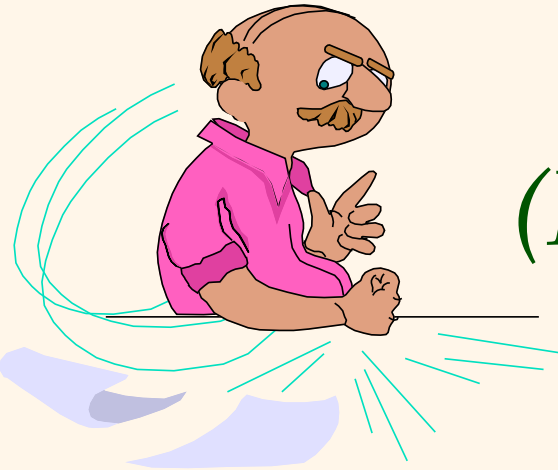


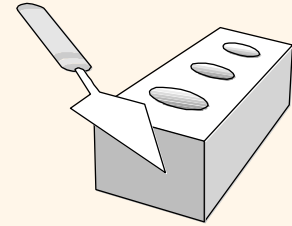
Introduction to Data Management

Lecture #11 *(Relational Languages I)*

Instructor: Mike Carey
mjcarey@ics.uci.edu



Announcements



❖ Homework notes

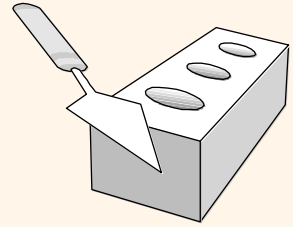
- HW #3 is due Friday (Saturday if one day late)
- HW #4 will come out on Monday (after the exam)

❖ Exam notes (time flies!)

- Midterm #1 is next Monday (**in class**)
- We'll use assigned seating – come early!
- You may bring an 8.5" x 11" (2-sided) cheat sheet
- Don't bank it all on last-minute Piazza answers...!

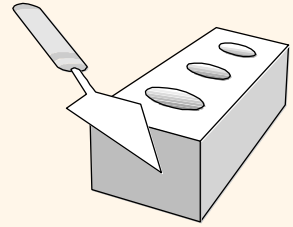
❖ Today's lecture plan:

- Relational languages – the next frontier...
- **Note:** Today's material won't be on Monday's exam! 😊



But 1st: A Word on Learning....

- ❖ “When **I** was your age...” (😊)
 - 8-bit μ processors, LEDs, hex keypad, 16-bit for its address space, ...
 - Unix first appeared, Unix/INGRES DB 64K process design point, and 1MB of memory was amazing!
 - FORTRAN, Pascal, C, Lisp, Snobol, APL, Quel ...
 - No web! (Just the early ArpaNet)
- ❖ Fast forward 35 years...
 - Your cell phones dwarf our ~1980 computing platforms
 - Python, Java, Go, C++, Ruby, SQL, SQL++, ...
 - Twitter... (😞)
 - WHAT THIS MEANS: It's critical to learn how to read and learn, how to search for info/resources, etc...!!!



On to Relational Query Languages!

- ❖ Query languages: Allow manipulation and *retrieval* of data from a database.
操作和恢复
- ❖ Relational model supports simple, powerful QLs:
 - Strong formal foundation based on logic.
 - Allows for much optimization.
- ❖ Query Languages **≠** programming languages!
 - QLs not expected to be “Turing complete.”
 - QLs not intended to be used for complex calculations.
 - *QLs support easy, efficient access to large data sets.*

语句结构

结构化查询语言包含**6个部分**：

一： **数据查询语言**（**DQL**:Data Query Language）：

其语句，也称为“数据检索**语句**”，用以从表中获得数据，确定数据怎样在应用程序给出。保留字**SELECT**是DQL（也是所有SQL）用得最多的动词，其他DQL常用的保留字有WHERE，ORDER BY，GROUP BY和HAVING。这些DQL保留字常与其他类型的SQL语句一起使用。

二： **数据操作语言**（DML：Data Manipulation Language）：

其语句包括动词**INSERT**，**UPDATE**和**DELETE**。它们分别用于添加，修改和删除表中的行。也称为动作查询语言。

三： **事务处理语言**（TPL）：

它的语句能确保被DML语句影响的表的所有行及时得以更新。TPL语句包括BEGIN TRANSACTION，COMMIT和ROLLBACK。

四： **数据控制语言**（DCL）：

它的语句通过GRANT或REVOKE获得许可，确定单个用户和用户组对**数据库对象**的访问。某些RDBMS可用GRANT或REVOKE控制对**表**单个列的访问。

