

# CIS4301 Notes

Ryan Roden-Corrent

February 5, 2014

## 1 Relational Algebra

See <http://cise.ufl.edu/class/cis4301sp14/slides/ra.ppt> for class slides on this. These notes are mostly a condensed version of the slides, the slides contain some nice table images to help you visualize the operations.

### 1.1 What is Relational Algebra?

**Operators** most common actions you execute on relations

**Operands** relations or variables that represent relations

### 1.2 Core Relational Algebra

**Union, Intersection, Difference** most common actions you execute on relations

**Selection** picking certain rows

**Projection** picking certain columns

**Products/Joins** compositions of relations

**Renaming** of relations and attributes

#### 1.2.1 Selection

$R1 := \sigma_c(R2)$

C is a condition that refers to attributes of R2

R1 is all tuples of R2 that satisfy C

Relation Sells:			$JoeMenu := \sigma_{bar="Joe's"}(Sells) :$		
bar	beer	price	bar	beer	price
Joe's	Bud	2.50	Joe's	Bud	2.50
Joe's	Miller	2.75	Joe's	Miller	2.75
Sue's	Bud	2.50	Sue's	Bud	2.50
Sue's	Miller	3.00	Sue's	Miller	3.00

### 1.2.2 Projection

$R1 := \pi_L(R2)$

L is a list of attributes from R2's schema

R1 contains only the attributes of R2 listed in L (in the order they are listed)

Set operation: removes duplicate tuples

### 1.2.3 Extended Projection

$R1 := \pi_L(R2)$

Like projection, but L can contain arbitrary expressions involving attributes. Example:  $\pi_{A+B \rightarrow C, A, A}(R)$  will create a new table where the first column, C, is the sum of A and B, and the next two columns A1 and A2 are copies of the original A.

### 1.2.4 Product

$R3 := R1 \times R2$

- pair each tuple t1 of R1 with each tuple t2 of R2
- Concatenation  $t_1 t_2$  is a tuple of  $R_3$
- Schema of R3 is the attributes of R1 and then R2, in order
- Beware attribute A of same name in R1 and R2: use R1.A and R2.A

Example:  $R3 := R1 \times R2$

Duplicate column B, so use R1.B and R2.B to differentiate.

### 1.2.5 Theta Join

$R3 := R1 \bowtie_C R2$  Equivalent to taking the product  $R1 \times R2$  and applying  $\sigma_C$  to the result.

$R \bowtie_\theta S \equiv \sigma_\theta(R \times S)$

C can be any boolean-values condition.

### 1.2.6 Natural Join

Remove duplicate columns, only return columns that naturally combine.  $R3 := R1 \bowtie R2$

### 1.2.7 Renaming

Gives a new schema to a relation.

$R1 := \rho_{R1(A_1, \dots, A_n)}(R2)$  or  $R1(A_1, \dots, A_n) := R2$  (simplified notation)

R1 is a relation with the same tuples as R2 but the attributes  $A_1, \dots, A_n$ .

## 1.3 Building Complex Expressions

Combine operators with parentheses and precedence rules.

Three notations:

1. Sequences of assignment statements
2. Expressions with several operators
3. Expression trees

### 1.3.1 Operator Precedence

1.  $[\sigma, \pi, \rho]$  (highest)
2.  $[X, \bowtie]$
3.  $\cap$
4.  $\cup$

### 1.3.2 Expression Trees

Leaves are operands (variables or constant relations).

Interior nodes are operators applied to children.

**Example:**

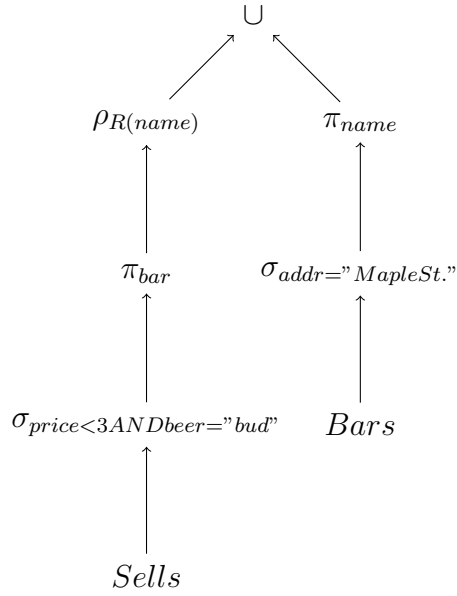


Figure 1: Find the names of all bars that are either on maple street or sell Bud for less than \$3

## 1.4 Relational Algebra on Bags

A bag is like a set, but duplicate elements are allowed. SQL is a bag language. Operations like projection are more efficient on bags than on sets.

