Week-6

Introduction to Database Systems

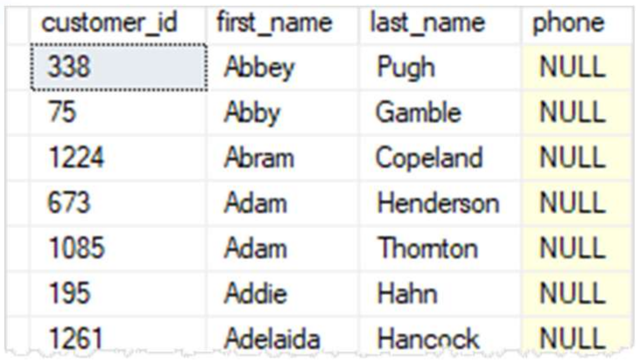
Shah Nawaz

Introduction to SQL (DML), IS/IS NOT , DISTINCT, ORDER BY, BETWEEN, ARITHMETIC OPERATIONS , LIKE ('%’, ‘_’), Relational Algebra

Operator: IS/IS NOT

- Query: show employees who do not have any phone number

```
Select *
From customer
Where  phone IS NULL ;
```

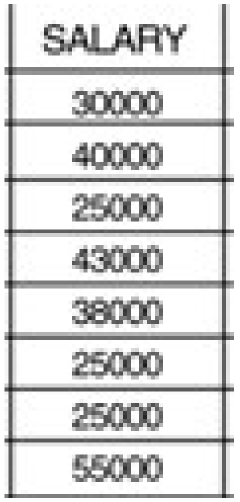


customer_id	first_name	last_name	phone
338	Abbey	Pugh	NULL
75	Abby	Gamble	NULL
1224	Abram	Copeland	NULL
673	Adam	Henderson	NULL
1085	Adam	Thornton	NULL
195	Addie	Hahn	NULL
1261	Adelaida	Hancock	NULL

DISTINCT

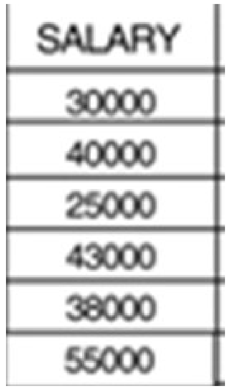
To eliminate duplicate tuples in a query result, the keyword DISTINCT is used

Select Salary
from employee;