五： **数据定义语言**（**DDL**）：

其语句包括动词CREATE和DROP。在数据库中创建新表或删除表（CREAT TABLE 或 DROP TABLE）；为表加入索引等。DDL包括许多与人**数据库目录**中获得数据有关的保留字。它也是动作查询的一部分。

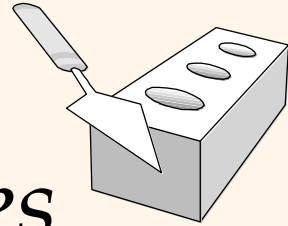
六： **指针控制语言**（CCL）：

它的语句，像DECLARE CURSOR，FETCH INTO和UPDATE WHERE CURRENT用于对一个或多个表单独行的操作。

分享



Formal Relational Query Languages

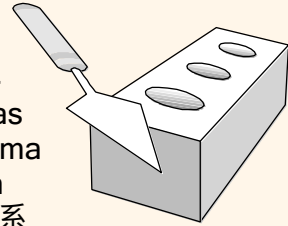


- ❖ Two mathematical Query Languages form the basis for “real” languages (e.g., SQL), and for their implementation:
 - Relational Algebra: 关系代数 More **operational**, very useful for representing execution plans.
 - Relational Calculus: 关系演算 Lets users describe **what** they want, rather than how to compute it.
(**Non-operational**, or declarative.)

Preliminaries

初步

在数据库中, schema (发音 “skee-muh” 或者 “skee-mah”, 中文叫模式) 是数据库的组织 and 结构, schemas and schemata 都可以作为复数形式。模式中包含了 schema 对象, 可以是表 (table)、列 (column)、数据类型 (data type)、视图 (view)、存储过程 (stored procedures)、关系 (relationships)、主键 (primary key)、外键 (foreign key) 等。数据库模式可以用一个可视化的图来表示, 它显示了数据库对象及其相互之间的关系



- ❖ A query is applied to *relation instances*, and the result of a query is *also* a relation instance.
 - *Schemas of input* relations for a query are *fixed* (but query will run regardless of instance!)
查询应用于关系实例, 查询的结果也是关系实例
 - The *schema for the result* of a given query is also *fixed*! Determined by applying the definitions of the query language's constructs.
- ❖ Positional vs. named-field notation:
 - Positional notation easier for formal definitions, but named-field notation far more readable.
 - Both used in SQL (but try to avoid positional stuff!)

Relation instance

A relation instance is a tuple or row in a relation, i.e. one particular combination of attribute values.

Let there be a relation Student. The schema of the relation Student has the following attributes :

{ID,first_name,last_name}

Now each of the attributes have their own domain i.e. a range of values they can take up.

The relation is an unordered set of tuples in which each tuple is a part of the cross-product of the domains. In other words, every tuple present in the Student relation should be a combination of valid values (present in the domain) of each attribute.

Example Instances

R1

<u>sid</u>	<u>bid</u>	<u>day</u>
22	101	10/10/96
58	103	11/12/96

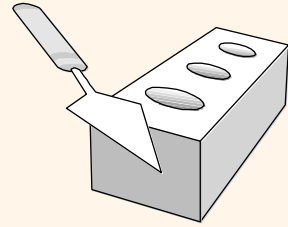
S1

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

- ❖ “Sailors” and “Reserves” relations for our examples.
- ❖ We’ll use positional or named field notation, and assume that names of fields in query results are “^{继承}inherited” from names of fields in query input relations (when possible).



Relational Algebra

❖ Basic operations:

- Selection (σ) Selects a subset of rows from relation.
- Projection (π) Omits unwanted columns from relation.
- Cross-product (\times) Allows us to combine two relations.
- Set-difference ($-$) Tuples in reln. 1, but not in reln. 2.
- Union (\cup) Tuples in reln. 1 and/or in reln. 2.

❖ Additional operations:

- Intersection, join, division, renaming: Not essential, but (very!) useful. (I.e., don't add expressive power, but...)

❖ Since each operation returns a relation, **operations can be composed!** (Algebra is “closed”.)

