SQL学习

Part-III-1

Preview简览

SQL queries
SQL查询语句

sqltutorial.org/sql-cheat-sheet

This Lecture
本节内容

- Set operators & nested queries
  - 集合算子和嵌套查询
- Aggregation & GROUP BY
  - 聚合函数：SUM，Count，MIN，MAX，AVG
- GROUP BY子句、Having子句

练习代码：SQL-2.sql, SQL-2.ipynb

集合算子

Set Operators

(复习) 集合代数 Set algebra

多集 （multiset），一个允许有重复的集合

连表 List: [1, 1, 2, 3]
集合 Set: {1, 2, 3}
多集 Multiset: {1, 1, 2, 3}

A **multiset** is an unordered list (or: a set with multiple duplicate instances allowed)

合并运算 UNIONs

Set: {1, 2, 3} U { 2 } = { 1, 2, 3 }
Multiset: {1, 1, 2, 3} U { 2 } = { 1, 1, 2, 2, 3 }

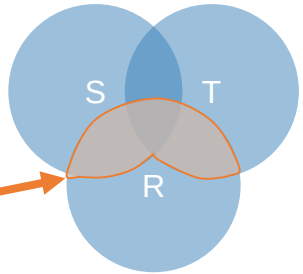
交叉积 Cross-product

{1, 1, 2, 3} * { y, z } =
{ <1, y>, <1, y>, <2, y>, <3, y>
<1, z>, <1, z>, <2, z>, <3, z>
}

i.e. no *next()*, etc. methods!
集合没有顺序，因此没有next()方法

An Unintuitive Query
一个非直观的查询

```
SELECT DISTINCT R.A
FROM   R, S, T
WHERE  R.A=S.A OR R.A=T.A
```



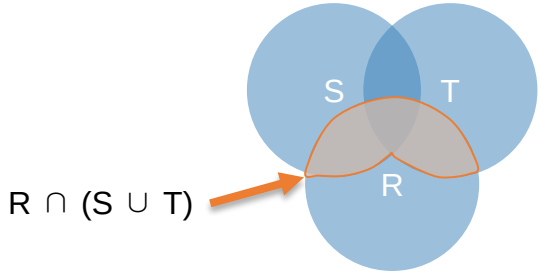
But what if S = ϕ?
但是如果S是空集，怎么办?

Computes R ∩ (S ∪ T)
计算 R ∩ (S ∪ T)

Go back to the semantics!
回头看一下语义

What does this look like in Python?
在Python中如何计算？

```
SELECT DISTINCT R.A
FROM   R, S, T
WHERE  R.A=S.A OR R.A=T.A
```



- 语义Semantics:

  1. Take cross-product
  进行交叉积运算

  1. Apply selections / conditions
  应用选择/条件过滤

  1. Apply projection
  应用投射

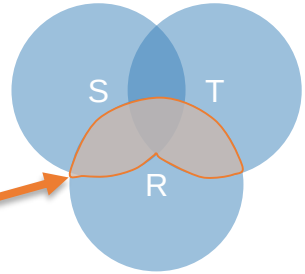
Joins / cross-products are just **nested for loops** (in simplest implementation)!

If-then statements!

What does this look like in Python?
在Python中如何计算？

```
SELECT DISTINCT R.A
FROM   R, S, T
WHERE  R.A=S.A OR R.A=T.A
```

R ∩ (S ∪ T)




```
output = {}

for r in R:
   for s in S:
      for t in T:
         if r['A'] == s['A'] or r['A'] == t['A']:
            output.add(r['A'])
return list(output)
```

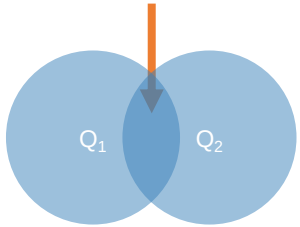
Can you see now what happens if S = []?

Explicit Set Operators: INTERSECT

显式集合运算 交集

```
SELECT R.A
FROM   R, S
WHERE  R.A=S.A
INTERSECT
SELECT R.A
FROM   R, T
WHERE  R.A=T.A
```

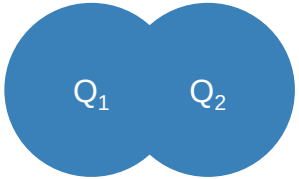
$$\{r.A \mid r.A = s.A\} \cap \{r.A \mid r.A = t.A\}$$



UNION 显式集合运算 并集

$$\{r.A \mid r.A = s.A\} \cup \{r.A \mid r.A = t.A\}$$

```
SELECT R.A
FROM   R, S
WHERE  R.A=S.A
UNION
SELECT R.A
FROM   R, T
WHERE  R.A=T.A
```



Why aren't there duplicates?
为什么没有重复值呢?