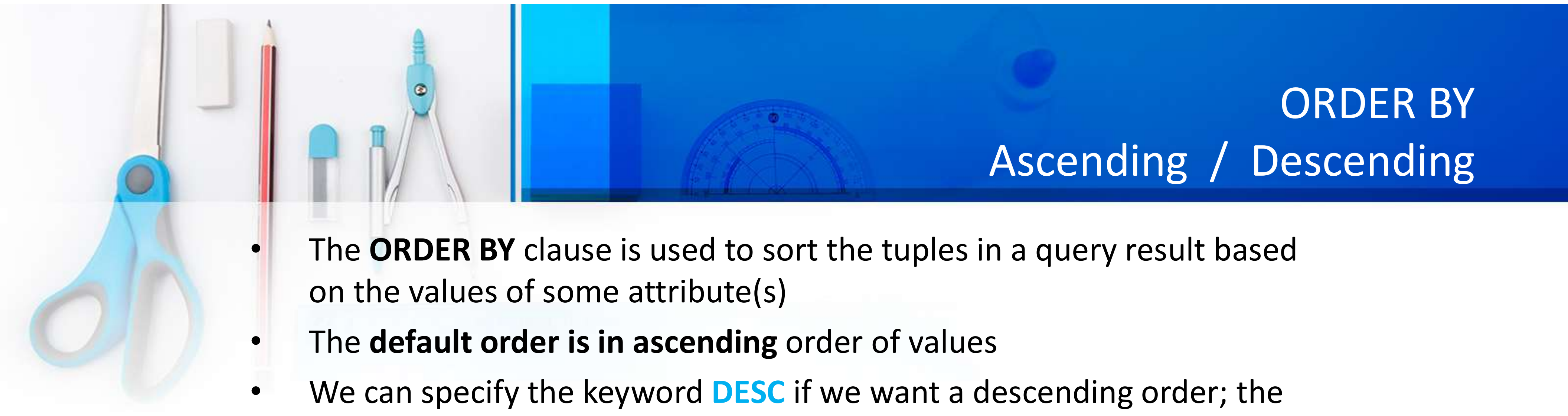


SALARY
30000
40000
25000
43000
38000
25000
25000
55000

Select distinct salary
from employee;



SALARY
30000
40000
25000
43000
38000
55000



ORDER BY
Ascending / Descending

- The **ORDER BY** clause is used to sort the tuples in a query result based on the values of some attribute(s)
- The **default order is in ascending** order of values
- We can specify the keyword **DESC** if we want a descending order; the keyword **ASC** can be used to explicitly specify ascending order, even though it is the default

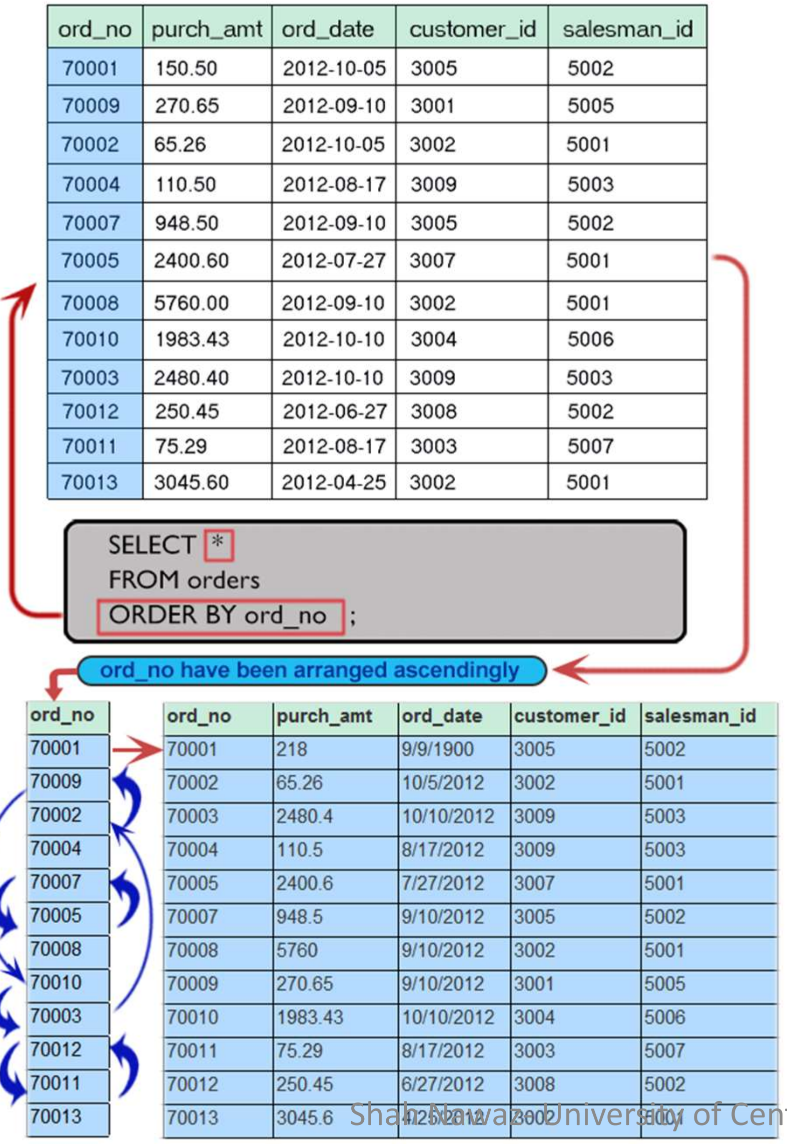
- Show all employee names in ascending order of their names.

```
Select Fname , Minit, Lname
From employee
Order by Fname;
```

