Relational Algebra

Complete Summary

Instructor:

Deepak Poonia

MTech, IISc Bangalore

GATE CSE AIR 53; AIR 67;
AIR 107; AIR 206; AIR 256

DBMS Complete Course:
https://www.goclasses.in/courses/Database-Management-Systems

NOTE :

Complete Discrete Mathematics & Complete Engineering

Mathematics Courses, by GO Classes, are **FREE** for ALL learners.

Visit here to watch : https://www.goclasses.in/s/store/

SignUp/Login on Goclasses website for free and start learning.

GO CLASSES

GO CLASSES

FREE

COMPLETE COURSE

Engineering Mathematics

GATE CS IT

Freely Available Features

FREE

- All Video Lectures
- Quizzes
- Homeworks, Practice Sets
- Annotated Notes
- Toppers' Hand Written Notes
- GATE PYQs Video Solutions

SACHIN MITTAL SIR

www.goclasses.in

/ GoClasses

GO Classes

GO CLASSES

GO CLASSES

FREE

COMPLETE COURSE

Discrete Mathematics

GATE CS IT

Freely Available Features

FREE

- All Video Lectures
- Quizzes, Homeworks
- Annotated Notes
- Toppers' Hand Written Notes
- Summary Lectures
- GATE PYQs Video Solutions

DEEPAK POONIA SIR

www.goclasses.in

/ GoClasses

GO Classes

GO CLASSES

GO CLASSES

FREE

COMPLETE COURSE

Linear Algebra

GATE DA

Freely Available Features

FREE

- All Video Lectures
- Quizzes
- Homeworks, Practice Sets
- Annotated Notes
- Toppers' Hand Written Notes
- GATE PYQs Video Solutions

SACHIN MITTAL SIR

www.goclasses.in

/ GoClasses

GO Classes

We are on Telegram. **Contact us** for any help.

Link in the Description!!

Join GO Classes **Doubt Discussion** Telegram Group :

Username:

@GATECSE_GOCLASSES

We are on Telegram. **Contact us** for any help.

Join GO Classes Telegram Channel, Username: **@GOCLASSES_CSE**

Join GO Classes **Doubt Discussion** Telegram Group :

Username: **@GATECSE_Goclasses**

(Any doubt related to Goclasses Courses can also be asked here.)

Join GATEOverflow Doubt Discussion Telegram Group :

Username: **@GateOverflow_CSE**

GO Classes

GO CLASSES

Relational Algebra

Complete Summary

Relational Algebra:

Relational Algebra is a Theoretical (or Mathematical or Formal) Query Language for Relational Data Model.

Recap:

Relational Model

Relational data model

GO Classes

- A database is a collection of relations (or tables)
- Each relation has a set of attributes (or columns)
- Each attribute has a name and a domain (or type)
  - Values are "atomic"
  - Can be strings, integers, reals, characters,..
  - Cannot be a struct, set, list, array, …
- Each relation contains a set of tuples (or rows)
  - Each tuple has a value for each attribute of the relation
  - Duplicate tuples are not allowed
    - Two tuples are duplicates if they agree on all attributes

☞ Recall: Simplicity is a virtue!

GO CLASSES

GO Classes

A Table

Table name

Column names

Product:

Name	Price	Category	Manufacturer
gizmo	$19.99	gadgets	GizmoWorks
Power gizmo	$29.99	gadgets	GizmoWorks
SingleTouch	$149.99	photography	Canon
MultiTouch	$203.99	household	Hitachi

Rows / Tuples / Records

Relational Model

- A relation: a set of records (more formally later)
- Schema: a description of data in terms of a data model
  - Name of the schema, name of each field, type of each field

Schema

```
Students(
    roll:   string,
    name:   string,
    age:    integer,
    gpa:    real
)
```

Students

roll	name	age	gpa
0001	John	23	9.1
0002	Amit	24	8.7
0003	Vivek	25	9.2
0004	Sandhya	23	9.0
....	....	....	....

Instance

Schema vs. instance

- Schema (metadata)
  - Specifies how the logical structure of data
  - Is defined at setup time
  - Rarely changes
  - But columns can be added/deleted
- Instance
  - Represents the data content
  - Changes rapidly, but always conforms to the schema

- Schema
  - *User* (*uid* int, *name* string, *age* int, *pop* float)
  - *Group* (*gid* string, *name* string)
  - *Member* (*uid* int, *gid* string)
- Instance
  - *User*: $\{\langle 142, \text{Bart}, 10, 0.9\rangle, \langle 857, \text{Milhouse}, 10, 0.2\rangle, \ldots\}$
  - *Group*: $\{\langle \text{abc}, \text{Book Club}\rangle, \langle \text{gov}, \text{Student Government}\rangle, \ldots\}$
  - *Member*: $\{\langle 142, \text{dps}\rangle, \langle 123, \text{gov}\rangle, \ldots\}$

Relational Query Languages

- Query = "retrieval program"
- Language examples:
  - Theoretical:
    1. Relational Algebra
    2. Relational Calculus
       a. tuple relational calculus (TRC)
       b. domain relational calculus (DRC)
  - Practical
    1. SQL (SEQUEL from System R)
    2. QUEL (Ingres)
    3. Datalog (Prolog-like)

- Theoretical QL's:
  - give semantics to practical QL's
  - key to understand query optimization in relational DBMSs

Next Topic:

Relational Algebra

Procedural vs Declarative

NOTE:

Relational Algebra is Procedural Query Language, Not Declarative.

Declarative programming is where you say what you want without having to say how to do it.

With procedural programming, you have to specify exact steps to get the result.

SQL is declarative, Relational Algebra is Procedural.

Let me give you a real-world example: I need a cup of tea.

Task

Procedural: (Relational Algebra)

1. Go to kitchen
2. Get sugar, milk, and tea,
3. Mix them, and heat over the fire till it boils
4. Put that in a cup and bring it to me

Declarative: (SQL)

1. Get me a cup of tea.

In a procedural language, you define the whole process and provide the steps how to do it.

In a declarative language, you just set the command or order, and let it be on the system how to complete that order. You just need your result without digging into how it should be done.

Queries in relational algebra are composed using a collection of operators, and each query describes a step-by-step procedure for computing the desired answer;
that is, queries are specified in an operational (Procedural) manner.

SQL is a declarative programming language. This means that you specify what you want, but you don't have to specify how to do it.

Relational Algebra is a procedural programming language (meaning that the query specifies exactly what operators to use and in what order).

Next Topic:

Relational Algebra

Some Important Points

Rel. Alg (RA) works on Pure Relational model.

1. Every Relation (Table) is a Set.
2. No Duplication of Tuples
3. No NULL values
4. One Primary key

THE RELATIONAL DATA MODEL (RM)

and the Relational Algebra

Characteristics of Relations

- Relations are defined as a (mathematical) *set* of tuples.
- Duplicate tuples are not allowed
- Order of tuples inside a relation is immaterial

NOTE:

There are no NULL values in relational algebra.

PS: Outer joins are not considered part of the classical relational algebra.

course.ccs.neu.edu/cs3200sp18s3/ssl/lectures/lecture_03_sql_1.pdf

lecture_03_sql_1 19 / 98 118%

*There are actually no concept of NULL in relational algebra

SQL: Part 1 (DML, Relational Algebra)

February 6, 2018

Next Topic:

Let's get started...

Relation Alg :

So, what we need to do to become Expert of Relational Algebra??

Simple:

1. Understand Each Operator Clearly.
2. Apply them to write & understand queries…

Next Topic:

Algebra Operators

GO CLASSES

Relational Algebra

GO Classes

- Procedural language
- Queries in relational algebra are applied to relation instances, result of a query is again a relation instance
- Six basic operators in relational algebra:

select	σ	selects a subset of tuples from reln
project	π	deletes unwanted columns from reln
Cartesian Product	$\times$	allows to combine two relations
Set-difference	$-$	tuples in reln. 1, but not in reln. 2
Union	$\cup$	tuples in reln 1 plus tuples in reln 2
Rename	ρ	renames attribute(s) and relation

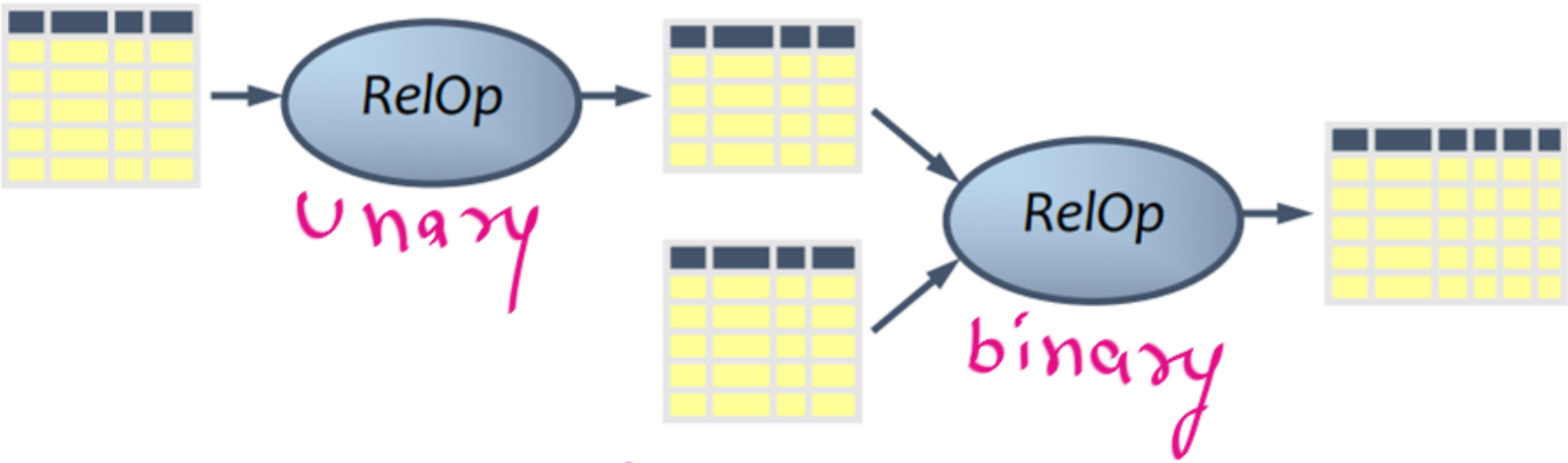
- The operators take one or two relations as input and give a new relation as a result (relational algebra is "closed").

Additional Operators (Derived operators)

These operators do not add any power (expressiveness) to the relational algebra but simplify common (often complex and lengthy) queries.

Set-Intersection	$\cap$	
Natural Join	$\bowtie$	
Condition Join	$\bowtie_C$	(also called Theta-Join)
Division	$\div$	

Relational algebra Operators



- Core operators: (6)
  - Selection, projection, cross product, union, difference, and renaming
- Additional, derived operators:
  - Join, natural join, intersection, etc.
- Compose operators to make complex queries

GO Classes

Next Topic:

Algebra Operators

1.

Selection Operator

Selection Operator: Symbol: σ

sigma

Task: filter Tuples based on a Tuple selection Condition C.

Syntax:

selection operation

$\sigma_{c}(R)$: Those Tuples that satisfy Condition c will be in the Result.