By default:
SQL retains set semantics for UNIONs, INTERSECTs!
缺省条件下,
SQL保留集合语义对于合集和交集

What if we want duplicates?
但是如果我们需要重复值呢?
用Union ALL

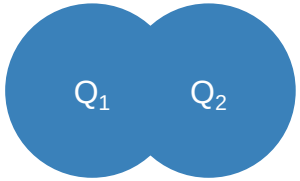
UNION ALL

```
SELECT  R.A
FROM    R, S
WHERE   R.A=S.A
UNION ALL
SELECT R.A
FROM    R, T
WHERE   R.A=T.A
```

ALL indicates Multiset operations

ALL关键词表明是多集运算

$$\{r.A \mid r.A = s.A\} \cup \{r.A \mid r.A = t.A\}$$



显式多集运算 多集并

嵌套查询

- Nested Query

Nested queries: Sub-queries Return Relations

嵌套查询 子查询返回关系

嵌套查询(Nested queries)指的是**子查询(Sub-queries)**是嵌套在较大查询中的 SQL 查询。

1. 子查询可以嵌套在 SELECT，INSERT，UPDATE 或 DELETE 语句内或另一个子查询中。
2. 子查询通常会在另一个 SELECT 语句的 WHERE 子句中添加。
3. 子查询必须被圆括号 () 括起来。

子查询也称为**内部查询(Inner query)**或内部选择，而包含子查询的语句也称为**外部查询(Outer query)**或外部选择。

1. 内部查询首先在其外部查询之前执行，以便可以将内部查询的结果传递给外部查询。
2. 比较运算符也可以是多行运算符，比较运算符，如 >，<，或 =。如 IN，ANY 或 ALL。

Nested queries: Sub-queries Return Relations
嵌套查询 子查询返回关系

Another example:
另一个示例：

Company(name, city)
Product(name, maker)
Purchase(id, product, buyer)

```
SELECT c.city
FROM   Company c
WHERE  c.name  IN (
              SELECT pr.maker
       FROM     Purchase p, Product pr
       WHERE  p.product = pr.name
                          AND p.buyer = 'Mickey')
```

“
- Companies making products bought by Mickey”
- Location of companies?
”

米奇购买的产品的生产商的公司位置

Nested Queries
嵌套查询

Are these queries equivalent?
这两个查询等价吗?

```
SELECT c.city
FROM   Company c
WHERE  c.name  IN (
SELECT pr.maker
FROM   Purchase p, Product pr
WHERE  p.name = pr.product
           AND p.buyer = 'Mickey')
```

```
SELECT c.city
FROM   Company c,
           Product pr,
           Purchase p
WHERE  c.name = pr.maker
    AND  pr.name = p.product
    AND  p.buyer = 'Mickey'
```

Beware of duplicates!
请注意重复的结果

Nested Queries
嵌套查询

```
SELECT DISTINCT c.city
FROM   Company c,
       Product pr,
       Purchase p
WHERE  c.name = pr.maker
  AND  pr.name = p.product
  AND  p.buyer = 'Mickey'
```

```
SELECT DISTINCT c.city
FROM   Company c
WHERE  c.name  IN (
  SELECT pr.maker
  FROM   Purchase p, Product pr
  WHERE  p.product = pr.name
               AND p.buyer = 'Mickey')
```

Now they are equivalent (both use set semantics)
现在两种写法相等了（都使用了集合语义）

Subqueries Return Relations

子查询返回关系

- You can also use operations of the form:
- 你可以使用以下的操作形式：
- s > ALL R
  - 大于R中的所有值
- s < ANY R
  - 至少小于R中的一个值
- EXISTS R
  - R中存在值

*ANY and ALL not supported by SQLite.

*SQLite不支持ANY和ALL操作

Find products that are more expensive than all those produced by “Gizmo-Works”

找到产品比“Gizmo-Works”生产的所有产品都要贵

Ex: 示例

Product(name, price, category, maker)

```
SELECT name
FROM   Product
WHERE  price > ALL(
    SELECT price
    FROM   Product
    WHERE  maker = ‘Gizmo-Works’)
```

Subqueries Returning Relations

子查询返回关系

Find ‘copycat’ products, i.e. products made by competitors with the same names as products made by “Gizmo-Works”

找到山寨“Gizmo-Works”公司的产品，即和“Gizmo-Works”生产的产品同名，但非“Gizmo-Works”生产的产品

Product(name, price, category, maker)