### 1.4.1 Bag Union

Just add elements together, including duplicates.

$$\{1, 2, 1\} \cup \{1, 1, 2, 3, 1\} = \{1, 1, 1, 1, 1, 2, 2, 3\}$$

### 1.4.2 Bag Intersection

Minimum number of duplicates in resulting set.

$$\{1, 2, 1, 1\} \cap \{1, 2, 1, 3\} = \{1, 1, 2\}$$

### 1.4.3 Bag Difference

Result contains all tuples in first relation that aren't in the second.

$$\{1, 2, 1, 1\} - \{1, 2, 3\} = 1, 1$$

### 1.4.4 Bag Laws != Set Laws

Set union is **idempotent**, (result does not change if applied multiple times.)

For a bag union, if  $x$  appears  $n$  times in  $S$ , then it appears  $2n$  times in union.

## 2 Why SQL?

Say what to do rather than how to do it.

**SELECT** desired attributes

**FROM** one or more tables

**WHERE** some condition holds

### 2.1 Example:

Names of beers made by Anheuser-Busch:

---

```
SELECT name
FROM Beers
WHERE manf = 'Anheuser-Busch'
```

---

name	manf
Bud	Anheuser-Busch

\* indicates "all attributes in relation.

---

```
SELECT name
FROM Beers
WHERE manf = 'Anheuser-Busch'
```

---

Try adding things to SELECT \*

Rename *name* to *beer* using **AS**

---

```
SELECT name AS beer, manf
FROM Beers
WHERE manf = 'Anheuser-Busch'
```

---

---

```
SELECT bar, ber,
price*114 AS PriceInYen
FROM Beers
```

---

Using **Likes**(*drinker*, *beer*): Put 'likes Bud' in attribute whoLikesBud

---

```
SELECT drinker
  'likes Bud' AS whoLikesBud
FROM Likes
WHERE beer = 'Bud'
```

---

## 2.2 Information Integration

Each **Data warehouse** manages several subdatabases that each have multiple tables.

Example: Each bar has its own relation Menu(beer,price) tat we need to integrate.

---

```
SELECT 'Joe''s Bar', beer, price
FROM Menu;
```

---

Do this for each bar to create a consistent schema, then take the Union.

**Note:** the repeated single quote is used to escape the nested quote.

## 2.3 Complex Conditions

Using Sells(bar, beer, price), find the price Joe charges for Bud.

---

```
SELECT price
FROM Sells
WHERE bar = 'Joe''s Bar' AND beer = 'Bud';
```

---

## 2.4 Patterns

Usable on **VARCHAR**. Useable on **DATE**, **INT**? Maybe.

```
<Attribute> LIKE <pattern>
<Attribute> iLIKE <pattern>    --case insensitive like
<Attribute> NOT LIKE <pattern>
```

% means any string

\_ means any character

Example: choose numbers with two digits

---

```
FROM NUMBERS
WHERE number LIKE "__"
```

---

Example: choose numbers with at least two digits

---

```
FROM NUMBERS
WHERE number LIKE "__%"
```

---

Find drinkers with phone number starting with 555.

---

```
SELECT name
FROM Drinkers
WHERE phone LIKE '%555- _ _ _ _';
```

---

## 2.5 NULL Values

NULL could mean:

**Missing Value** we know it exists but don't have value

**Inapplicable** e.g. spouse of unmarried person

NULLs can be tricky, try to avoid if possible.