Condition for Tuple Selection

$\sigma_c(R)$

1. σ is unary opⁿ
2. Filter Tuples
3. Schema Remains Unchanged

2.4.6 Selection

The *selection* operator, applied to a relation R, produces a new relation with a subset of R's tuples. The tuples in the resulting relation are those that satisfy some condition C that involves the attributes of R. We denote this operation $\sigma_C(R)$. The schema for the resulting relation is the same as R's schema, and we conventionally show the attributes in the same order as we use for R.

Selection example

- Users with popularity higher than 0.5

$$\sigma_{pop>0.5} User$$

User

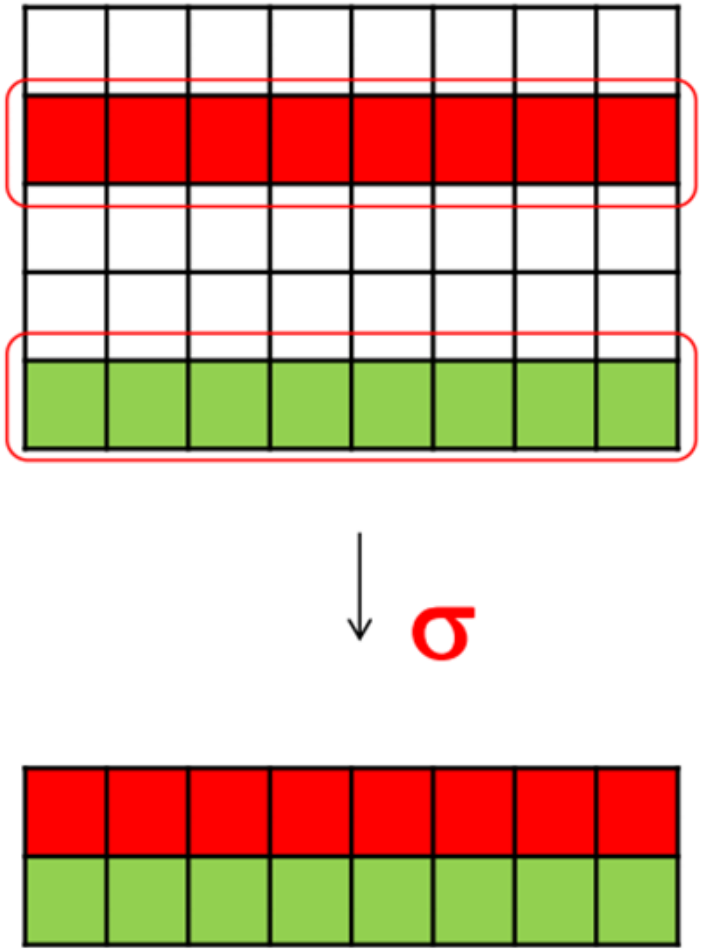
uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

$\sigma_{pop>0.5}$

uid	name	age	pop
142	Bart	10	0.9
857	Lisa	8	0.7
…	…	…	…

operations: Selection

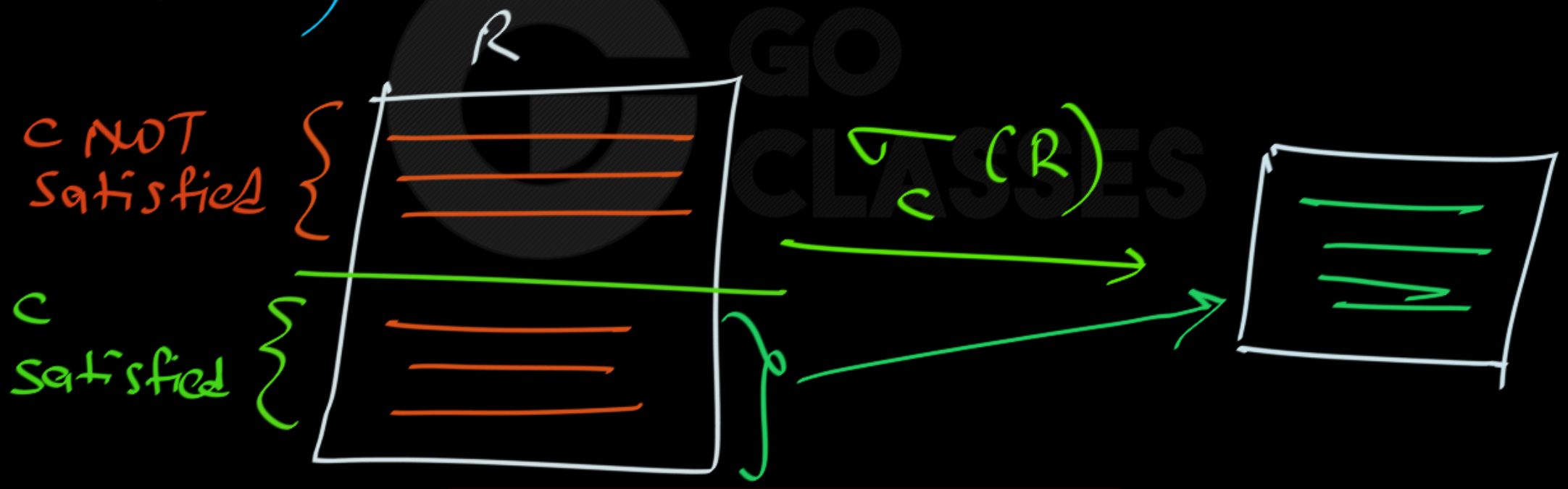
Produces a new relation with those tuples of **R** which satisfy condition **C**.



$$S = \sigma_{condition}(R)$$

σ : Horizontal partition of Table.

$\sigma_c(R)$



Selection example.

Query: list students with GPA >3

Student		
Name	GPA	Country
Bob	3	Canada
John	3	Britain
Tom	3.5	Canada
Maria	4	Mexico

S		
Name	GPA	Country
Tom	3.5	Canada
Maria	4	Mexico

$$S = \boxed{\sigma_{gpa>3}\,(\text{Student})}$$

GO Classes

Selection

- Unary opn
- Input: a table R
- Notation: $\sigma_p R$
  - p is called a selection condition (or predicate)
- Purpose: filter rows according to some criteria
- Output: same columns as R, but only rows or R that satisfy p

Select Operation – Example

Relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

$\sigma_{A=B \wedge D > 5}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

$\wedge$: AND

$\vee$: OR

$\neg$: NOT

$\sigma_{(A=B) \wedge (D>5)}(r)$

Tuple Selection Condition:

$\sigma_{P}(R)$

$(C > D)$ AND/OR $(D > 5)$ AND/OR $(E \leq 6)$

Select Operation

- Notation: $\sigma_p(r)$
- p is called the **selection predicate**

Where p is a formula in propositional calculus consisting of **terms** connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)
Each **term** is one of:

<attribute>*op* <attribute> or <constant>

where *op* is one of: $=, \neq, >, \geq. <. \leq$

- Example of selection:

$$\sigma_{dept_name=\text{“Physics”}}(instructor)$$

Selection

- R1 := σ_C(R2)
  - *C* is a condition (as in "if" statements) that refers to attributes of R2.
  - R1 is all those tuples of R2 that satisfy *C*.

More on selection

User (uid, name, age, pop)
Member (uid, gid)

- Selection condition can include any column of R, constants, comparison ($=$, $\leq$, etc.) and Boolean connectives ($\wedge$: and, $\vee$: or, $\neg$: not)

GO Classes

Example: Selection

Relation Sells:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

JoeMenu := $\sigma_{bar=\text{"Joe's"}}$(Sells):

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75

8.1.1 The SELECT Operation

The SELECT operation is used to choose a *subset* of the tuples from a relation that satisfies a **selection condition**.[3] We can consider the SELECT operation to be a *filter* that keeps only those tuples that satisfy a qualifying condition. Alternatively, we can consider the SELECT operation to *restrict* the tuples in a relation to only those tuples that satisfy the condition. The SELECT operation can also be visualized as a *horizontal partition* of the relation into two sets of tuples—those tuples that satisfy the condition and are selected, and those tuples that do not satisfy the condition and are filtered out. For example, to select the EMPLOYEE tuples whose department is 4, or those whose salary is greater than $30,000, we can individually specify each of these two conditions with a SELECT operation as follows:

$$\sigma_{Dno=4}(\text{EMPLOYEE})$$
$$\sigma_{Salary>30000}(\text{EMPLOYEE})$$

The selection operator σ specifies the tuples to retain through a *selection condition*. In general, the selection condition is a boolean combination (i.e., an expression using the logical connectives $\wedge$ and $\vee$) of *terms* that have the form *attribute* op *constant* or *attribute1* op *attribute2*, where op is one of the comparison operators $<, <=, =, \neq, >=,$ or $>$. The reference to an attribute can be by position (of the form $.i$ or i) or by name (of the form *.name* or *name*). The schema of the result of a selection is the schema of the input relation instance.

select the tuples for all employees who either work in department 4 and make over $25,000 per year, or work in department 5 and make over $30,000, we can specify

select the tuples for all employees who either work in department 4 and make over $25,000 per year, or work in department 5 and make over $30,000, we can specify the following SELECT operation:

$$\sigma_{(\text{Dno}=4 \textbf{ AND } \text{Salary}>25000) \textbf{ OR } (\text{Dno}=5 \textbf{ AND } \text{Salary}>30000)}(\text{EMPLOYEE})$$

Selection Operation

NOTE 1: The Condition in Selection Operation evaluates over each single row of the input table..

i.e. We can see ONLY one row at a time..

The <selection condition> is applied independently to each individual tuple t in R.

Results: $\sigma_c(R)$

1. Schema Unchanged.

2. Degree (Arity): Number of attributes

$$\text{Degree}(\sigma_c(R)) = \text{Degree}(R)$$

3. Cardinality: Number of Tuples

$$|\sigma_c(R)| \leq |R|$$

In general, the result of a SELECT operation can be determined as follows. The <selection condition> is applied independently to each *individual tuple t* in *R*. This

The SELECT operator is **unary**; that is, it is applied to a single relation. Moreover, the selection operation is applied to *each tuple individually*; hence, selection conditions cannot involve more than one tuple. The **degree** of the relation resulting from a SELECT operation—its number of attributes—is the same as the degree of *R*. The number of tuples in the resulting relation is always *less than or equal to* the number of tuples in *R*. That is, $|\sigma c\ (R)| \leq |R|$ for any condition *C*.

- You must be able to evaluate the condition over each single row of the input table!
  - Example: the most popular user

$$\sigma_{pop \geq every\ pop\ in\ User}\ User$$

WRONG!