Ex:

```
SELECT p1.name
FROM   Product p1
WHERE  p1.maker = ‘Gizmo-Works’
  AND EXISTS(
        SELECT p2.name
        FROM   Product p2
       WHERE  p2.maker <> ‘Gizmo-Works’
                    AND p1.name = p2.name)
```

<> means !=

- You can also use operations of the form:
- 你可以使用以下的操作形式：
- s > ALL R
  - 大于R中的所有值
- s < ANY R
  - 至少小于R中的一个值
- EXISTS R
  - R中存在值

Nested queries as alternatives to INTERSECT and EXCEPT

嵌套查询是实现集合交集和差集的另外一种实现方法

```
(SELECT R.A, R.B
 FROM   R)
INTERSECT
(SELECT S.A, S.B
 FROM   S)
```

```
SELECT R.A, R.B
FROM   R
WHERE EXISTS(
                SELECT *
                FROM S
      WHERE R.A=S.A AND R.B=S.B)
```

```
(SELECT R.A, R.B
 FROM   R)
EXCEPT
(SELECT S.A, S.B
 FROM   S)
```

```
SELECT R.A, R.B
FROM   R
WHERE NOT EXISTS(
                SELECT *
                FROM S
      WHERE R.A=S.A AND R.B=S.B)
```

INTERSECT and EXCEPT not in some DBMSs!
在某些数据库中没有INTERSECT 和 EXCEPT

思考题：想一想，练一练

找出所有的电影，它的名称出现一次以上

Find movies whose title appears more than once.

Movie(title, year, director, length)

使用SFW语句如何完成查询?

Correlated Queries Using External Vars in Internal Subquery

相关查询：在内部子查询中使用外部变量

Find movies whose title appears more than once.
找出电影名称多于1次的电影（重拍过的电影）

Note the scoping of the variables!

注意变量的有效范围!

Movie(title, year, director, length)

```
SELECT DISTINCT title
FROM   Movie AS m
WHERE  year <> ANY(
                    SELECT  year
                    FROM    Movie
               WHERE title =  m.title)
```

Note also: this can still be expressed as single SFW query…

聚合与分组

- Aggregate, Group by &Having

What you will learn about in this section
本节学习内容

1. Aggregation operators 聚合函数算子
2. GROUP BY 分组
3. GROUP BY: with HAVING, semantics
   分组：Having子句，语义

聚合函数 Aggregation

```
SELECT AVG(price)
FROM   Product
WHERE  maker = “Toyota”
```

```
SELECT COUNT(*)
FROM   Product
WHERE  year > 1995
```

- SQL supports several **aggregation** operations:
  - SUM, COUNT, MIN, MAX, AVG

- SQL支持一些聚合操作，包括：
  - 求和SUM，总数计数Count，最小值MIN，最大值MAX和平均值AVG

Aggregation: COUNT
聚合：计数

- COUNT applies to duplicates, unless otherwise stated
- 计数适用于重复值，除非另外声明

```
SELECT COUNT(category)
FROM   Product
WHERE  year > 1995
```

We probably want:
我们可能需要：

```
SELECT COUNT(DISTINCT category)
FROM   Product
WHERE  year > 1995
```

销售统计的例子：（常用）
More Examples 更多示例

```
Purchase(product, date, price, quantity)
```

```
SELECT SUM(price * quantity)
FROM   Purchase
```

```
SELECT SUM(price * quantity)
FROM   Purchase
WHERE  product = 'bagel'
```

What do these mean?
以上SQL语句的意思是?

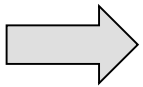
计算销售总额
计算某一种产品的销售总额

Simple Aggregations
简单聚合

Purchase

Product	Date	Price	Quantity
bagel	10/21	1	20
banana	10/3	0.5	10
banana	10/10	1	10
bagel	10/25	1.50	20

```
SELECT SUM(price * quantity)
FROM   Purchase
WHERE  product = 'bagel'
```



50 (= 1*20 + 1.50*20)

Grouping and Aggregation
分组和聚合

Purchase(product, date, price, quantity)

Find total sales after 10/1/2005 per product.
找出10/1/2005的每种产品的总销售额

```
SELECT   product,
                SUM(price * quantity) AS TotalSales
FROM     Purchase
WHERE    date > '10/1/2005'
GROUP BY product
```

Let's see what this means…
让我们看一下含义：

分组和聚合Grouping and Aggregation

```
SELECT   product,
                  SUM(price * quantity) AS TotalSales
FROM      Purchase
WHERE    date > '10/1/2005'
GROUP BY product
```

