

```
SELECT ... FROM table1 RIGHT OUTER JOIN table2 ON conditional_expression ...
```

- 示例：

```
postgres=# SELECT EMP_ID, NAME, DEPT FROM COMPANY RIGHT OUTER JOIN DEPARTMENT ON COMPANY.ID = DEPARTMENT.EMP_ID;
   emp_id | name    | dept
-----+-----+
      1 | Paul    | IT Billing
      2 | Allen   | Engineering
      7 | James   | Finance
(3 rows)
```

外连接 (FULL OUTER JOIN)

- 外连接的基本语法：

```
SELECT ... FROM table1 FULL OUTER JOIN table2 ON conditional_expression ...
```

- 示例：

```
postgres=# SELECT EMP_ID, NAME, DEPT FROM COMPANY FULL OUTER JOIN DEPARTMENT ON COMPANY.ID = DEPARTMENT.EMP_ID;
          emp_id | name      | dept
-----+-----+-----+
          1 | Paul     | IT Billing
          2 | Allen    | Engineering
          7 | James    | Finance
          | James    |
          | David    |
          | Paul     |
          | Kim      |
          | Mark     |
          | Teddy    |
          | James    |
(10 rows)
```

SQL UNIONS

- UNION 操作符用于合并两个或多个 SELECT 语句的结果集。
- 请注意，UNION 内部的每个 SELECT 语句必须拥有相同数量的列。列也必须拥有相似的数据类型。同时，每个 SELECT 语句中的列的顺序必须相同。
- UNIONS 基础语法如下：

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]

UNION

SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

SQL UNIONS 示例

```
postgres=# SELECT EMP_ID, NAME, DEPT FROM COMPANY INNER JOIN DEPARTMENT
postgres-#      ON COMPANY.ID = DEPARTMENT.EMP_ID
postgres-#      UNION
postgres-# SELECT EMP_ID, NAME, DEPT FROM COMPANY LEFT OUTER JOIN DEPARTMENT
[postgres-#      ON COMPANY.ID = DEPARTMENT.EMP_ID;
   emp_id | name    |          dept
-----+-----+
       7 | James   | Finance
       2 | Allen   | Engineering
       | David   |
       | James   |
       | Kim     |
       1 | Paul    | IT Billing
       | Teddy   |
       | Paul    |
(9 rows)
```

SQL GROUP BY语句

- GROUP BY语句将具有相同值的行分组为摘要行，例如“查找每个国家/地区的客户数量”。
- GROUP BY语句通常与聚合函数 (COUNT, MAX, MIN, SUM, AVG) 一起使用，以按一个或多个列对结果集进行分组。

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

SQL GROUP BY语句示例

- 根据 NAME 字段值进行分组，找出每个客户的工资总额，并按工资总额排序：

```
[postgres=# SELECT NAME, SUM(SALARY) FROM COMPANY GROUP BY NAME ORDER BY SUM;
   name  |    sum
-----+-----
 Allen  | 15000
 James  | 20000
 Teddy  | 20000
 Paul   | 40000
 Kim    | 45000
 Mark   | 65000
 David  | 85000
(7 rows)
```

SQL HAVING 子句

- HAVING 子句可以让我们筛选分组后的各组数据。
- WHERE 子句在所选列上设置条件，而 HAVING 子句则在由 GROUP BY 子句创建的分组上设置条件。
- HAVING 子句必须放置于 GROUP BY 子句后面， ORDER BY 子句前面，下面是 HAVING 子句在 SELECT 语句中基础语法：

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

SQL HAVING 子句 示例

- 找出根据 name 字段值进行分组，并且名称的计数大于 1 数据：

```
[postgres=# SELECT NAME FROM COMPANY GROUP BY name HAVING count(name) > 1;
      name
-----
    Paul
    James
(2 rows)
```

Q & A

You Have Questions
We Have Answers

THANK YOU