- Show all employee names in descending order of their names.

```
Select Fname , Minit, Lname
From employee
Order by Fname desc;
```

ORDER BY
Ascending / Descending



ord_no	purch_amt	ord_date	customer_id	salesman_id
70001	150.50	2012-10-05	3005	5002
70009	270.65	2012-09-10	3001	5005
70002	65.26	2012-10-05	3002	5001
70004	110.50	2012-08-17	3009	5003
70007	948.50	2012-09-10	3005	5002
70005	2400.60	2012-07-27	3007	5001
70008	5760.00	2012-09-10	3002	5001
70010	1983.43	2012-10-10	3004	5006
70003	2480.40	2012-10-10	3009	5003
70012	250.45	2012-06-27	3008	5002
70011	75.29	2012-08-17	3003	5007
70013	3045.60	2012-04-25	3002	5001

```
SELECT *
FROM orders
ORDER BY ord_no ;
```

ord_no
70001
70009
70002
70004
70007
70005
70008
70010
70003
70012
70011
70013

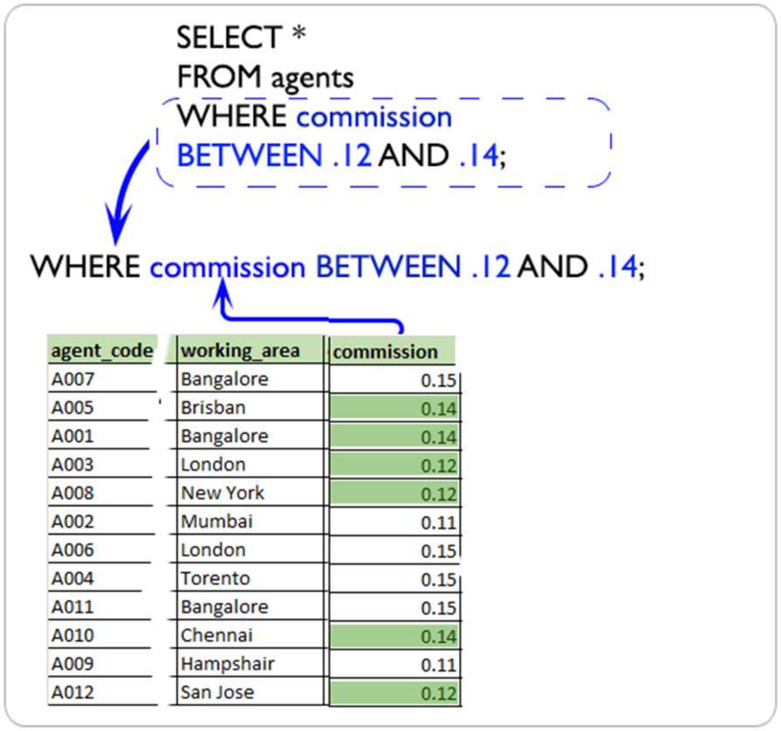
ord_no	purch_amt	ord_date	customer_id	salesman_id
70001	218	9/9/1900	3005	5002
70002	65.26	10/5/2012	3002	5001
70003	2480.4	10/10/2012	3009	5003
70004	110.5	8/17/2012	3009	5003
70005	2400.6	7/27/2012	3007	5001
70007	948.5	9/10/2012	3005	5002
70008	5760	9/10/2012	3002	5001
70009	270.65	9/10/2012	3001	5005
70010	1983.43	10/10/2012	3004	5006
70011	75.29	8/17/2012	3003	5007
70012	250.45	6/27/2012	3008	5002
70013	3045.6	4/25/2012	3002	5001

BETWEEN [Comparison Operator]