Q:

Is Selection Operation Commutative ??

$$\sigma_{c_2}\left(\sigma_{c_1}(R)\right) \overset{?}{=} \sigma_{c_1}\left(\sigma_{c_2}(R)\right)$$

$$\sigma_{c_2}(\sigma_{c_1}(R)) \equiv \sigma_{c_1 \wedge c_2}(R)$$

$$\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_1 \wedge c_2}(R)$$

$$\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$$

Important Property of Selection Operator:

$$\boxed{\sigma_{c_1}\left(\sigma_{c_2}(R)\right) \equiv \sigma_{c_2}\left(\sigma_{c_1}(R)\right)}$$

$$\equiv \sigma_{c_1 \wedge c_2}(R)$$

Selection is Commutative

Note:

$$\sigma_{P_1}\left(\sigma_{P_2}\left(\sigma_{P_3}(R)\right)\right) \equiv \sigma_{P_1 \wedge P_2 \wedge P_3}(R)$$

properties of the SELECT operation:

- Determination of the result of SELECT: apply the select condition to each tuple and evaluate the condition
- SELECT operation is unary
- degree($\sigma_{condition}(R)$) = degree(R)
- # of tuples of $\sigma_{condition}(R) \leq$ # of tuples of R
- selectivity of a condition: how many tuples are selected
- commutativity of the SELECT operation:
  $\sigma_{condition1}(\sigma_{condition2}(R)) = \sigma_{condition2}(\sigma_{condition1}(R))$
- nested SELECT operations can be replaced with a multiple AND condition:
  $\sigma_{c_1}(\sigma_{c_2}(\ldots(\sigma_{c_n}(R)))) = \sigma_{c1\ AND\ c2\ AND \ldots AND\ c_n}(R)$

Next Topic:

Algebra Operators

2.

Projection Operator

Projection Operator: Symbol: π

→ filter attributes

Syntax:

$$\pi_{A_1, A_2, A_3}(R)$$

→ attribute List

R

A	B	C
1	1	2
1	1	3
1	2	1

$\pi_{A,B}(R)$

A	B
1	1
1	2

2.4.5 Projection

The *projection* operator is used to produce from a relation R a new relation that has only some of R's columns. The value of expression $\pi_{A_1, A_2, \ldots, A_n}(R)$ is a relation that has only the columns for attributes $A_1, A_2, \ldots, A_n$ of R. The schema for the resulting value is the set of attributes $\{A_1, A_2, \ldots, A_n\}$, which we conventionally show in the order listed.

Project Operation – Example

Relation *r*:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

$\Pi_{A,C}(r)$

A	C
α	1
α	1
β	1
β	2

=

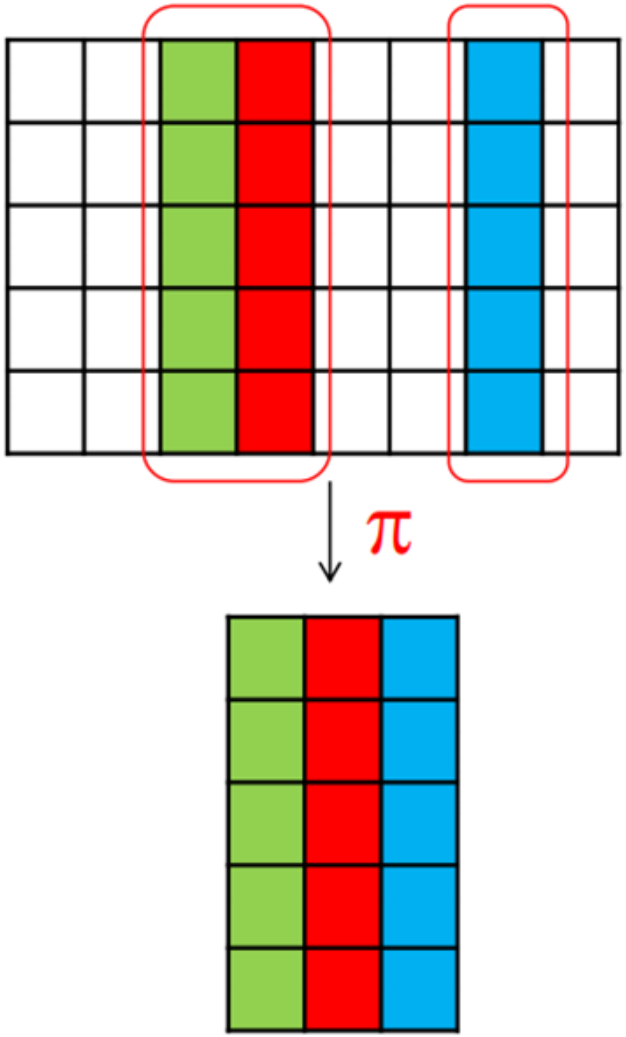
A	C
α	1
β	1
β	2

GO Classes

operations: Projection

Classes

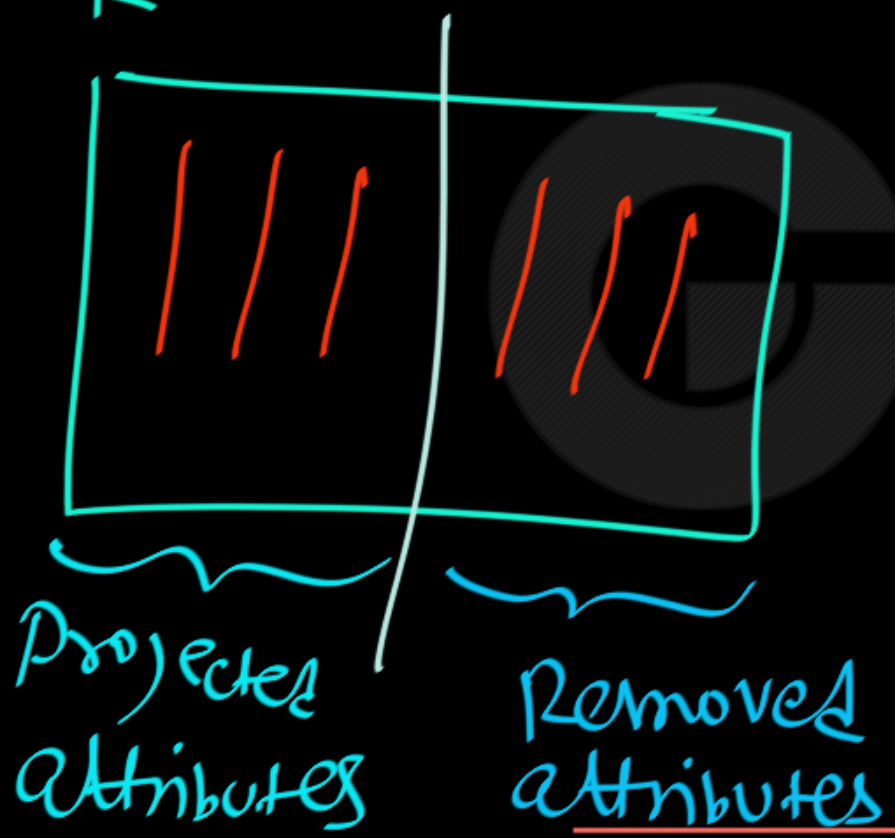
Produces from relation **R** a new relation that has only the A_1, …, A_n columns of **R**.



$$S=\pi_{\text{attribute list}}(R)$$

Note:

π : Vertical Partition of Table



Projection: example

Query: list names of students

Student			
SIN	Name	GPA	Country
111	Bob	3	Canada
222	John	3	Britain
333	Tom	3.5	Canada
444	Maria	4	Mexico

S
Name
Bob
John
Tom
Maria

$S = \pi_{Name}(Student)$

Projection

- Unary opn
- Input: a table R
- Notation: $\pi_L R$
  - L is a list of columns in R
- Purpose: output chosen columns
- Output: same rows, but only the columns in L

Projection example

- IDs and names of all users

$$\pi_{uid,name}\ User$$

uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

uid	name
142	Bart
123	Milhouse
857	Lisa
456	Ralph
…	…

Project Operation

- Notation:

$$\Pi_{A_1, A_2, \ldots, A_k}(r)$$

where A_1, A_2 are attribute names and r is a relation name.

- The result is defined as the relation of k columns obtained by erasing the columns that are not listed

- Duplicate rows removed from result, since relations are sets

- Example: To eliminate the *dept_name* attribute of *instructor*

$$\Pi_{ID,\ name,\ salary}(instructor)$$

GO Classes

More on projection

- Duplicate output rows are removed (by definition)
  - Example: user ages

$$\pi_{age}\, User$$

uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

age
10
8
…

GO Classes

Projection

- R1 := π_L(R2)
  - L is a list of attributes from the schema of R2.
  - R1 is constructed by looking at each tuple of R2, extracting the attributes on list L, in the order specified, and creating from those components a tuple for R1.
  - Eliminate duplicate tuples, if any.

Example: Projection

Relation Sells:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

Prices := $\pi_{beer,price}$(Sells):

beer	price
Bud	2.50
Miller	2.75
Miller	3.00

2 Relational Algebra Introduction

All of the operators in relational algebra take in a relation and output a different relation. A basic query looks like this:

$$\pi_{name}(dogs)$$

The π operator picks only the columns that it wants to advance to the next operator (just like SQL SELECT). In this case, the operator takes the `dogs` relation in as a parameter and returns a relation that only has the `name` column of the `dogs` relation. An important fact about relational algebra is that the relations are sets of tuples, meaning that they cannot have duplicates in them.

If the dogs relation is initially:

name	age
Scooby	10
Buster	15
Buster	20

The query above would return:

name
Scooby
Buster

Initially the two Busters are different because they have different ages, but once you get rid of the age column, they become duplicates, so only one remains in the output relation.

8.1.2 The PROJECT Operation

If we think of a relation as a table, the SELECT operation chooses some of the *rows* from the table while discarding other rows. The **PROJECT** operation, on the other hand, selects certain *columns* from the table and discards the other columns. If we are interested in only certain attributes of a relation, we use the PROJECT operation to *project* the relation over these attributes only. Therefore, the result of the PROJECT operation can be visualized as a *vertical partition* of the relation into two relations:

one has the needed columns (attributes) and contains the result of the operation, and the other contains the discarded columns. For example, to list each employee's first and last name and salary, we can use the PROJECT operation as follows:

$$\pi_{\text{Lname, Fname, Salary}}(\text{EMPLOYEE})$$

The resulting relation is shown in Figure 8.1(b). The general form of the PROJECT operation is

$$\pi_{\text{<attribute list>}}(R)$$

where π (pi) is the symbol used to represent the PROJECT operation, and <attribute list> is the desired sublist of attributes from the attributes of relation *R*. Again, notice that *R* is, in general, a *relational algebra expression* whose result is a relation, which in the simplest case is just the name of a database relation. The result of the PROJECT operation has only the attributes specified in <attribute list> *in the same order as they appear in the list*. Hence, its **degree** is equal to the number of attributes in <attribute list>.

Results: $\pi_L(R)$

L: attribute list

① Schema may change.

② $\text{Degree}(\pi_L(R)) \leq \text{Degree}(R)$

③ Cardinality:

$|\pi_L(R)| \leq |R|$

Because Duplicates gets removed if any

R

A	B
1	2
1	3
1	4
2	4

$|\pi_A(R)| = 2$

$|R| = 4$

$|\pi_B(R)| = 3$

Q:

To make sure that Cardinality of Projected relation is necessarily same the input relation for every instance;

What can you say about the Attribute List of projection operation?

Q:

To make sure that Cardinality of Projected relation is necessarily same the input relation for every instance;

What can you say about the Attribute List of projection operation?

