|  |  |
| --- | --- |
| （标题）**Querying Data**  SELECT 的使用 | |
| （概要）   1. 通常SELECT与FROM配合使用，可以指定SELECT的字段 2. “\*”代表所有字段 3. SELECT多个字段 4. SELECT等关键字通常使用大写，如果表名或者字段名中包含大写，则需要将通过双引号包起来 5. SELECT拼接内容 6. 给拼接内容赋名 7. 计算数值 8. 查看pg版本 | （详情） |
| SELECT first\_name FROM customer; |
| SELECT \* FROM customer; |
| SELECT first\_name, last\_name FROM customer; |
| SELECT “FIRST\_NAME” FROM customer; |
| SELECT first\_name || ‘ ’ || last\_name FROM customer; |
| SELECT first\_name || ‘ ‘ || last\_name AS full\_name FROM customer; |
| SELECT 5 \* 3 AS result |
| SELECT version(); |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| （总结）   1. SELECT与FROM配合使用 2. SELECT的对象如果复杂，可以赋“别名” | |

|  |  |
| --- | --- |
| （标题）**Querying Data**  ORDER BY的使用 | |
| （概要）   1. ORDER BY 默认是升序 2. ASC：升序 3. DESC：降序 4. 多个排序是，排序对象逗号分隔 5. 通过expression排序 | （详情） |
| SELECT first\_name FROM customer ORDER BY first\_name |
| SELECT first\_name FROM customer ORDER BY first\_name ASC |
| SELECT first\_name FROM customer ORDER BY first\_name DESC |
| SELECT first\_name, last\_name FROM customer ORDER BY first\_name ASC, last\_name DESC |
| SELECT first\_name LENGTH（first\_name） len FROM customer ORDER BY first\_name |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| （总结）  ORDER BY配合SELECT一起使用，ORDER BY的对象是字段，方法有“升序”和“降序” | |

|  |  |
| --- | --- |
| （标题）**Querying Data**  SELECT DISTINCT的使用 | |
| （概要）   1. 指定一个字段 2. 指定两个字段，联合唯一 3. 指定一个字段并SELECT关联字段 4. 指定两个字段并SELECT关联字段 5. 与ORDER BY配合使用 | （详情）去除重复数据，保证被SELECT字段唯一性 |
| SELECT DISTINCT first\_name FROM customer; |
| SELECT DISTINCT length, rating FROM film; |
| SELECT DISTINCT ON (rating) title, rating, length FROM film; |
| SELECT DISTINCT ON (rating, length) title, rating, length FROM film; |
| SELECT DISTINCT ON (rating) length, rating FROM film ORDER BY rating; |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| （总结）  1、DISTINCT获取的“第一条”数据是随机的，通常配合ORDER BY获取需要的第一条数据 | |

|  |  |  |  |
| --- | --- | --- | --- |
| （标题）**Filtering Data**  WHERE的使用 | | | |
| （概要）   1. WHERE与“条件”配合使用，“过滤”返回的数据 2. WHERE配合AND 3. WHERE配合OR 4. WHERE配合IN 5. WHERE配合LIKE 6. WHERE配合BETWEEN 7. WHERE配合<>使用 | （详情） | | |
| SELECT select\_list FROM table\_name WHERE condition; | | |
| SELECT first\_name FROM "customer" WHERE first\_name = 'Jamie' **AND** last\_name = 'Rice'; | | |
| SELECT first\_name FROM "customer" WHERE first\_name = 'Jamie' **OR** last\_name = 'Rice'; | | |
| SELECT first\_name, last\_name FROM customer WHERE first\_name **IN** ('Ann','Anne','Annie'); | | |
| SELECT first\_name, last\_name FROM customer WHERE first\_name **LIKE** 'An%'; | | |
| SELECT first\_name, "length"(first\_name) name\_length FROM customer WHERE "length"(first\_name) **BETWEEN** 3 AND 5; | | |
| SELECT first\_name, last\_name FROM customer WHERE first\_name LIKE'Bra%' AND last\_name **<>** 'Motley' ; | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
| （总结）  WHERE就是过滤条件，用来对指定的数据做处理操作。 | 常用“操作符” | = | Equal |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal |
| <= | Less than or equal |
| <> or != | Not equal |
| AND | Logical operator AND |
| OR | Logical operator OR |

|  |  |
| --- | --- |
| （标题）**Filtering Data**  LIMIT的使用  可与OFFSET配合，与ORDER BY配合 | |
| （概要）   1. 与ORDER BY配合 2. 与OFFSET配合，跳过前几条数据，如果OFFSET的值比较大，那些数据同样会被计算（即使他们没有被返回） | （详情） |
| SELECT \* FROM film ORDER BY film\_id LIMIT 5; |
| SELECT \* FROM film ORDER BY film\_id LIMIT 5 OFFSET 3; |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| （总结）  LIMIT常用于数据内容测试，检查数据表格中是否有数据。  与ORDER BY配合使用，帮助用户获取指定顺序的数据  与OFFSET配合，跳过指定数据的数据。 | |