Retrieve all employees in department 5 whose salary is between 30,000 and 50,000

SELECT *
FROM EMPLOYEE
WHERE (Salary BETWEEN 30000 AND 50000)
AND DNO =5 ;

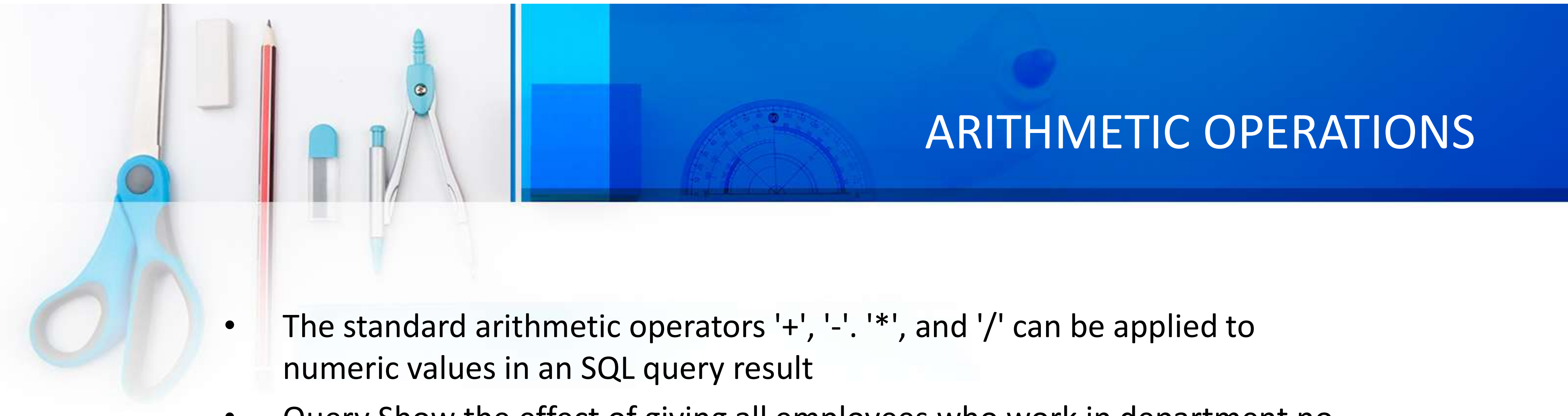
SAME AS
(Salary >= 30000)AND (Salary <= 50000)



```
SELECT *
FROM agents
WHERE commission
BETWEEN .12 AND .14;
```

WHERE commission BETWEEN .12 AND .14;

agent_code	working_area	commission
A007	Bangalore	0.15
A005	Brisban	0.14
A001	Bangalore	0.14
A003	London	0.12
A008	New York	0.12
A002	Mumbai	0.11
A006	London	0.15
A004	Torento	0.15
A011	Bangalore	0.15
A010	Chennai	0.14
A009	Hampshair	0.11
A012	San Jose	0.12