Super key

R ; $\pi_L(R)$

If L : Superkey then

$$|\pi_L(R)| = |R|$$

If the attribute list includes only nonkey attributes of R, duplicate tuples are likely to occur. The PROJECT operation *removes any duplicate tuples*, so the result of the PROJECT operation is a set of distinct tuples, and hence a valid relation. This is known as **duplicate elimination**. For example, consider the

The number of tuples in a relation resulting from a PROJECT operation is always less than or equal to the number of tuples in R. If the projection list is a superkey of R—that is, it includes some key of R—the resulting relation has the *same number* of tuples as R. Moreover,

Q:

Is Projection Operation Commutative ??

$$\pi_{L_2}\left(\pi_{L_1}(R)\right) \overset{??}{=} \pi_{L_1}\left(\pi_{L_2}(R)\right)$$

Q:

Is Projection Operation Commutative ??

No

$$\pi_{L_2}\left(\pi_{L_1}(R)\right) \overset{??}{=} \pi_{L_1}\left(\pi_{L_2}(R)\right)$$

No

$R(ABCD)$

$$\pi_{AB}(\pi_{ABC}(R)) \stackrel{?}{=} \pi_{ABC}(\pi_{AB}(R))$$

L_1 L_2

Invalid

Property of Projection Operation:

$R(ABCD)$

$$\pi_{AB}\left(\pi_{ABC}(R)\right) \overset{?}{=} \pi_{AB}(R)$$

L_2 L_1 Yes

$$\boxed{\pi_{L_2}(\pi_{L_1}(R))}$$

: To be Valid,

$L_2 \subseteq L_1$

Property of Projection Operation:

$$\pi_{L_2}(\pi_{L_1}(R)) \equiv \pi_{L_2}(R)$$

$$\boxed{\text{If } L_2 \subseteq L_1}$$

general syntax/properties of the PROJECT operation:

$\pi_{attribute\ list}(R)$

- PROJECT operation is unary
- degree($\pi_{attribute\ list}(R)$) = # attributes in $attributelist$)
- # of tuples of $\pi_{attribute\ list}(R) \leq$ # of tuples of R
  (it can be less because of duplicate elimination)
  (equality occurs when the attribute list is a superkey of R)
- commutativity does not hold for the PROJECT operation
- $\pi_{l_1}(\pi_{l_2}(R)) = \pi_{l_1}(R)$ when l_2 contains the attributes of l_1

Next Topic:

PROPERTIES of Selection, Projection Operations

1. Selection Operation is Commutative.

$$\sigma_{c_1}\left(\sigma_{c_2}(R)\right) \equiv \sigma_{c_2}\left(\sigma_{c_1}(R)\right)$$

$$\equiv \sigma_{c_1 \wedge c_2}(R)$$

Selection Splitting

- For any database relation R and predicates p, q, we have

$$\sigma_{p \wedge q}(R) = \sigma_p(\sigma_q(R))$$

- A corollary is that selection is commutative, that is,

$$\sigma_p(\sigma_q(R)) = \sigma_q(\sigma_p(R))$$

Notice that the SELECT operation is **commutative**; that is,

$$\sigma_{<cond1>}(\sigma_{<cond2>}(R)) = \sigma_{<cond2>}(\sigma_{<cond1>}(R))$$

Hence, a sequence of SELECTs can be applied in any order. In addition, we can always combine a **cascade** (or **sequence**) of SELECT operations into a single SELECT operation with a conjunctive (AND) condition; that is,

$$\sigma_{<cond1>}(\sigma_{<cond2>}(\ldots(\sigma_{<cond\,n>}(R))\ldots)) = \sigma_{<cond1>\ \mathbf{AND} <cond2>\ \mathbf{AND} \ldots \mathbf{AND}\ <cond\,n>}(R)$$

2. Projection Operation is Not Commutative.

$$\Pi_{L_1}(\Pi_{L_2}(R)) \neq \Pi_{L_2}(\Pi_{L_1}(R))$$

Projection Refinement

- For any subsets a and b of a database relation R such that $a \subseteq b$, we have

$$\pi_a(R) = \pi_a(\pi_b(R))$$

$$\pi_{<list1>}(\pi_{<list2>}(R)) = \pi_{<list1>}(R)$$

as long as <list2> contains the attributes in <list1>; otherwise, the left-hand side is an incorrect expression. It is also noteworthy that commutativity *does not* hold on PROJECT.

Q:

Can we Commute Selection, Projection Operation?

$$\pi_L(\sigma_c(R)) \overset{?}{\equiv} \sigma_c(\pi_L(R))$$

$$\pi_{C}\left(\sigma_{A=\alpha}(R)\right) :$$

C
1
5

$$\boxed{\sigma_{A=\alpha}\left(\pi_{C}(R)\right)}$$

Invalid

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

Q:

Can we Commute Selection, Projection Operation?

$$\pi_L(\sigma_c(R)) \overset{?}{=} \sigma_c(\pi_L(R))$$

No

$$\Pi_{A,B}\left(\sigma_{A=\alpha}(R)\right) :$$

A	B
α	α
α	β

$$\sigma_{A=\alpha}\left(\Pi_{A,B}(R)\right) :$$

A	B
α	α
α	β

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

Q:

Can we Commute Selection, Projection Operation?

No.

$$\pi_L(\sigma_C(R)) \neq \sigma_C(\pi_L(R))$$

Q:

Can we Commute Selection, Projection Operation? No.

If c uses attributes of L only:

$$\pi_L(\sigma_c(R)) = \sigma_c(\pi_L(R))$$

Note: If C only uses attributes of List L then

$$\sigma_C(\pi_L(R)) = \pi_L(\sigma_C(R))$$

$$\pi_{sname,rating}(\sigma_{rating>8}(S2))$$

L C

sid	*sname*	*rating*	*age*
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

Figure 4.2 Instance $S2$ of Sailors

Can we Commute (swap) π, σ ??

$$\pi_{sname,rating}(\sigma_{rating>8}(S2)) :$$

sid	sname	rating	age
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

Figure 4.2 Instance $S2$ of Sailors

sname	rating
Yuppy	9
Rusty	10

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

sname	rating
Yuppy	9
Rusty	10

Commutativity of Selection and Projection

- For any subset a of the attributes of a database relation R, and any predicate p that names only attributes in a, we have

$$\pi_a(\sigma_p(R)) = \sigma_p(\pi_a(R))$$

PROPERTIES Conclusion of Selection, Projection Operations:

transformation between selection and projection;

1. *Cascading of selection:* $\sigma_{C_1 \wedge C_2}(R) \equiv \sigma_{C_1}(\sigma_{C_2}(R))$
2. *Commutativity of selection:* $\sigma_{C_1}(\sigma_{C_2}(R)) \equiv \sigma_{C_2}(\sigma_{C_1}(R))$
3. *Cascading of projection:* $\pi_{Atts}(R) = \pi_{Atts}(\pi_{Atts'}(R))$ if $Atts \subseteq Atts'$
4. *Commutativity of projection:* $\pi_{Atts}(\sigma_C(R)) = \sigma_C(\pi_{Atts}((R))$ if $Atts$ includes all attributes used in C.

3.13.3 Relational Calculus: GATE CSE 1998 | Question: 2.19 top

https://gateoverflow.in/1692

Which of the following query transformations (i.e., replacing the l.h.s. expression by the r.h.s expression) is incorrect? R_1 and R_2 are relations, C_1 and C_2 are selection conditions and A_1 and A_2 are attributes of R_1.

tests.gatecse.in goclasses.in tests.gatecse.in

A. $\sigma_{C_1}(\sigma_{C_2}(R_1)) \rightarrow \sigma_{C_2}(\sigma_{C_1}(R_1))$

B. $\sigma_{C_1}(\pi_{A_1}(R_1)) \rightarrow \pi_{A_1}(\sigma_{C_1}(R_1))$

C. $\sigma_{C_1}(R_1 \cup R_2) \rightarrow \sigma_{C_1}(R_1) \cup \sigma_{C_1}(R_2)$

D. $\pi_{A_1}(\sigma_{C_1}(R_1)) \rightarrow \sigma_{C_1}(\pi_{A_1}(R_1))$

3.13.3 Relational Calculus: GATE CSE 1998 | Question: 2.19 top

https://gateoverflow.in/1692

Which of the following query transformations (i.e., replacing the l.h.s. expression by the r.h.s expression) is incorrect? R_1 and R_2 are relations, C_1 and C_2 are selection conditions and A_1 and A_2 are attributes of R_1.

A. $\sigma_{C_1}(\sigma_{C_2}(R_1)) \rightarrow \sigma_{C_2}(\sigma_{C_1}(R_1))$ Correct

B. $\sigma_{C_1}(\pi_{A_1}(R_1)) \rightarrow \pi_{A_1}(\sigma_{C_1}(R_1))$

C. $\sigma_{C_1}(R_1 \cup R_2) \rightarrow \sigma_{C_1}(R_1) \cup \sigma_{C_1}(R_2)$ → Proof

D. $\pi_{A_1}(\sigma_{C_1}(R_1)) \rightarrow \sigma_{C_1}(\pi_{A_1}(R_1))$ → WRONG

tests.gatecse.in goclasses.in tests.gatecse.in

Variation :

https://gateoverflow.in/1692

Which of the following query transformations (i.e., replacing the l.h.s. expression by the r.h.s expression) is [redacted] R_1 and R_2 are relations, C_1 and C_2 are selection conditions and A_1 and A_2 are attributes of R_1.

A. $\sigma_{C_1}(\sigma_{C_2}(R_1)) \rightarrow \sigma_{C_2}(\sigma_{C_1}(R_1))$

B. $\sigma_{C_1}(\pi_{A_1}(R_1)) \rightarrow \pi_{A_1}(\sigma_{C_1}(R_1))$

C. $\sigma_{C_1}(R_1 \cup R_2) \rightarrow \sigma_{C_1}(R_1) \cup \sigma_{C_1}(R_2)$

D. $\pi_{A_1}(\sigma_{C_1}(R_1)) \rightarrow \sigma_{C_1}(\pi_{A_1}(R_1))$

C_1 uses attributes of A_1

Given

Correct

Ans: A, B, C, D

(A) $\sigma_{c_1}(\sigma_{c_2}(R)) \rightleftarrows \sigma_{c_2}(\sigma_{c_1}(R))$

$\sigma_{c_1 \wedge c_2}(R) \equiv \sigma_{c_1 \wedge c_2}(R)$

$$\sigma_{c_1}(\pi_{A_1}(R)) \overset{?}{\equiv} \pi_{A_1}(\sigma_{c_1}(R))$$

No

(B) $\underbrace{\sigma_{C_1}(\pi_{A_1}(R))}$ $\longrightarrow$ $\pi_{A_1}(\sigma_{C_1}(R))$

Correct

If valid; then

C_1 uses attributes of A_1 only

(c)

$$\sigma_{c_1}(R_1 \cup R_2) \equiv \sigma_{c_1}(R_1) \cup \sigma_{c_1}(R_2)$$

t

t satisfies c_1 &

$t \in R_1$ or $t \in R_2$

t is present

c

$$\sigma_{c_1}(R_1 \cup R_2) \equiv \sigma_{c_1}(R_1) \cup \sigma_{c_1}(R_2)$$

t

t

t

If t present here

t

t satisfies c_1

$\forall\, t \in R_1$

$t \in \sigma_{c_1}(R_1)$ or

$t \in \sigma_{c_1}(R_2)$

(c)

$$\sigma_{c_1}(R_1 \cup R_2) \equiv \underbrace{\sigma_{c_1}(R_1) \cup \sigma_{c_1}(R_2)}$$

t present

If t present here

t

$t \in \sigma_{c_1}(R_1)$ or

$t \in \sigma_{c_1}(R_2)$

(c)

$$\sigma_{c_1}(R_1 \cup R_2) \equiv \sigma_{c_1}(R_1) \cup \sigma_{c_1}(R_2)$$

$$\pi_{A_1}(\sigma_{C_1}(R_1)) \overset{?}{\equiv} \sigma_{C_1}(\pi_{A_1}(R_1))$$

No

D

$$\boxed{\pi_{A_1}(\sigma_{c_1}(R_1))} \longrightarrow \sigma_{c_1}(\pi_{A_1}(R_1))$$

Assume Valid

WRONG

Invalid

$$\pi_{B,D}(\sigma_{A=0}(R_1)) \longrightarrow \boxed{\sigma_{A=0}(\pi_{B,D}(R_1))}$$

c_1

$R_1(ABCD)$

Next Topic:

Algebra Operators

3.