|  |  |
| --- | --- |
| （标题）**Filtering Data**  FETCH的使用 | |
| （概要）   1. 获取第一行数据 2. 获取第一行数据 3. 获取前五行数据 4. 跳过5行获取前五行 | （详情） |
| SELECT \* FROM film ORDER BY title FETCH FIRST ROW ONLY; |
| SELECT \* FROM film ORDER BY title FETCH FIRST 1 ROW ONLY; |
| SELECT \* FROM film ORDER BY title FETCH FIRST 5 ROW ONLY; |
| SELECT \* FROM film ORDER BY title OFFSET 5 ROWS FETCH FIRST 5 ROW ONLY; |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| （总结）   1. FETCH中 OFFSET使用与LIMIT类似 2. 参数FIRST与NEXT可相互替换 3. ROW与ROWS可相互替换 | |

|  |  |
| --- | --- |
| （标题）**Filtering Data**  IN 的使用 | |
| （概要）   1. IN 配合列表 2. IN配合“subquery” 3. NOT IN | （详情） |
| value IN (value1,value2,...)  SELECT \* FROM "rental" WHERE customer\_id IN (1,2); |
| value IN (SELECT value FROM tbl\_name);  SELECT \* FROM "rental" WHERE CAST(return\_date AS DATE) = '2005-06-03'; |
| SELECT \* FROM "rental" WHERE customer\_id NOT IN (1,2); |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| （总结）   1. IN配合列表，等价于“equal (=) and OR operators”，但“IN”更快 2. NOT IN配合列表，等价于“not equal (<>) and the AND operators” 3. NOT IN配合“subquery”时候，如果“subquery”中包含“NULL”，则将出错或者速度很慢。 4. “CAST”是将一种datatype转化成另外一种。“CAST ( expression AS target\_type );” | |

|  |  |
| --- | --- |
| （标题）**Filtering Data**  BETWEEN 的使用 | |
| （概要）   1. 简单应用 2. 配合NOT使用 3. 配合日期使用 | （详情） |
| SELECT \* FROM "payment" WHERE amount BETWEEN 8 AND 9; |
| SELECT \* FROM "payment" WHERE amount NOT BETWEEN 8 AND 9; |
| SELECT \* FROM "payment" WHERE payment\_date BETWEEN '2007-02-07' AND '2007-02-15'; |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| （总结）  1、BETWEEN 常与数值、日期配合使用。 | |

|  |  |
| --- | --- |
| （标题）**Filtering Data**  LIKE 的使用 | |
| （概要）   1. 使用% 2. 使用举例 3. 与\_配合使用 4. 与NOT配合使用 5. 忽略大小写的匹配 | （详情） |
| SELECT \* FROM customer WHERE first\_name LIKE 'Jen%'; |
| SELECT 'foo' LIKE 'foo', 'foo' LIKE 'f%', 'foo' LIKE '\_o\_', 'foo' LIKE 'b\_' |
| SELECT \* FROM customer WHERE first\_name LIKE '\_her%'; |
| SELECT \* FROM customer WHERE first\_name NOT LIKE 'Jen%'; |
| SELECT \* FROM customer WHERE first\_name ILIKE 'BAR%'; |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| （总结）  Percent ( %) for matching any sequence of characters.  Underscore ( \_) for matching any single character.  ~~ is equivalent to LIKE  ~~\* is equivalent to ILIKE  !~~ is equivalent to NOT LIKE  !~~\* is equivalent to NOT ILIKE | |

|  |  |
| --- | --- |
| （标题）**Filtering Data**  IS NULL的使用 | |
| （概要）   1. IS NULL判断 2. IS NOT NULL判断 | （详情）  用来判断字段的值是否为空 |
| SELECT \* FROM "payment" WHERE amount IS NULL; |
| SELECT \* FROM "payment" WHERE amount IS NOT NULL; |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| （总结） | |

|  |  |
| --- | --- |
| （标题）**Filtering Data**  Alias的使用 | |
| （概要）   1. 给字段赋别名 2. 给expression赋别名 3. 给很长的tablename赋别名 | （详情） |
| SELECT first\_name AS fn FROM "customer"; |
| SELECT first\_name || ' ' || last\_name AS fn FROM "customer" ORDER BY fn; |
| SELECT t.column\_name FROM a\_very\_long\_table\_name t; |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| （总结）  别名可赋值给：字段名、字段名组合成expression、tablename | |