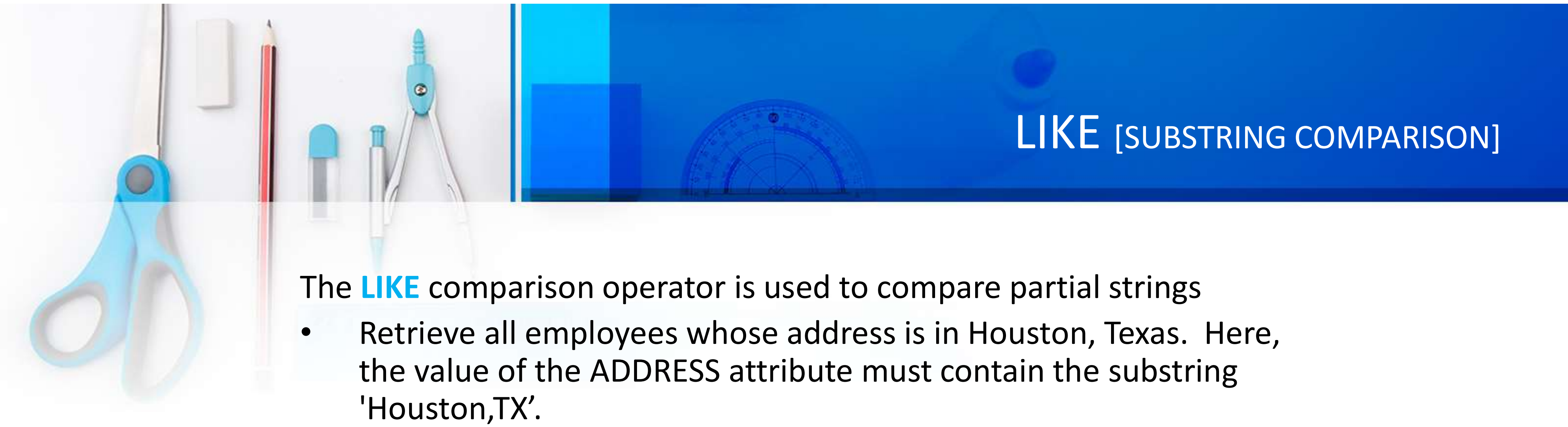
ARITHMETIC OPERATIONS

- The standard arithmetic operators '+', '-'. '*', and '/' can be applied to numeric values in an SQL query result
- Query Show the effect of giving all employees who work in department no 5 a 10% raise.

```
SELECT FNAME, LNAME, 1.1 * SALARY
FROM EMPLOYEE
WHERE dno =5;
```

LIKE [SUBSTRING COMPARISON]

- The **LIKE** comparison operator is used to compare partial strings
  - reserved characters used: '%' , '_'
  - % replaces an arbitrary number of characters i.e. 0 to n
  - '_' replaces a single arbitrary character i.e. only 1



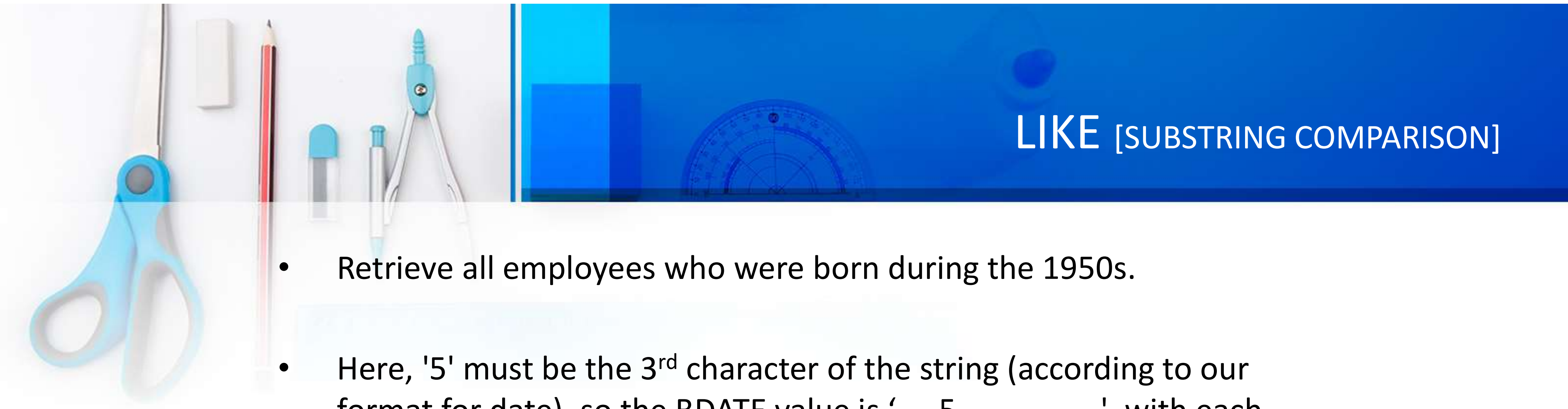
LIKE [SUBSTRING COMPARISON]