Semantics of the query:

1. Compute the FROM and WHERE clauses
计算FROM和WHERE子句

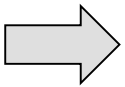
2. Group by the attributes in the GROUP BY
GROUP BY子句的属性分组合并

3. Compute the SELECT clause: grouped attributes and aggregates
计算SELECT子句获得分组后的属性和聚合

1. Compute the FROM and WHERE clauses
计算FROM-WHERE子句

```
SELECT   product, SUM(price*quantity) AS TotalSales
FROM     Purchase
WHERE    date > '10/1/2005'
GROUP BY product
```

FROM



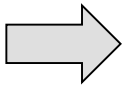
Product	Date	Price	Quantity
Bagel	10/21	1	20
Bagel	10/25	1.50	20
Banana	10/3	0.5	10
Banana	10/10	1	10

2. Group by the attributes in the GROUP BY
GROUP BY子句的属性分组合并

```
SELECT   product, SUM(price*quantity) AS TotalSales
FROM     Purchase
WHERE    date > '10/1/2005'
GROUP BY product
```

Product	Date	Price	Quantity
Bagel	10/21	1	20
Bagel	10/25	1.50	20
Banana	10/3	0.5	10
Banana	10/10	1	10

GROUP BY



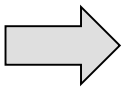
Product	Date	Price	Quantity
Bagel	10/21	1	20
	10/25	1.50	20
Banana	10/3	0.5	10
	10/10	1	10

3. Compute the SELECT clause: grouped attributes and aggregates
计算SELECT子句获得分组后的属性和聚合

```
SELECT   product, SUM(price*quantity) AS TotalSales
FROM     Purchase
WHERE    date > '10/1/2005'
GROUP BY product
```

Product	Date	Price	Quantity
Bagel	10/21	1	20
	10/25	1.50	20
Banana	10/3	0.5	10
	10/10	1	10

SELECT



Product	TotalSales
Bagel	50
Banana	15

GROUP BY v.s. Nested Queries
分组和嵌套查询

```
SELECT   product, Sum(price*quantity) AS TotalSales
FROM     Purchase
WHERE    date > '10/1/2005'
GROUP BY product
```

```
SELECT DISTINCT x.product,
                (SELECT Sum(y.price*y.quantity)
           FROM   Purchase y
           WHERE  x.product = y.product
                  AND y.date > '10/1/2005') AS TotalSales
FROM   Purchase x
WHERE  x.date > '10/1/2005'
```

HAVING子句
HAVING Clause

同样的查询，增加了限定条件买家数目大于100

Same query as before, except that we consider only products that have more than 100 buyers

```
SELECT   product, SUM(price*quantity)
FROM     Purchase
WHERE    date > '10/1/2005'
GROUP BY product
HAVING   SUM(quantity) > 100
```

HAVING clauses contains conditions on **aggregates**
HAVING子句包含了聚合中的条件

*Whereas WHERE clauses condition on **individual tuples…***
***但是** WHERE 子句适用于每一行或元组*

General form of Grouping and Aggregation
分组和聚合运算的通用的形式

```
SELECT      S
FROM        R1,…,Rn
WHERE       C1
GROUP BY    a1,…,ak
HAVING      C2
```

Evaluation steps:
运算步骤：

1. Evaluate FROM-WHERE: apply condition C_1 on the attributes in $R_1,\ldots,R_n$

对FROM-WHERE子句进行运算：在属性$R_1,\ldots,R_n$上进行条件C_1计算

2. GROUP BY the attributes $a_1,\ldots,a_k$

按照属性 $a_1,\ldots,a_k$ 进行分组

3. Apply condition C_2 to each group (may need to compute aggregates)

对每一个分组进行Having子句条件C_2计算

4. Compute aggregates in S and return the result

在S中计算聚合，返回结果

想一想，练一练

1.销售数据表Sales的准备：
（1）插入10行以上的销售数据；
（2）至少3个以上产品（自行填充）；

2. 销售数据表Sales数据聚合分析：
（1）计算每个单类产品的销售总额，应用Group By;
（2）找出销售中销售最少的产品，应用MIN；
（2’）找出销售中销售总量最少的产品，应用MIN；（用嵌套）
（3）找出销售中销售最多的产品，应用MAX；
（3’）找出销售中销售总量最多的产品，应用MAX；（嵌套）
（4）计算每种产品的销售量的总数，应用SUM；
（5）计算销售的产品的种类，应用count，注意要去重用distinct；
（6）计算出销售量大于10的产品的销售总额，应用Having子句条件过滤。

谢谢指正！