关系代数是一种**过程化查询语言**。它包括一个运算的集合，这些运算以一个或两个关系为输入，产生一个新的关系作为结果。关系代数的**基本运算**有：

名称	英文	符号
选择	select	σ
投影	project	Π
并	union	\cup
集合差	set-difference	$-$
笛卡儿积	Cartesian-product	\times
更名	rename	ρ

除了上面的6种基本运算之外，还有一些**其他运算**，其他运算是可以用基本运算来表示的。但是在实际中，我们为了方便使用，单独的建立了一种运算来表示，其他运算有：

名称	英文	符号
集合交	intersection	\cap
自然连接	natural join	\bowtie
赋值	assignment	\leftarrow

选择某table里的东西，table要放在小括号里，可以并列，或者同列，然后符合情况是写在小写处在中间那里

选择运算

英文： select

字符： σ

读音： sigma

选择运算在关系中选择出能够满足给定谓词的元组。将那些不满足谓词的元组剔除，组成一个新的关系。在 σ 后面小写谓词代表查询条件，括号中写要操作的关系。可以使用=， \neq ，>，<， \leq ， \geq 来进行谓词的判断。另外，也可以使用and(\wedge)or(\vee)not(\neg)将多个谓词组合成一个较大的谓词。

示例:

$\sigma_{\text{age}>18}(\text{User})$ 在User关系中查找出年龄大于18的所有元组并返回这些元组组成的关系

$\sigma_{\text{age}>20 \wedge \text{salary}>10000}(\text{User})$ 在User关系中查找出年龄大于20并且工资高于10000的所有元组并返回这些元组组成的关系

找出的table后再找出某列

投影运算

英文： project

字符： Π

读音： pi（是希腊字母 π 的大写形式）

如果我们只对一个关系中的某些属性感兴趣而不是所有，那么我们使用投影关系来选择出我们想要的属性，投影关系返回一个仅含有这些属性的关系。因为关系是集合，所以将返回的关系中所有重复的元组将被剔除。

示例：

$\Pi_{\text{name}}(\sigma_{\text{age}>18}(\text{User}))$ 在User关系中查找出年龄大于18的所有元组并返回这些元组的姓名name组成的关系。

相同属性的并在一起

并运算

英文： union

字符： U

有时我们需要把两个关系中的内容联系起来，或者一个关系经过不同的查询，我们希望把结果联系在一起。这就要使用并运算。没有什么不同的，和集合中的并很相似。需要注意的是，并运算处理的两个关系必须具有相同的属性，否则并运算将没有意义。

示例：

可以运行，但是运行出来毫无意义

$\Pi_{\text{name}}(\sigma_{\text{age}>18}(\text{User})) \cup \Pi_{\text{name}}(\sigma_{\text{address}=\text{"NewYork"}}(\text{Home}))$

在User关系中找出所有年龄大于18的姓名，在Home关系中找出所有家在NewYork的人的姓名，将这两个关系取并集，得到一个并关系。

差运算

英文： set-difference

字符： -

我们用一个关系A差运算另一个关系B，表示剩下A中有但是B中没有的元组组成的关系。和并运算相同的，我们必须保证-运算在相容的关系之间进行。

示例：

$\Pi_{\text{name}}(\sigma_{\text{age}>18}(\text{User})) - \Pi_{\text{name}}(\sigma_{\text{address}=\text{"NewYork"}}(\text{Home}))$

在User关系中找出所有年龄大于18的姓名，在Home关系中找出所有家在NewYork的人的姓名，得到User中存在而Home中不存在的人的姓名的关系。

笛卡儿积运算

左边的一个对应右边的一整个table

英文： Cartesian-product

字符： ×

有时我们需要把两个不相关的关系连接起来，但是这两个关系之中的属性却各不相同。对于这种不相容的情况我们不能使用交并差运算。笛卡儿乘积，用第一个关系A中的一个元组和第二个关系B中的所有元组结合，创造出一条包含着所有属性的新元组（如果在两个关系中有同名属性c，则在新关系中用A.c和B.c分别来表示）。这样得到的新关系中一共有A的元组数乘以B的元组数条信息。

示例：

User × Home 将User关系和Home关系做笛卡儿乘积运算

更名

英文：rename

字符： ρ

读音：rho

关系表达式的结果没有给我们能够引用的名字。如果想要引用一个产生的关系，最基础的就是把这句话再写一遍，但是这种方法不是很好的。我们可以通过 ρ 表示更名或是命名，为一个关系起个名。

示例：

$\rho_{adult}(\Pi_{name}(\sigma_{age>18}(User)))$ 将User关系中所有年龄大于18的人的姓名取出作为一个新的关系，并把这个关系起名为adult。