The LIKE comparison operator is used to compare partial strings

- Retrieve all employees whose address is in Houston, Texas. Here, the value of the ADDRESS attribute must contain the substring 'Houston,TX'.

```
SELECT    FNAME, LNAME
FROM      EMPLOYEE
WHERE     ADDRESS LIKE  '%Houston,TX%';
```

- reserved characters used: '%' , '_'
- % replaces an arbitrary number of characters i.e. 0 to n



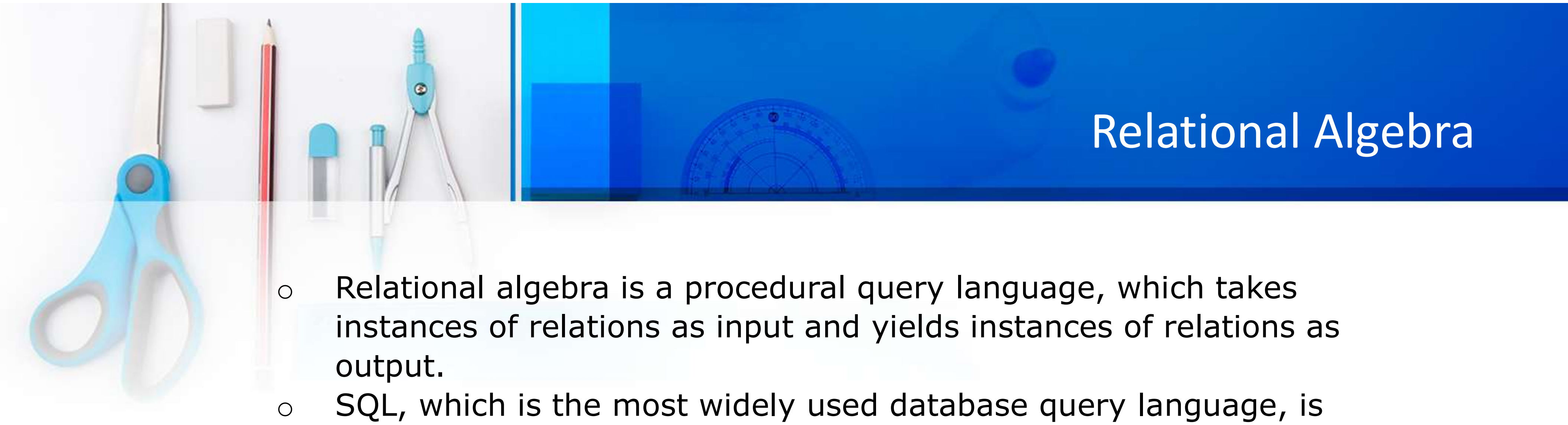
LIKE [SUBSTRING COMPARISON]

- Retrieve all employees who were born during the 1950s.
- Here, '5' must be the 3rd character of the string (according to our format for date), so the BDATE value is '_ _5_ _ _ _ _ _ _', with each underscore as a place holder for a single arbitrary character.

```
SELECT      FNAME, LNAME
FROM        EMPLOYEE
WHERE       BDATE LIKE '195 _ _ _ _ _ _ _ _'
```