Set Operations

Relation (Table) : a Set of Tuples

↳ So set opⁿ can be applied

Set opⁿ :

Basic

$\cup$, $-$, $\times$

$\cap$

↳ Derived

Set opⁿ:

$$R \cup S = \{ x \mid x \in R \text{ or } x \in S \}$$

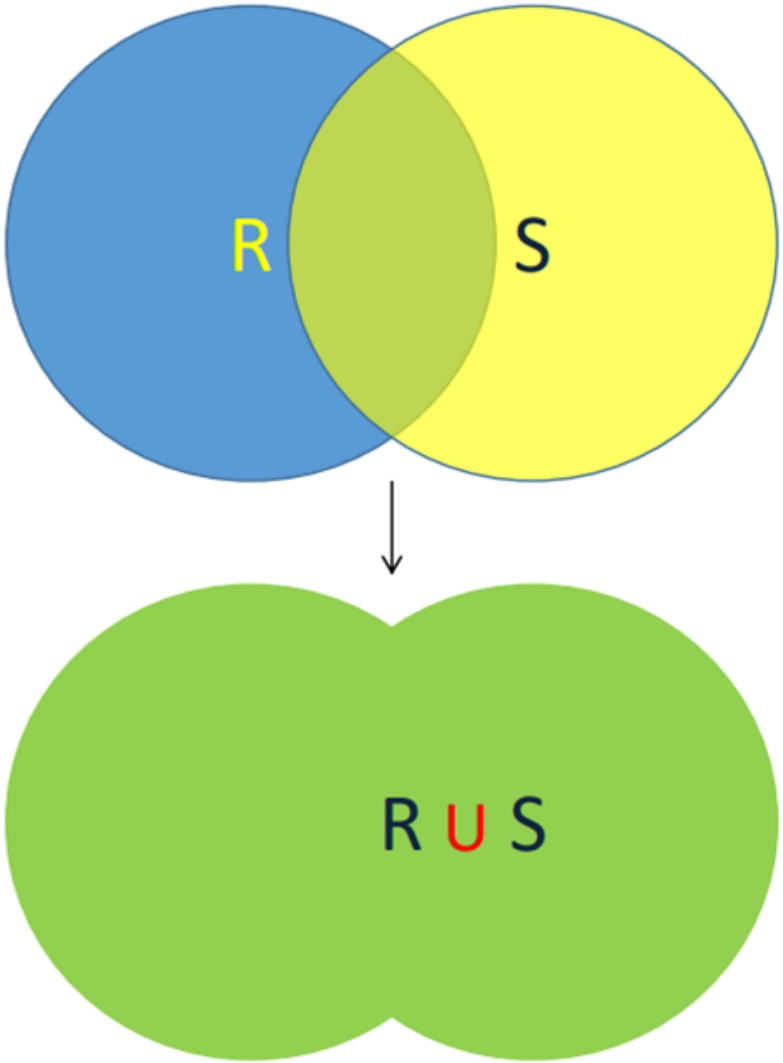
$$R \cap S = \{ x \mid x \in R \text{ and } x \in S \}$$

$$R - S = \{ x \mid x \in R \text{ but } x \notin S \}$$

$$R \times S = \{ (x, y) \mid x \in R, y \in S \}$$

Union

T=R ∪ S



GO Classes

Union Operation – Example

Relations *r, s:*

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

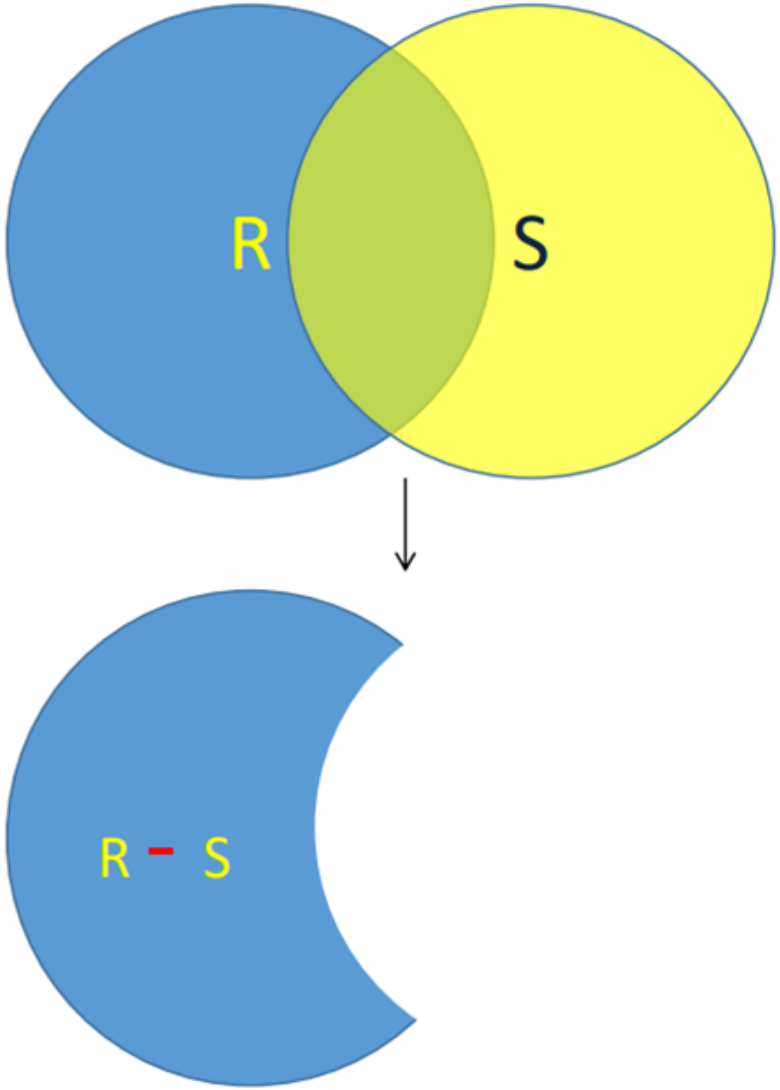
s

$r \cup s$:

A	B
α	1
α	2
β	1
β	3

Difference

R – S



GO Classes

Set difference of two relations

Relations *r*, *s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

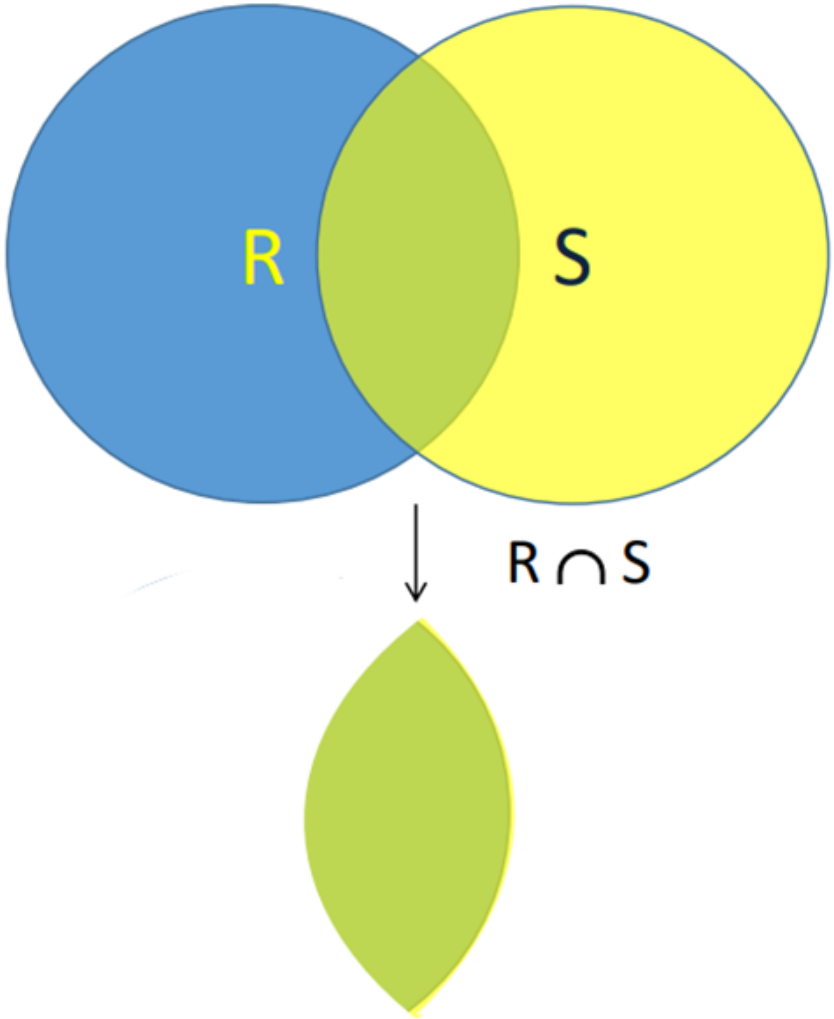
s

r – *s*:

A	B
α	1
β	1

Intersection

$T = R \cap S$



GO Classes

Set difference of two relations

Relations *r*, *s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

$r \cap s$

A	B
α	2

Cartessian Product (cross product) : X

R x S : Combine Every tuple of R with Every tuple of S

Cartesian-Product Operation – Example

GO Classes

Relations *r*, *s*:

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

r x *s*:

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

NOTE:

To Apply Union, Intersection, Set Difference Operations; the Relations Must be Union-Compatible.

Union-Compatible:

$R(a_1, a_2, --, a_n)$; $S(b_1, b_2, --, b_m)$

R, S are Union Compatible iff

1. $n = m$ &

2. $Domain(a_i) = Domain(b_i)$; $\forall_i$

- The 3 set-theoretic operations, UNION, INTERSECTION & SET DIFFERENCE are binary
- To ensure the applicability of the 3 set-theoretic operations, the Union Compatibility condition must hold
- Union Compatibility:

$$R(A_1, A_2, \ldots, A_n), S(B_1, B_2, \ldots, B_n)$$

same degree n and $dom(A_i) = dom(B_i), i = 1, \ldots, n$.

4.2.2 Set Operations

The following standard operations on sets are also available in relational algebra: *union* ($\cup$), *intersection* ($\cap$), *set-difference* ($-$), and *cross-product* ($\times$).

- **Union:** $R \cup S$ returns a relation instance containing all tuples that occur in *either* relation instance R or relation instance S (or both). R and S must be *union-compatible*, and the schema of the result is defined to be identical to the schema of R.