交运算（其他运算）

英文：intersection

字符： \cap

集合交运算表示出在A和B两个关系中都存在的元组的新关系。A和B两个元组应该是属性相同的。交运算是其他运算而不是基础运算。我们可以通过 $A-(A-B)$ 得到 $A \cap B$ 。

示例：

$\Pi_{name}(\sigma_{age>18}(User)) \cap \Pi_{name}(\sigma_{address="NewYork"}(Home))$

在User关系中找出所有年龄大于18的姓名，在Home关系中找出所有家在NewYork的人的姓名，将这两个关系中都存在的姓名取出作为新的关系。

自然连接（其他运算）

英文： natural join

字符： \bowtie

有的时候我们需要把两个属性并不是完全相同的关系连接在一起，就像笛卡儿积做的那样。但是我们又不想直接使用笛卡儿积，因为这种方法实在是耗时耗力，我们希望得到更为简练有效的数据。这时我们就需要自然连接。自然连接将两个关系A和B的共同属性作为根本，将两个表中这些共同属性的值都相同元组拼接在一起作为一个新的元组，而将剩下不能拼接的部分全部舍弃，这样得到一个新的关系。

自然连接也是一个其他运算。我们可以先将A和B关系做笛卡儿积，然后基于两个关系中相同的属性做相等性判断，去掉不等的元组，最后将属性简化，得到自然连接的结果。

示例：

User \bowtie Home 将User关系和Home关系做自然连接

如果完全没有相同的属性，那就是把两个table并在一起,跟笛卡尔一样

赋值运算（其他运算）

英文： assignment

字符： \leftarrow

实际上，赋值和更名很像，只不过赋值可以单独的写在一句话中，下面可以使用箭头左侧的名字作为右边关系的表示。

示例：

$\text{temp1} \leftarrow \Pi_{\text{name}}(\sigma_{\text{address}=\text{"NewYork"}}(\text{Home}))$

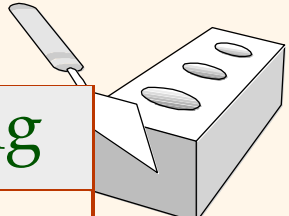
$\text{temp2} \leftarrow \Pi_{\text{name}}(\sigma_{\text{age}>18}(\text{User}))$

$\text{temp1} \cap \text{temp2}$

在User关系中找到所有年龄大于18的姓名，在Home关系中找到所有家在NewYork的人的姓名，将这两个关系中都存在的姓名取出作为新的关系。

Projection

- ❖ Removes attributes that are not in *projection list*.
- ❖ *Schema* of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- ❖ *Relational* projection operator has to eliminate *duplicates*! (Why??)
 - Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it. (Q: Why not?)



sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$\pi_{sname, rating}(S2)$

这个是希望最后出来什么
有多少种类就拿多少

age
35.0
55.5

$\pi_{age}(S2)$

Selection



sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

- ❖ Selects rows that satisfy a *selection condition*.
- ❖ No duplicates in result! (Why?)
- ❖ *Schema* of result identical to schema of its (only) input relation.
- ❖ *Result* relation can be the *input* for another relational algebra operation! (This is *operator composition*.)

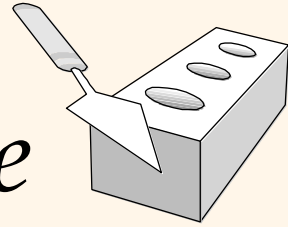
$$\sigma_{rating > 8}(S_2)$$

sname	rating
yuppy	9
rusty	10

$$\pi_{sname, rating}(\sigma_{rating > 8}(S_2))$$

选择sname和rusty这两个列里面的rating大于8的

Union, Intersection, Set-Difference



- ❖ All of these operations take two input relations, which must be union-compatible:
 - Same number of fields.
 - “Corresponding” fields are of the same type.
- ❖ What is the *schema* of result?

sid	sname	rating	age
22	dustin	7	45.0

$S1 - S2$ 拿S1里面S2没有的

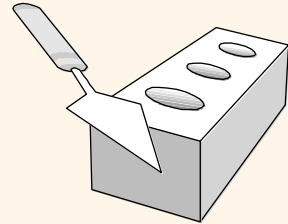
sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$S1 \cup S2$ 拿S1和S2一起的

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$S1 \cap S2$ 拿S1和S2不同的

Q: Any issues w/ duplicates?



Cross-Product

- ❖ $S1 \times R1$: Each S row is paired with each R1 row.
- ❖ *Result schema* has one field per field of S1 and R1, with field names “inherited” if possible.
 - *Conflict*: Both S1 and R1 have a field called *sid*.