### 2.5.1 3-valued logic

TRUE = 1, FALSE = 0, UNKNOWN = 1/2

AND = MIN, OR = MAX, NOT(x) = 1-x

Ex:

TRUE AND (FALSE OR NOT(UNKNOWN)) = MIN(1, MAX(0, (1-1/2)))

Start on innermost paren and change truth values to numeric values. E.g. NOT(UNKNOWN) = (1-1/2)

### 2.5.2 Surprising Example

bar	beer	price
Joes	Bud	NULL

---

```
SELECT bar
FROM Sells
WHERE price < 2.00 OR price >= 2.00;
```

---

The comparisons become UNKNOWN because they are undefined for NULL.  
Result is

NULL OR NULL  
Max(1/2, 1/2) = 1/2

Expected to get everything, got nothing

## 2.6 Multirelation Queries

Distinguish between attributes of same name by "relation.attribute"

### 2.6.1 Examples

Using `Likes(drinker,beer)`, `Frequents(drinker, bar)`, find the beers liked by at least one person who frequents Joe's.

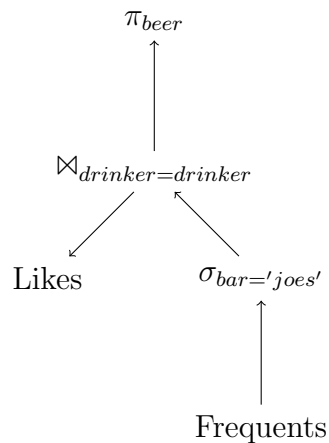
---

```
SELECT beer
FROM Likes, Frequents
WHERE Bar = 'Joes' AND
      Frequents.drinker =
      Likes.drinker;
```

---

A theta join  $\bowtie_{=}$  is used to equate the drinker attribute from each table. Can be done in any order.

For example:



## 3 Multirelational Queries

### 3.1 Joining Two Relations

Find the beers liked by at least one person who frequents Joe's

---

```
SELECT beer
FROM Likes, Frequents
WHERE bar = 'Joes' AND
      Frequents.drinker =
      Likes.drinker;
```

---

Same as a  $\bowtie_{Frequents.drinker=Likes.drinker}$

### 3.2 Self-Join

Get all pairs of beers that have same manufacturer and a different name.



---

```

SELECT b1.name, b2.name
FROM Beers b1, Beers b2 --avoid naming conflicts
WHERE b1.manf = b2.manf AND
      b1.name < b2.name;

```

---

Why use  $\boxed{<}$  instead of  $\boxed{<>}$ ?  $<$  will order the results and get rid of duplicates.

### 3.3 Subqueries

A parenthesized SELECT-FROM-WHERE statement

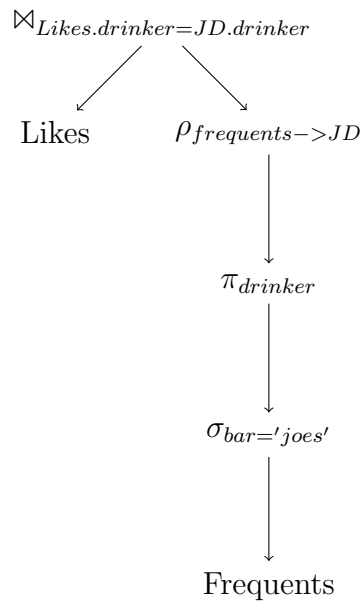
---

```

SELECT beer
FROM Likes, (SELECT drinker
              FROM Frequents                      --create a table of drinkers who frequent joes
              WHERE bar = 'Joes') AS JD          --and call that table JD
WHERE Likes.Drinker = JD.drinker --now operate on that table

```

---



#### 3.3.1 Query + Subquery Solution

Get the the bars where Miller is sold for the same price as bud at Joe's.

---

```

SELECT bar
FROM Sells
WHERE beer = 'Miller' AND
      price = (SELECT price --get the price from a subquery
              FROM Sells
              WHERE bar = 'Joes')

```

```
AND beer = 'Bud'  
LIMIT 1);      --Make sure you only get a single value
```

---

There is a runtime error if the subquery returns more than one value (hence the use of LIMIT).

### 3.3.2 IN

Get the name and manufacturer of each beer that Fred likes.