|  |  |
| --- | --- |
| （标题）**Joining Multiple Tables**  Join的使用 | |
| （概要）  1、SELECT      a.id id\_a,      a.fruit fruit\_a,      b.id id\_b,      b.fruit fruit\_b  FROM      basket\_a a  INNER JOIN basket\_b b ON a.fruit = b.fruit;  2、SELECT      a.id id\_a,      a.fruit fruit\_a,      b.id id\_b,      b.fruit fruit\_b  FROM      basket\_a a  LEFT JOIN basket\_b b ON a.fruit = b.fruit;  3、SELECT      a.id id\_a,      a.fruit fruit\_a,      b.id id\_b,      b.fruit fruit\_b  FROM      basket\_a a  RIGHT JOIN basket\_b b ON a.fruit = b.fruit; | （详情） |
| Inner join |
| Left join |
| Right join |
| Full join |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| （总结） | |

|  |  |
| --- | --- |
| （标题）**Joining Multiple Tables**  INNER JOIN的使用 | |
| （概要）  1、SELECT     customer.customer\_id,     first\_name,     last\_name,     email,     amount,     payment\_date  FROM     customer  INNER JOIN payment ON payment.customer\_id = customer.customer\_id  ORDER BY     customer.customer\_id;  2、SELECT     customer.customer\_id,     customer.first\_name customer\_first\_name,     customer.last\_name customer\_last\_name,     customer.email,     staff.first\_name staff\_first\_name,     staff.last\_name staff\_last\_name,     amount,     payment\_date  FROM     customer  INNER JOIN payment ON payment.customer\_id = customer.customer\_id  INNER JOIN staff ON payment.staff\_id = staff.staff\_id; | （详情） |
| 两个表的join |
| 三个表的join |
| 公共字段完全相同时：  ON payment.customer\_id = customer.customer\_id 可简写为：  USING (customer\_id) |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| （总结）  根据两个表的公共字段，每一行数据匹配到相应的结果。 | |
| （标题）**Joining Multiple Tables**  LEFT JOIN的使用 | |
| （概要）  1、SELECT     film.film\_id,     film.title,     inventory\_id  FROM     film  LEFT JOIN inventory ON inventory.film\_id = film.film\_id;  2、SELECT     film.film\_id,     film.title,     inventory\_id  FROM     film  LEFT JOIN inventory ON inventory.film\_id = film.film\_id  WHERE     inventory.film\_id IS NULL; | （详情） |
| 常用 |
| 找出“右表”没有数据的行 |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| （总结）   1. 没有匹配到的左表数据，直接赋值NULL 2. 如果第二种用法，使用的是“IS NOT NULL”，LEFT JOIN就等同INNER JOIN了 3. RIGHT JOIN与LEFT JOIN类似。 | |

|  |  |
| --- | --- |
| （标题）**Joining Multiple Tables**  self-join的使用 | |
| （概要）  1、SELECT  e.first\_name || ' ' || e.last\_name employee,  m .first\_name || ' ' || m .last\_name manager  FROM  employee e  INNER JOIN employee m ON m .employee\_id = e.manager\_id  ORDER BY  manager;  2、SELECT  e.first\_name || ' ' || e.last\_name employee,  m .first\_name || ' ' || m .last\_name manager  FROM  employee e  LEFT JOIN employee m ON m .employee\_id = e.manager\_id  ORDER BY  manager;  3、SELECT  f1.title,  f2.title,  f1. length  FROM  film f1  INNER JOIN film f2 ON f1.film\_id <> f2.film\_id  AND f1. length = f2. length; | （详情） |
| 两次调用employee表，根据manager\_id和employee\_id 字段查询员工的上下级关系 |
| 同1，将INNER JOIN修改为LEFT JOIN，为了显示最上级领导 |
| 同一个表的不同行，根据某个字段的属性关联到一起 |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| （总结）   1. SELF JOIN的用法1：一张表包含一个元素的多个字段（属性相关），自己与自己对比，用以显示上下级关系 2. 同一个表的不同行，通过某个字段的属性（比如：长度相等）关联 | |

|  |  |
| --- | --- |
| （标题）**Joining Multiple Tables**  FULL [OUTER] JOIN的使用  OUTER 是可选的参数 | |
| （概要）  1、SELECT     employee\_name,     department\_name  FROM     employees e  FULL OUTER JOIN  departments d ON d.department\_id = e.department\_id;  2、SELECT     employee\_name,     department\_name  FROM     employees e  FULL OUTER JOIN  departments d ON d.department\_id = e.department\_id  WHERE     employee\_name IS NULL;  3、SELECT     employee\_name,     department\_name  FROM     employees e  FULL OUTER JOIN departments d ON d.department\_id = e.department\_id  WHERE     department\_name IS NULL; | （详情） |
| 一般应用 |
| 查出没有员工的部门 |
| 查询没有部门的员工 |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| （总结）   1. OUTER JOIN能将两个表指定的字段匹配，包括：匹配上的（雷同INNER JOIN）和没有匹配上的（LEFT JOIN 和RIGHT JOIN），没有匹配上的赋NULL值 2. 配合WHERE条件，筛选出需要的数据。 | |