  Two relation instances are said to be **union-compatible** if the following conditions hold:
  - they have the same number of the fields, and
  - corresponding fields, taken in order from left to right, have the same *domains*.

Note that field names are Not used in defining union-compatibility.

www.goclasses.in

Several set theoretic operations are used to merge the elements of two sets in various ways, including **UNION**, **INTERSECTION**, and **SET DIFFERENCE** (also called **MINUS** or **EXCEPT**). These are **binary** operations; that is, each is applied to two sets (of tuples). When these operations are adapted to relational databases, the two relations on which any of these three operations are applied must have the same **type of tuples**; this condition has been called *union compatibility* or *type compatibility*. Two relations $R(A_1, A_2, \ldots, A_n)$ and $S(B_1, B_2, \ldots, B_n)$ are said to be **union compatible** (or **type compatible**) if they have the same degree n and if $\text{dom}(A_i) = \text{dom}(B_i)$ for $1 \leq i \leq n$. This means that the two relations have the same number of attributes and each corresponding pair of attributes has the same domain.

$R \cup S$ ≡ Result schema will be same as schema of LHS (R)

Union-Compatible

(Same for $R \cap S$, $R - S$)

Union: Example

∪ = union

Checking-account ∪ Savings-account

Checking-account	c-num	c-owner	c-balance
	101	J. Smith	1000.00
	102	W. Wei	2000.00
	104	M. Jones	1000.00
	105	H. Martin	10,000.00

Savings-account	s-num	s-owner	s-balance
	103	J. Smith	5000.00

	c-num	c-owner	c-balance
	101	J. Smith	1000.00
	102	W. Wei	2000.00
	104	M. Jones	1000.00
	105	H. Martin	10,000.00
	103	J. Smith	5000.00

Union Compatible

- Two relations are *union-compatible* if they have the same degree (i.e., the same number of attributes) and the corresponding attributes are defined on the same domains.
- Suppose we have these tables:

Checking-Account (c-num:str, c-owner:str, c-balance:real)

Savings-Account (s-num:str, s-owner:str, s-balance:real)

These are union-compatible tables.

- *Union, intersection, & difference require union-compatible tables*

We can define the three operations UNION, INTERSECTION, and SET DIFFERENCE on two union-compatible relations R and S as follows:

- UNION: The result of this operation, denoted by $R \cup S$, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.
- INTERSECTION: The result of this operation, denoted by $R \cap S$, is a relation that includes all tuples that are in both R and S.
- SET DIFFERENCE (or MINUS): The result of this operation, denoted by $R - S$, is a relation that includes all tuples that are in R but not in S.

Intersection

$\cap$ = intersection

Checking-account $\cap$ Savings-account

What's the answer to this query?

Checking-account

c-num	c-owner	c-balance
101	J. Smith	1000.00
102	W. Wei	2000.00
104	M. Jones	1000.00
105	H. Martin	10,000.00

Savings-account

s-num	s-owner	s-balance
103	J. Smith	5000.00

Intersection (cont.)

Checking-account

c-num	c-owner	c-balance
101	J. Smith	1000.00
102	W. Wei	2000.00
104	M. Jones	1000.00
105	H. Martin	10,000.00

Savings-account

s-num	s-owner	s-balance
103	J. Smith	5000.00

Checking-account ∩ Savings-account

What's the answer to this query?

It's empty. There are no tuples that are in both tables.

$(\pi_{\text{c-owner}} \text{Checking-account}) \cap (\pi_{\text{s-owner}} \text{Savings-account})$

What's the answer to this new query?

Intersection (cont.)

Checking-account

c-num	c-owner	c-balance
101	J. Smith	1000.00
102	W. Wei	2000.00
104	M. Jones	1000.00
105	H. Martin	10,000.00

Savings-account

s-num	s-owner	s-balance
103	J. Smith	5000.00

Checking-account ∩ Savings-account

What's the answer to this query?

It's empty. There are no tuples that are in both tables.

$(\pi_{\text{c-owner}} \text{Checking-account}) \cap (\pi_{\text{s-owner}} \text{Savings-account})$

What's the answer to this new query?

	c-owner
	J. Smith

Difference

— = difference

Checking-account

c-num	c-owner	c-balance
101	J. Smith	1000.00
102	W. Wei	2000.00
104	M. Jones	1000.00
105	H. Martin	10,000.00

Savings-account

s-num	s-owner	s-balance
103	J. Smith	5000.00

Find all the customers that own a Checking-account and do not own a Savings-account.

$(\pi_{\text{c-owner}} \text{Checking-account}) - (\pi_{\text{s-owner}} \text{Savings-account})$

- What is the *schema* of result?

c-owner
W. Wei
M. Jones
H. Martin

Set Operations on Relations

R $\cup$ **S**, the **union** of **R** and **S**, is the set of tuples that are in **R** or **S** or both.

R – **S**, the **difference** of **R** and **S**, is the set of tuples that are in **R** but not in **S**.

Note that **R** – **S** is different from **S** – **R**.

R $\cap$ **S**, the **intersection** of **R** and **S**, is the set of tuples that are in both **R** and **S**.

- **Intersection:** $R \cap S$ returns a relation instance containing all tuples that occur in *both* R and S. The relations R and S must be union-compatible, and the schema of the result is defined to be identical to the schema of R.

- **Set-difference:** $R - S$ returns a relation instance containing all tuples that occur in R but not in S. The relations R and S must be union-compatible, and the schema of the result is defined to be identical to the schema of R.

- **Cross-product:** $R \times S$ returns a relation instance whose schema contains all the fields of R (in the same order as they appear in R) followed by all the fields of S (in the same order as they appear in S). The result of $R \times S$ contains one tuple $\langle r, s \rangle$ (the concatenation of tuples r and s) for each pair of tuples $r \in R$, $s \in S$. The cross-product opertion is sometimes called **Cartesian product**.

  We will use the convention that the fields of $R \times S$ inherit names from the corresponding fields of R and S. It is possible for both R and S to contain one or more fields having the same name; this situation creates a *naming conflict*. The corresponding fields in $R \times S$ are unnamed and are referred to solely by position.

www.goclasses.in

Union Operation

- Notation: $r \cup s$
- Defined as:

  $$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$

- For $r \cup s$ to be valid.

  1. *r, s* must have the *same* **arity** (same number of attributes)
  2. The attribute domains must be **compatible** (example: 2nd column of *r* deals with the same type of values as does the 2nd column of *s*)

- Example: to find all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or in both

$$\Pi_{course_id}\,(\sigma_{semester=\text{“Fall”} \wedge year=2009}(section)) \cup$$

$$\Pi_{course_id}\,(\sigma_{semester=\text{“Spring”} \wedge year=2010}(section))$$

5 Union ($\cup$)

The first way we will learn how to combine data from different relations is with the **union operator**. Just like the UNION clause in SQL, we take all the rows from each tuple and combine them removing duplicates along the way. As an example, say we have a dogs table:

name	**age**
Scooby	10
Buster	15
Garfield	20

and a cats table that looks like this:

name	**age**
Tom	8
Garfield	10

The expression:

$$\pi_{name}(dogs) \cup \pi_{name}(cats)$$

The expression:

$$\pi_{name}(dogs) \cup \pi_{name}(cats)$$

would return:

name
Scooby
Buster
Tom
Garfield

Note that Garfield only shows up once because relations are sets of tuples and remove all duplicates as a result. Additionally, note that one rule for all of these set operators is that they must operate on relations that have the same number of attributes (columns), and the attributes must have the same type. It would not be legal to union a relation with two columns with a relation that only has one column nor would it be legal to union a relation with a column of strings with another relation with one column of ints.

Set Difference Operation

- Notation $r - s$
- Defined as:

$$r - s = \{t \mid t \in r \textbf{ and } t \notin s\}$$

- Set differences must be taken between **compatible** relations.
  - r and s must have the same arity
  - attribute domains of r and s must be compatible

- Example: to find all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester

$$\Pi_{course_id}(\sigma_{semester=\text{“Fall”} \wedge year=2009}(section)) -$$

$$\Pi_{course_id}(\sigma_{semester=\text{“Spring”} \wedge year=2010}(section))$$

Union example.

Query: list names of all people in the department

Student		
Name	Country	GPA
Bob	Canada	3
John	Britain	3
Tom	Canada	3.5
Maria	Mexico	4

Professor	
Name	Rank
Dr. Monk	Professor
Dr. Pooh	Associate Professor
Dr. Patel	Assistant Professor

$\pi_{Name}(Student \cup Professor)$

Can we do ?

T=Student $\cup$ Professor

GO Classes

Union example.

Query: list names of all people in the department

NOT Union Compatible

Student		
Name	Country	GPA
Bob	Canada	3
John	Britain	3
Tom	Canada	3.5
Maria	Mexico	4

Professor	
Name	Rank
Dr. Monk	Professor
Dr. Pooh	Associate Professor
Dr. Patel	Assistant Professor

Invalid

$\pi_{Name}(Student \cup professor)$

Can we do ?

T=Student $\cup$ Professor

GO Classes

Union example.

Query: list names of all people in the department

Student
Name
Bob
John
Tom
Maria

Professor
Name
Dr. Monk
Dr. Pooh
Dr. Patel

$$T = \pi_{name}(\text{Student}) \cup \pi_{name}(\text{Professor})$$

Note: if attributes in 2 operands have different names, the names of the left relation are used in the union (PostgreSQL)

Derived

Q:

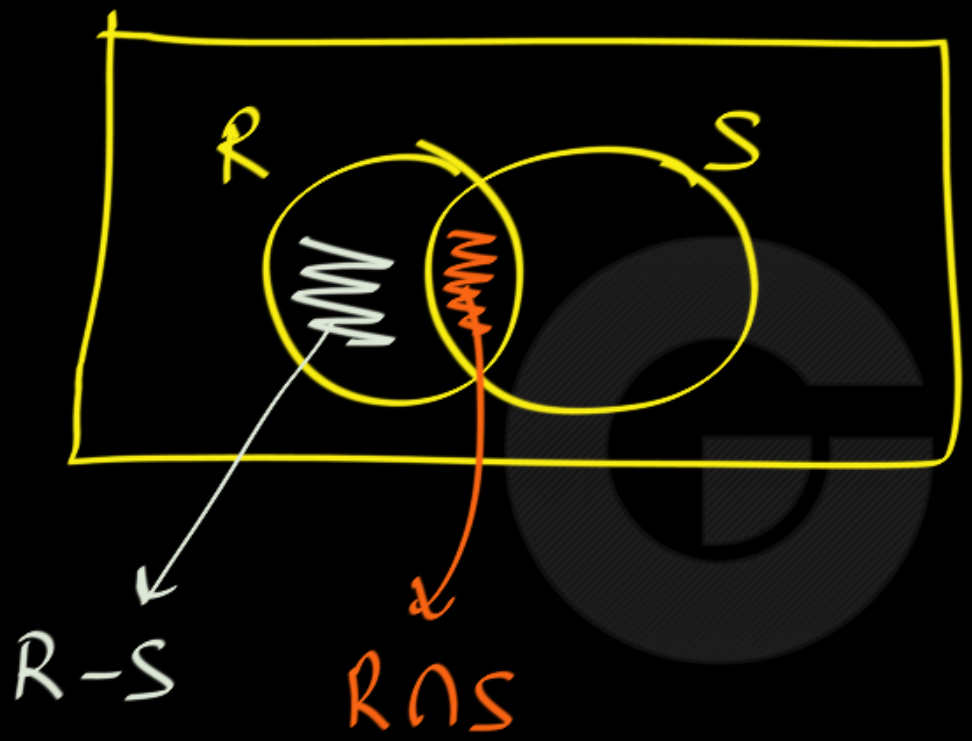
Intersection is Not Basic Operation in Relational Algebra.