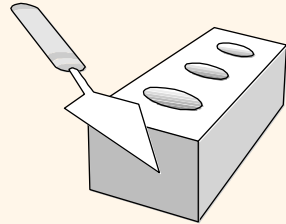
(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

Result
relation name

Attribute
renaming list

Source
expression
(anything!)

- *Renaming operator*: $\rho (C(1 \rightarrow sid1, 5 \rightarrow sid2), S1 \times R1)$



Renaming

❖ *Conflict*: S1 and R1 both had *sid* fields, giving:

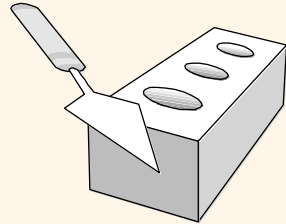
(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
...
58	rusty	10	35.0	58	103	11/12/96

❖ Several renaming notations available:

$\rho (S1R1(1 \rightarrow sid1), S1 \times R1)$ ← Positional renaming

$\rho (TempS1(sid \rightarrow sid1), S1)$ ← Name-based renaming
 $TempS1 \times R1$

$(\pi_{sid \rightarrow sid1, sname, rating, age} (S1)) \times R1$ ← **Generalized projection**
(I like this notation best! 😊)



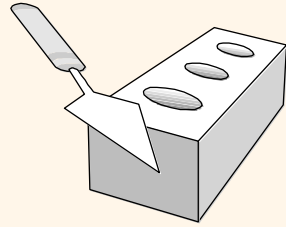
Joins

❖ Condition Join: $R \bowtie_c S = \sigma_c (R \times S)$

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	58	103	11/12/96

$$S_1 \bowtie_{S_1.sid < R_1.sid} R_1$$

- ❖ *Result schema* same as that of cross-product.
- ❖ Fewer tuples than cross-product, so might be able to compute more efficiently
- ❖ Sometimes (often!) called a *theta-join*.



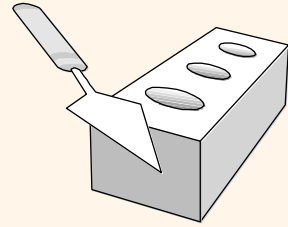
More Joins

- ❖ Equi-Join: A special case of condition join where the condition c contains only *equalities*.

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

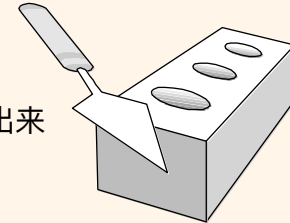
$$S_1 \bowtie_{S_1.sid = R_1.sid} R_1$$

- ❖ Result schema similar to cross-product, but only one copy of fields for which equality is specified.
- ❖ Natural Join: An equijoin on *all* commonly named fields.



Division

- ❖ Not a primitive operator, but extremely useful for expressing queries like:
Find sailors who have reserved all boats.
- ❖ Let A have 2 fields, x and y , while B has one field y ,
A/B的意思是A里面的x和y, 在b里面的y, 只有有相同y的才可以留下
so we have relations $A(x,y)$ and $B(y)$:
 - **A/B contains the x tuples (e.g., sailors) such that for every y tuple (e.g., boat) in B , there is an xy tuple in A .**
 - Or: If the set of y values (boats) associated with an x value (sailor) in A contains all y values in B , the x value is in A/B .
- ❖ In general, x and y can be any lists of fields; y is the list of fields in B , and $x \cup y$ is the list of fields of A .



首先A是一个要比B大的table，然后，B里面的每一个都能在A里共同对应上的，就拿出来

Examples of Division A/B

在A里与p2对应的是，
1234，但是与P4对应的只有
1，4，那共同的只有1，4

A里与p2对应的是1234

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2

B1



sno
s1
s2
s3
s4

A/B1

pno
p2
p4

B2



sno
s1
s4

A/B2

pno
p1
p2
p4

B3



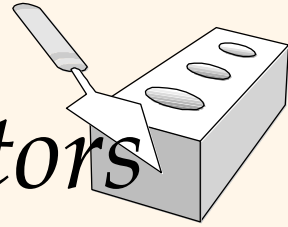
sno
s1

A/B3

同理因为P1只对1，所以最后留下1

Expressing A/B Using Basic Operators

(Advanced Topic – Just FYI ☺)

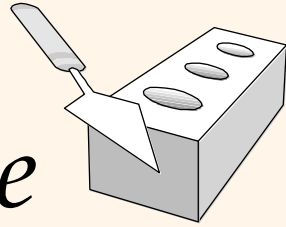


- ❖ Division not an essential op; just a useful shorthand.
(Also true of joins, but joins are so common and important that relational database systems implement joins specially.)
- ❖ *Idea*: For $A(x,y)/B(y)$, compute all x values that are not “disqualified” by some y value in B .
 - x value is *disqualified* if by attaching a y value from B , we obtain an xy tuple that *does not appear* in A .