---

```
SELECT *  
FROM Beers  
WHERE name IN (SELECT beer  
FROM Likes  
WHERE drinker = 'Fred');
```

---

### 3.3.3 EXISTS

Correlated subquery: Walk one-by-one through tuples. At each tuple, perform the WHERE check. b1 refers to the tuple being checked.

---

```
SELECT name  
FROM Beers b1  
WHERE NOT EXISTS ( --only true if subquery returns empty set  
SELECT *      --get manufacturers who have only one beer  
FROM Beers  
WHERE manf = b1.manf AND  
name <> b1.name); --returns a tuple (manf, beer)
```

---

**EXISTS** looks at a table and finds out if the table returns any values. Either returns  $\emptyset$  or a table of results. **NOT EXISTS**(query) returns true only if the result of query is  $\emptyset$ . Internal query returns beers where the manufacturer is 'Bud' and the name is not 'Lite'.

### 3.3.4 ANY

$x = ANY(subquery)$  checks if x equals at least one tuple in the subquery.

### 3.3.5 ALL

$x <> ALL(subquery)$  is true iff x is not equal to any tuple in the subquery.

### 3.3.6 DISTINCT

Remove duplicates

---

**SELECT** x,y

---

### 3.4 Union, Intersect, Difference

Set operators. Require that arguments have same schema.

## 4 Bag Semantics

Just add ALL to any of the set operators e.g. UNION ALL.

$\delta$  remove duplicate tuples

$\tau$  sort tuples

$\gamma$  grouping and aggregation

Sorting

$R1 := \tau_L(R2)$

L is a list of attributes to list by, listed in order of priority.

### 4.1 Aggregation Operators

Not operators of relational algebra, apply to an entire column.

Builtins: SUM, AVG, COUNT, MIN, MAX

Can define your own in postgres.

#### 4.1.1 Grouping

$R1 := \gamma_L(R2)$

A	B	C
1	2	3
4	5	6
1	2	5

$\gamma_{A,B,AVG(C) \rightarrow X}(R) = ??$

First, group R by A and B.

A	B	C
1	2	3
1	2	5
4	5	6

Then, average C within groups

A	B	C
1	2	4
4	5	6

## 4.2 Outerjoin

A	B	B	C
1	2	2	3
4	5	6	7

A	B	C
1	2	3
1	2	
	6	7
4	5	
	2	3
4	5	
	6	7

A	B	C
1	2	3
4	5	NULL
NULL	6	7

### 4.2.1 JOIN Modifiers

R OUTER JOIN S is the core of an outerjoin. Can be modified by

- NATURAL before JOIN

## 4.3 Aggregations

NOTE: NULL's are ignored in Aggregations.

### 4.3.1 Return a table with the single column AVG(Price).

---

```
SELECT AVG(price)
FROM Sells
WHERE beer = 'Bud';
```

---

### 4.3.2 Return number of distinct prices for bud beers

---

```
SELECT COUNT(DISTINCT price)
FROM Sells
WHERE beer = 'Bud';
```

---

## 4.4 Grouping

Follow SELECT-FROM-WHERE by GROUP-BY and a number of attributes

### 4.4.1 Get the Average price for each beer

---

```
SELECT Beer, AVG(price)
FROM Sells
GROUP BY beer;
```

---

Anything in GROUP BY must also be in SELECT.

In relational algebra:  $\gamma_{beer, AVG(price)}$

Result:

beer	AVG(price)
bud	2.33
coors	4.00
...	...

### 4.4.2 Average Age by Gender

---

```
SELECT gender, AVG(age)
FROM People
GROUP BY age;
```

---

Result:

gender	AVG(age)
M	24
F	23

### 4.4.3 Illegal Example

---

```
SELECT bar, MIN(price)
FROM Sells
GROUP BY beer = 'Bud';
```

---

## 4.5 Having

Get only certain rows based on a condition.

### 4.5.1 Get average age only for genders where it is greater than 24

---

```
SELECT gender, AVG(age)
FROM People
GROUP BY gender
```

HAVING AVG(age) > 23

---

Result:

gender	AVG(age)
M	24