Can we write Intersection in Terms of Union & Difference??

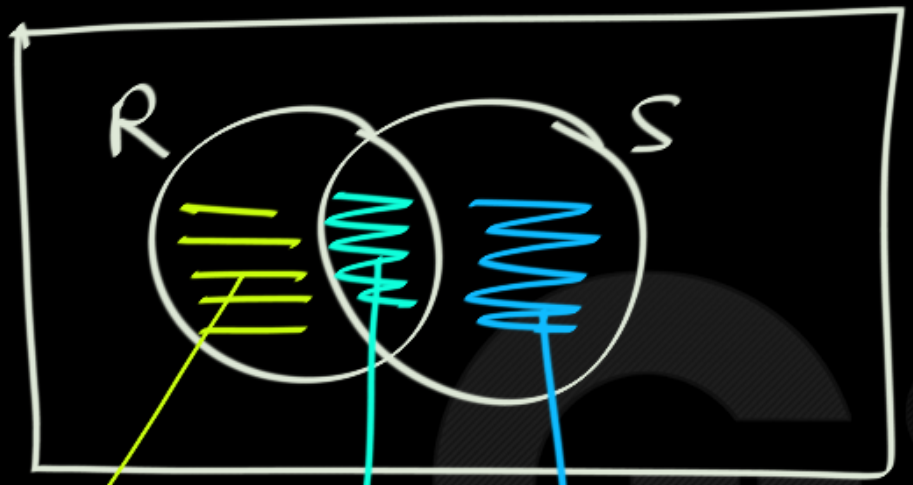
Challenge Question

- How could you express the intersection operation if you didn't have an Intersection operator in relational algebra? [Hint: Can you express Intersection using only the Difference operator?]

$A \cap B$ = ???

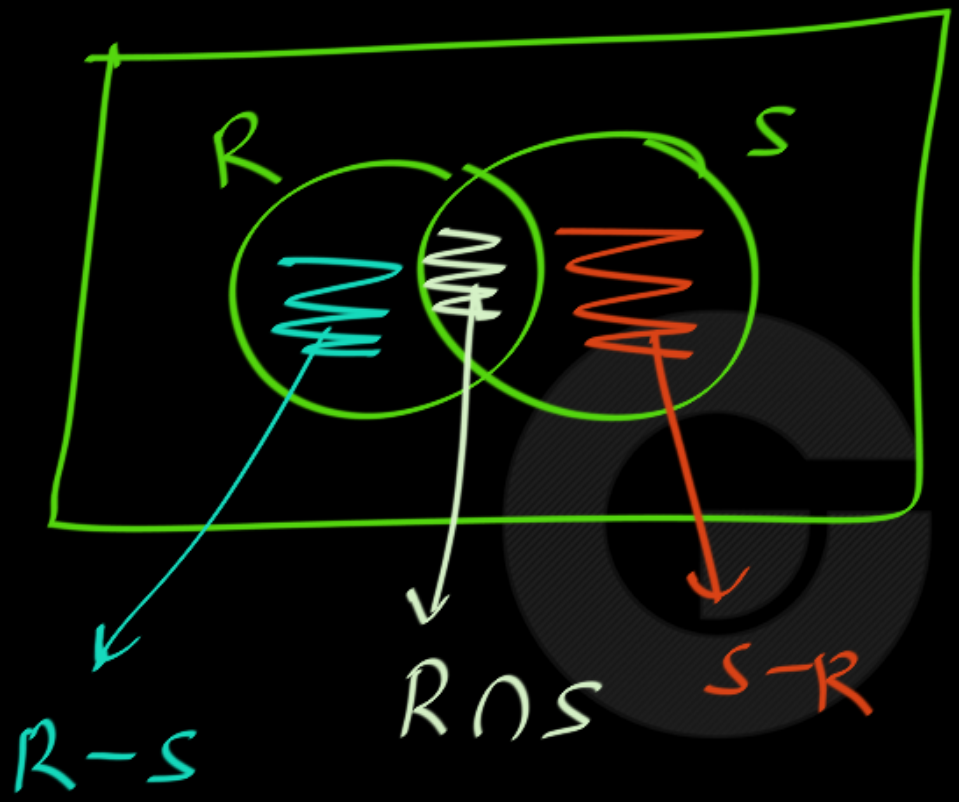


$$R \cap S = R - (R - S)$$



$R-S$ $\quad$ $R \cap S$ $\quad$ $S-R$

$$R \cap S = \boxed{(R \cup S) - (R-S)} - (S-R)$$



$$R \cap S = (R \cup S) - \left[(R-S) \cup (S-R) \right]$$

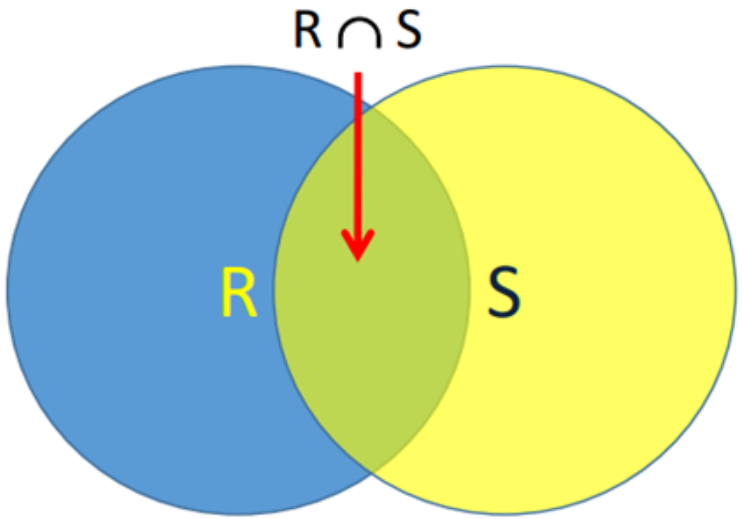
① $R \cap S = S - (S - R)$

② $R \cap S = R - (R - S)$

③ $R \cap S = (R \cup S) - [(R - S) \cup (S - R)]$

④ $R \cap S = [(R \cup S) - (R - S)] - (S - R)$

Intersection is a shortcut for R – (R – S)



R ∩ S can be derived using 2 difference operators
R – (R – S)

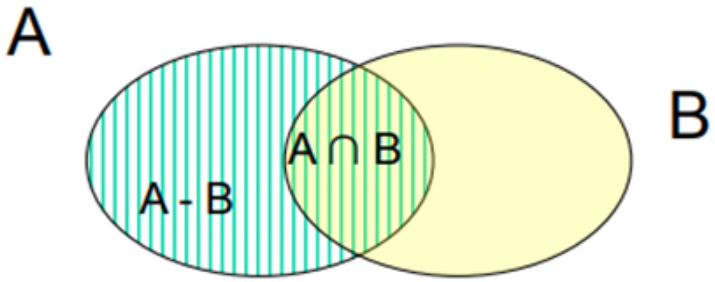
R - S (are in R but not in S)

R - (R-S)

Challenge Question

- How could you express the intersection operation if you didn't have an Intersection operator in relational algebra? [Hint: Can you express Intersection using only the Difference operator?]

$A \cap B = A - (A - B)$



Cross Product:

$R(a_1, a_2, \ldots, a_n)$; $S(b_1, b_2, \ldots, b_m)$

$R \times S(a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m)$

$\text{Degree}(R \times S) = \text{Deg}(R) + \text{Deg}(S)$

$|R \times S| = |R| * |S| = |R||S|$

Product

- R3 := R1 X R2
  - Pair each tuple t1 of R1 with each tuple t2 of R2.
  - Concatenation t1t2 is a tuple of R3.
  - Schema of R3 is the attributes of R1 and then R2, in order.
  - But beware attribute *A* of the same name in R1 and R2: use R1.*A* and R2.*A*.

Example: R3 := R1 X R2

R1(

A,	B
1	2
3	4

)

R2(

B,	C
5	6
7	8
9	10

)

R3(

A,	R1.B,	R2.B,	C
1	2	5	6
1	2	7	8
1	2	9	10
3	4	5	6
3	4	7	8
3	4	9	10

)

Cross product

- Input: two tables R and S
- Natation: $R \times S$
- Purpose: pairs rows from two tables
- Output: for each row r in R and each s in S, output a row rs (concatenation of r and s)

Cross product example

$User \times Member$

uid	name	age	pop
123	Milhouse	10	0.2
857	Lisa	8	0.7
…	…	…	…

uid	gid
123	gov
857	abc
857	gov
…	…

$\times$

[redacted]	name	age	pop	[redacted]	gid
123	Milhouse	10	0.2	123	gov
123	Milhouse	10	0.2	857	abc
123	Milhouse	10	0.2	857	gov
857	Lisa	8	0.7	123	gov
857	Lisa	8	0.7	857	abc
857	Lisa	8	0.7	857	gov
…	…	…	…	…	…

user.uid

member.uid

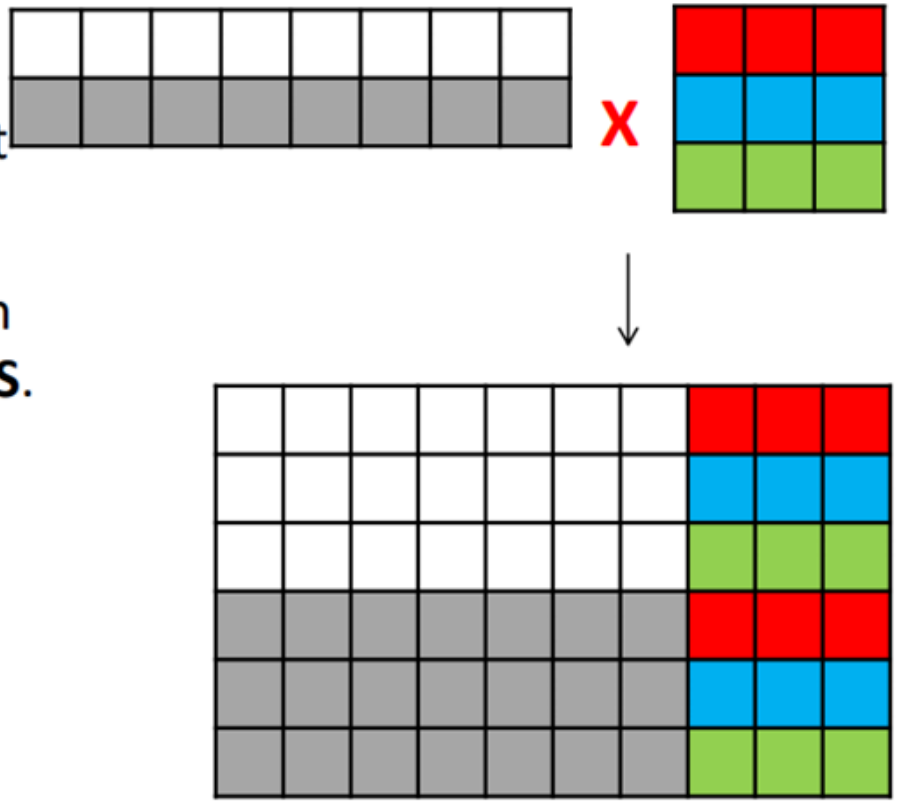
GO Classes

operation: Cartesian product (Cross-product)

1. Set of tuples ***rs*** that are formed by choosing the first part (***r***) to be any tuple of **R** and the second part (***s***) to be any tuple of **S**.