|  |  |
| --- | --- |
| （标题）**Joining Multiple Tables**  **CROSS JOIN**的使用 | |
| （概要）  1、SELECT \* FROM t1 CROSS JOIN t2;  2、SELECT \* FROM t1, t2;  3、  Image | （详情） |
| 写法1 |
| 写法2 |
| CROSS JOIN的产生流程 |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| （总结）  CROSS JOIN将生成一个“卡文迪许”的对应表，详细见上图 | |

|  |  |
| --- | --- |
| （标题）**Grouping Data**  **GROUP BY**的使用 | |
| （概要）  1、SELECT     customer\_id,     SUM (amount)  FROM     payment  GROUP BY     customer\_id;  2、SELECT     customer\_id,     SUM (amount)  FROM     payment  GROUP BY     customer\_id  ORDER BY     SUM (amount) DESC;  3、SELECT     DATE(payment\_date) paid\_date,     SUM(amount) sum  FROM     payment  GROUP BY     DATE(payment\_date); | （详情） |
| GROUP BY配合[aggregate function](https://www.postgresqltutorial.com/postgresql-aggregate-functions/)使用（SUM，COUNT） |
| 配合ORDER BY做排序 |
| 使用DATE()函数，给数据分组 |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| （总结）   1. GROUP BY把指定的字段分组，并可以配合[aggregate function](https://www.postgresqltutorial.com/postgresql-aggregate-functions/) 计算相应的结果，还可以与ORDER BY一起使用。 2. GROUP BY单独使用有去重的功能，与DISTINCT类似，但DISTINCT不能使用[aggregate function](https://www.postgresqltutorial.com/postgresql-aggregate-functions/) | |

|  |  |
| --- | --- |
| （标题）**Grouping Data**  **HAVING**的使用  在GROUP BY的基础上做筛选 | |
| （概要）  1、SELECT     customer\_id,     SUM (amount)  FROM     payment  GROUP BY     customer\_id  HAVING     SUM (amount) > 200; | （详情） |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| （总结）  1、HAVING在GROUP BY的基础上筛选，有点像WHERE，在配合使用时，通常是：  。。。WHERE condition GROUP BY field HAVING condition  2、在有ORDRE BY的情况下，使用情况如下：  。。。WHERE condition GROUP BY field HAVING condition ORDER BY field （上述的field可以是[aggregate function](https://www.postgresqltutorial.com/postgresql-aggregate-functions/) ） | |

|  |  |
| --- | --- |
| （标题）**Performing Set Operations**  **UNION**的使用 | |
| （概要）  1、SELECT \*  FROM     sales2007q1  UNION  SELECT \*  FROM     sales2007q2;  2、SELECT \*  FROM     sales2007q1  UNION ALL  SELECT \*  FROM     sales2007q2;  3、SELECT \*  FROM     sales2007q1  UNION ALL  SELECT \*  FROM     sales2007q2  ORDER BY  name ASC,  amount DESC; | （详情） |
| UNION一般应用，会去重 |
| UNION ALL，保留所有行 |
| 与ORDER BY配合使用（放在最后） |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| （总结）   1. UNION的两组字段，要求：每组字段数量一致，两组对应的字段包含“兼容”的数据类型 2. UNION合并的数据会去重，UNION ALL不会 3. UNION主要用于将同类数据合并，比如统计某类数据。 | |

|  |  |
| --- | --- |
| （标题）**Performing Set Operations**  **INTERSECT** 的使用 | |
| （概要）  1、SELECT     employee\_id  FROM     keys  INTERSECT  SELECT          employee\_id  FROM     hipos  ORDER BY employee\_id; | （详情） |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| （总结）  1、INTERSECT的应用结果有点类似INNER JOIN | |

|  |  |
| --- | --- |
| （标题）**Performing Set Operations**  **EXCEPT** 的使用 | |
| （概要）  1、SELECT  film\_id,  title  FROM  film  EXCEPT  SELECT  DISTINCT inventory.film\_id,  title  FROM  inventory  INNER JOIN film ON film.film\_id = inventory.film\_id  ORDER BY title;  2、 | （详情） |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| （总结）  1、 | |

|  |  |
| --- | --- |
| （标题）**Performing Set Operations**  **UNION**的使用 | |
| （概要） | （详情） |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| （总结）  1、 | |