Disqualified x values (D): $\pi_x ((\pi_x(A) \times B) - A)$

A/B : $\pi_x(A) - D$

Ex: Wisconsin Sailing Club Database



Sailors

sid	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	4	25.5
95	Bob	3	63.5

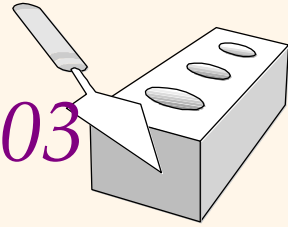
Reserves

sid	bid	date
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/93

Boats

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Find names of sailors who've reserved boat #103



Sailors(sid, sname, rating, age)

Reserves(sid, bid, day)

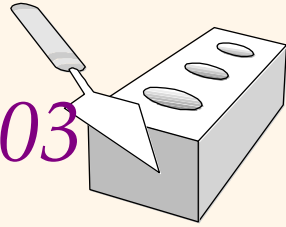
Boats(bid, bname, color)

❖ Solution 1: $\pi_{sname}((\sigma_{bid=103} \text{Reserves}) \bowtie \text{Sailors})$

❖ Solution 2: $\rho(\text{Temp1}, \sigma_{bid=103} \text{Reserves})$
 $\rho(\text{Temp2}, \text{Temp1} \bowtie \text{Sailors})$
 $\pi_{sname}(\text{Temp2})$

❖ Solution 3: $\pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie \text{Sailors}))$

Find names of sailors who've reserved boat #103



Sailors(sid, sname, rating, age)

Reserves(sid, bid, day)

Boats(bid, bname, color)

❖ Solution 1: $\pi_{sname}((\sigma_{bid=103} \text{Reserves}) \bowtie \text{Sailors})$

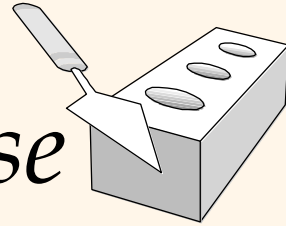
❖ Solution 2: $Temp1 = \sigma_{bid=103} \text{Reserves}$

$Temp2 = Temp1 \bowtie \text{Sailors}$

$\pi_{sname}(Temp2)$

❖ Solution 3: $\pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie \text{Sailors}))$

Ex: Wisconsin Sailing Club Database



$\sigma_{bid=103} \text{Reserves}$

sid	bid	date
22	103	10/8/98
31	103	11/6/98
74	103	9/8/93

$\pi_{sname}((\sigma_{bid=103} \text{Reserves}) \bowtie \text{Sailors})$

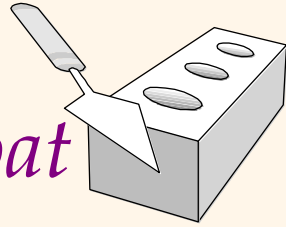
$(\sigma_{bid=103} \text{Reserves}) \bowtie \text{Sailors}$

sid	bid	date	sname	rating	age
22	103	10/8/98	Dustin	7	45.0
31	103	11/6/98	Lubber	8	55.5
74	103	9/8/93	Horatio	9	35.0

sname
Dustin
Lubber
Horatio

(Solution 1)

Find names of sailors who've reserved a *red* boat



Sailors(sid, sname, rating, age)

Reserves(sid, bid, day)

Boats(bid, bname, color)

❖ Information about boat color only available in Boats; so need to do another join:

$$\pi_{sname}((\sigma_{color='red'} Boats) \bowtie Reserves \bowtie Sailors)$$

bid只在boat和reserves里有，reserves和Sailors有bid，所以需要reserves做中间人

❖ A more “efficient” solution:

$$\pi_{sname}(\pi_{sid}((\pi_{bid} \sigma_{color='red'} Boats) \bowtie Res) \bowtie Sailors)$$

moreefficient就是同样的拿出来，但是就是只拿需要的，就好像Boats和Res只需要bid，所以就拿bid，Res和Sailors只需要sid连，所有只留sid，然后最后拿Sailors的sname就可以

A query optimizer will find the latter, given the 1st query!