2.Schema for the resulting relation is the union of schemas for **R** and **S**.

3.If **R** and **S** happen to have some attributes in common, then prefix those attributes by the relation name.



T=R x S

Cartesian product example

T=Course x Professor

Course	
Topic	Year
Algorithms	2
Python	2
Databases	3
GUI	3

Professor	
Name	Rank
Dr. Monk	Professor
Dr. Pooh	Associate Professor
Dr. Patel	Assistant Professor

Cartesian product output

Algorithms	2
Python	2
Databases	3
GUI	3

Dr. Monk	Dr. Pooh	Dr. Patel
Professor	Associate Professor	Assistant Professor

Topic	Y	Name	Rank
Algorithms	2	Dr. Monk	Professor
Algorithms	2	Dr. Pooh	Assoc. Professor
Algorithms	2	Dr. Patel	Assist. Professor
Python	2	Dr. Monk	Professor
Python	2	Dr. Pooh	Assoc. Professor
Python	2	Dr. Patel	Assist. Professor
Databases	3	Dr. Monk	Professor
Databases	3	Dr. Pooh	Assoc. Professor
Databases	3	Dr. Patel	Assist. Professor
GUI	3	Dr. Monk	Professor
GUI	3	Dr. Pooh	Assoc. Professor
GUI	3	Dr. Patel	Assist. Professor

GO Classes

CARTESIAN PRODUCT

- (alias CROSS PRODUCT or CROSS JOIN)
- binary operation, does not require Union Compatibility
- $R(A_1, A_2, \ldots, A_n) \times S(B_1, B_2, \ldots, B_m) \Longrightarrow Q(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$
- degree(Q) = degree(R) + degree(S)
- # tuples in Q = (# tuples in R) $\star$ (# tuples in S)

8.2.2 The CARTESIAN PRODUCT (CROSS PRODUCT) Operation

Next, we discuss the **CARTESIAN PRODUCT** operation—also known as **CROSS PRODUCT** or **CROSS JOIN**–which is denoted by $\times$. This is also a binary set operation, but the relations on which it is applied do *not* have to be union compatible. In its binary form, this set operation produces a new element by combining every member (tuple) from one relation (set) with every member (tuple) from the other relation (set). In general, the result of $R(A_1, A_2, \ldots, A_n) \times S(B_1, B_2, \ldots, B_m)$ is a relation Q with degree $n + m$ attributes $Q(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$, in that order. The resulting relation Q has one tuple for each combination of tuples—one from R and one from S. Hence, if R has n_R tuples (denoted as $|R| = n_R$), and S has n_S tuples, then $R \times S$ will have $n_R * n_S$ tuples.

The n-ary CARTESIAN PRODUCT operation is an extension of the above concept, which produces new tuples by concatenating all possible combinations of tuples from n underlying relations.

www.goclasses.in

Note: Assume only Data is Concerned

Commutativity:

Commutative

1. $R \cup S = S \cup R$ ✓
2. $R \cap S = S \cap R$ ✓
3. $R \times S = S \times R$ ✓
4. $R - S \neq S - R$ ← NOT Commutative

Note:

Eg:

R

A
1
2

S

A
1
3

R − S =

A
2

≠

S − R =

A
3

Note: If only Data is Concerned

Associativity:

Associative

1. $(R \cup S) \cup T = R \cup (S \cup T)$
2. $(R \cap S) \cap T = R \cap (S \cap T)$
3. $(R \times S) \times T = R \times (S \times T)$

NOT Associative

4. $(R - S) - T \neq R - (S - T)$

Notice that both UNION and INTERSECTION are *commutative operations*; that is,

$$R \cup S = S \cup R \quad \text{and} \quad R \cap S = S \cap R$$

Both UNION and INTERSECTION can be treated as *n*-ary operations applicable to any number of relations because both are also *associative operations;* that is,

$$R \cup (S \cup T) = (R \cup S) \cup T \quad \text{and} \quad (R \cap S) \cap T = R \cap (S \cap T)$$

The MINUS operation is *not commutative;* that is, in general,

$$R - S \neq S - R$$

Note that INTERSECTION can be expressed in terms of union and set difference as follows:

$$R \cap S = ((R \cup S) - (R - S)) - (S - R)$$

A note a column ordering

GO Classes

- Recall: Ordering of columns is unimportant as far as contents are concerned

uid	name	age	pop	uid	gid
123	Milhouse	10	0.2	123	gov
123	Milhouse	10	0.2	857	abc
123	Milhouse	10	0.2	857	gov
857	Lisa	8	0.7	123	gov
857	Lisa	8	0.7	857	abc
857	Lisa	8	0.7	857	gov
…	…	…	…	…	…

=

uid	gid	uid	name	age	pop
123	gov	123	Milhouse	10	0.2
857	abc	123	Milhouse	10	0.2
857	gov	123	Milhouse	10	0.2
123	gov	857	Lisa	8	0.7
857	abc	857	Lisa	8	0.7
857	gov	857	Lisa	8	0.7
…	…	…	…	…	…

- So cross product is commutative, i.e., for any R and S, $R \times S = S \times R$ (up to the ordering of columns)

Next Topic:

Algebra Operators

4. Rename Operation

5. Assignment Operation

$S = \boxed{}$

Rename opⁿ: Symbol: ρ → Roh

Rename Table Name or Attributes or both

① $\rho_S(R)$

② $\rho_{(B_1, B_2)}(R)$

③ $\rho_{S(B_1, B_2)}(R)$

$R(A, B)$

$\rho_S(R)$: Renaming Table R to S

$\rho_{(E,F)} R$: Renaming attributes $A \rightarrow E$, $B \rightarrow F$

$\rho_{S(E,F)} R$: Renaming both

$R(A, B)$

$$\rho_{(A \to E)}(R) \equiv$$

R

E	B

Renaming

- Input: a table R and S
- Notation: $\rho_S\ R$, $\rho_{(A_1,A_2,\ldots)}R$, or $\rho_{S(A_1,A_2,\ldots)}R$
- Purpose: "rename" a table and/or its columns
- Output: a table with the same rows as R, but called differently
- Used to
  - Avoid confusion caused by identical column names

RENAME operator

- The RENAME operator is denoted by ρ
- We may *rename* the attributes of a relation or the relation name or both
  - $\rho_{S(B1, B2, \ldots, Bn)}(R)$
    - Change the relation name from *R* to *S*, *and*
    - Change column (attribute) names to B1, B2, …..Bn
  - $\rho_S(R)$:
    - Change the *relation name* only to S
  - $\rho_{(B1, B2, \ldots, Bn)}(R)$:
    - Change the *column (attribute) names* only to B1, B2, …..Bn

general syntax of the RENAME operation:

suppose we have a relation R of degree n ,

$$R(A_1, A_2, \ldots, A_n)$$

we can rename the relation, the attributes or both:

- $\rho_{S(B_1,B_2,\ldots,B_n)}(R)$
- $\rho_S(R)$
- $\rho_{(B_1,B_2,\ldots,B_n)}(R)$

Rename Operation

If a relational-algebra expression Y has arity n, then

$$\rho_{X(A\text{->}A1,\ B\text{->}A2,\ \ldots)}\ Y$$

returns the result of expression Y under the name X, and with the attributes renamed to $A1, A2, \ldots, An$.

For example,

$$\rho_{\ myRelation(A\text{->}E,\ B\text{->}K)}\ (r - s)$$

Take the set difference of r and s, and call the result myRelation, while renaming the first field to E, and the second field to K.

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

E	K
α	1
β	1

myRelation

Renaming Operator

$\rho_{S(A1,A2,\ldots,An)}$ **(R)**

1. Resulting relation has exactly the same tuples as **R**, but the name of the relation is **S**.
2. Moreover, the attributes of the result relation **S** can be re-named A_1, A_2, …, A_n, in order from the left.
3. If not all attributes are renamed, can specify renamed attributes:

$\rho_{S,\, a \rightarrow a1,\, b \rightarrow b1}$ **(R)**

D Classes

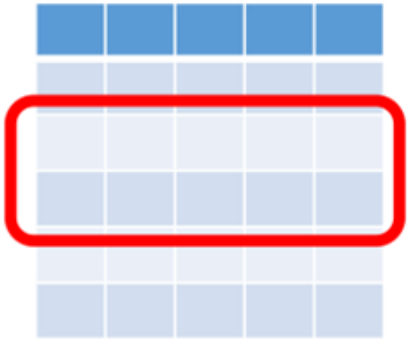
If no renaming is applied, the names of the attributes in the resulting relation of a SELECT operation are the same as those in the original relation and in the same order. For a PROJECT operation with no renaming, the resulting relation has the same attribute names as those in the projection list and in the same order in which they appear in the list.

We can also define a formal **RENAME** operation—which can rename either the relation name or the attribute names, or both—as a unary operator. The general RENAME operation when applied to a relation R of degree n is denoted by any of the following three forms:

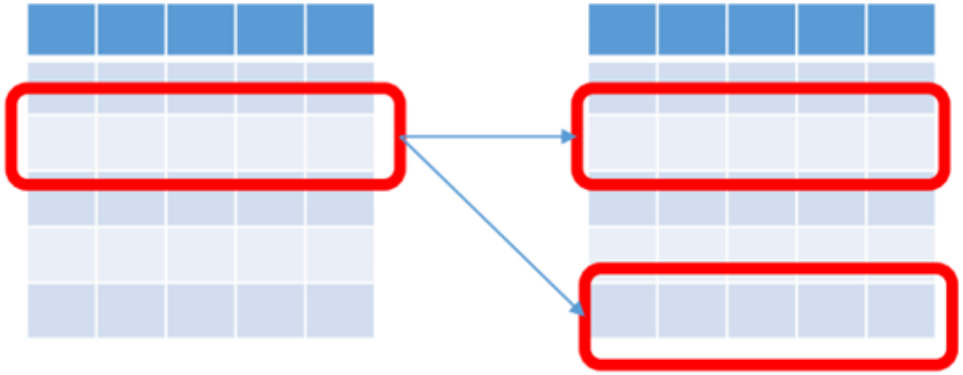
$$\rho_{S(B1,\,B2,\,\ldots,\,Bn)}(R) \quad \text{or} \quad \rho_S(R) \quad \text{or} \quad \rho_{(B1,\,B2,\,\ldots,\,Bn)}(R)$$

where the symbol ρ (rho) is used to denote the RENAME operator, S is the new relation name, and $B_1, B_2, \ldots, B_n$ are the new attribute names. The first expression renames both the relation and its attributes, the second renames the relation only, and the third renames the attributes only. If the attributes of R are $(A_1, A_2, \ldots, A_n)$ in that order, then each A_i is renamed as B_i.

Core operators of relational algebra



Selection σ



Cross-product x

Projection π

Union U
Difference –
Renaming ρ