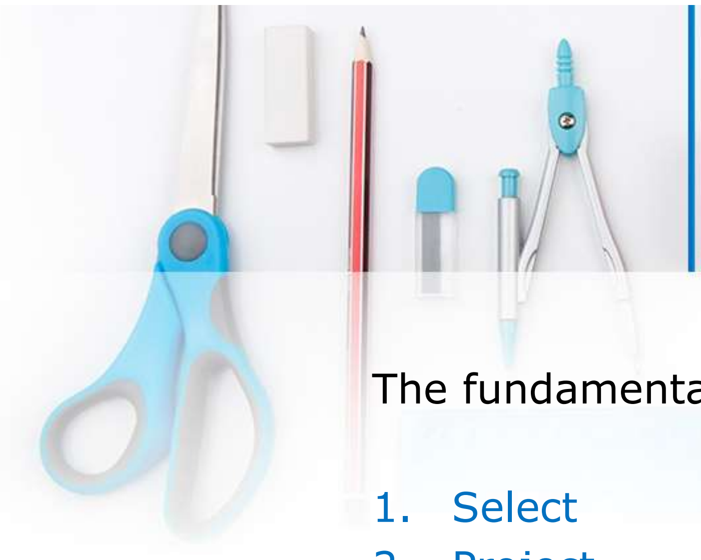
- '_' replaces a single arbitrary character i.e. only 1

Relational Algebra



Relational Algebra

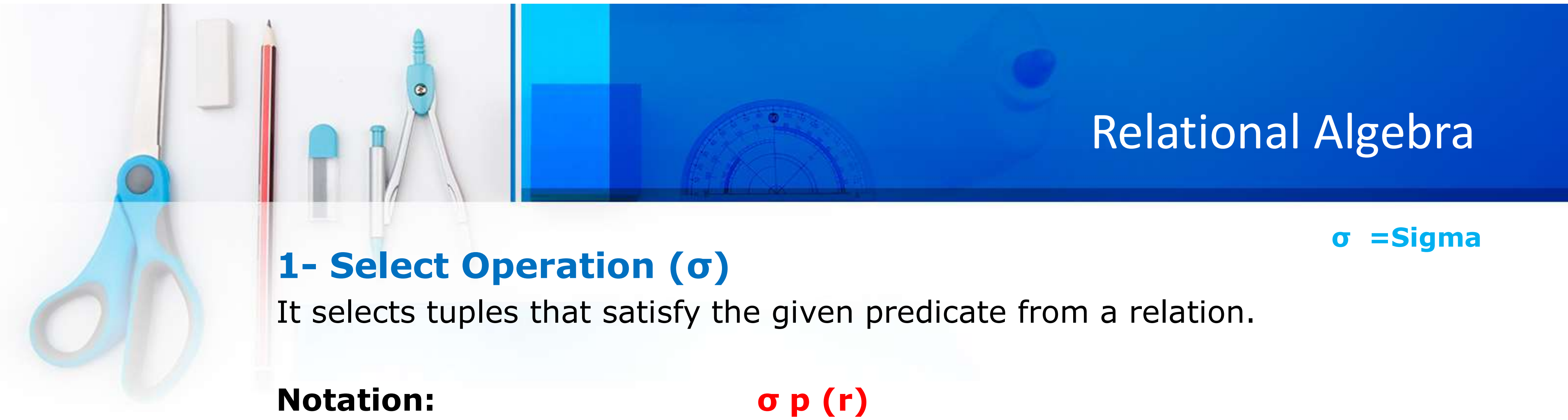
- Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output.
- SQL, which is the most widely used database query language, is based on the principles of relational algebra.
- It uses operators to perform queries.
- An operator can be either **unary** or **binary**.
  - **Unary operators:** Perform an action with a single operand. Unary operators use prefix notation, meaning that the operator precedes the operand. For example, ++ . --
  - **Binary operators:** Perform actions with two operands. For example, +, -, *, and /.
- They accept relations as their input and yield relations as their output.
- Relational algebra is performed recursively on a relation and intermediate results are also considered relations.



Relational Algebra

The fundamental operations of relational algebra are as follows:

1. Select
2. Project
   i. Union
   ii. Set different
   iii. Cartesian product
3. Rename



Relational Algebra

σ =Sigma

1- Select Operation (σ)

It selects tuples that satisfy the given predicate from a relation.

Notation: **σ p (r)**

Pronunciation: "The selection operation on relation r with the condition p."

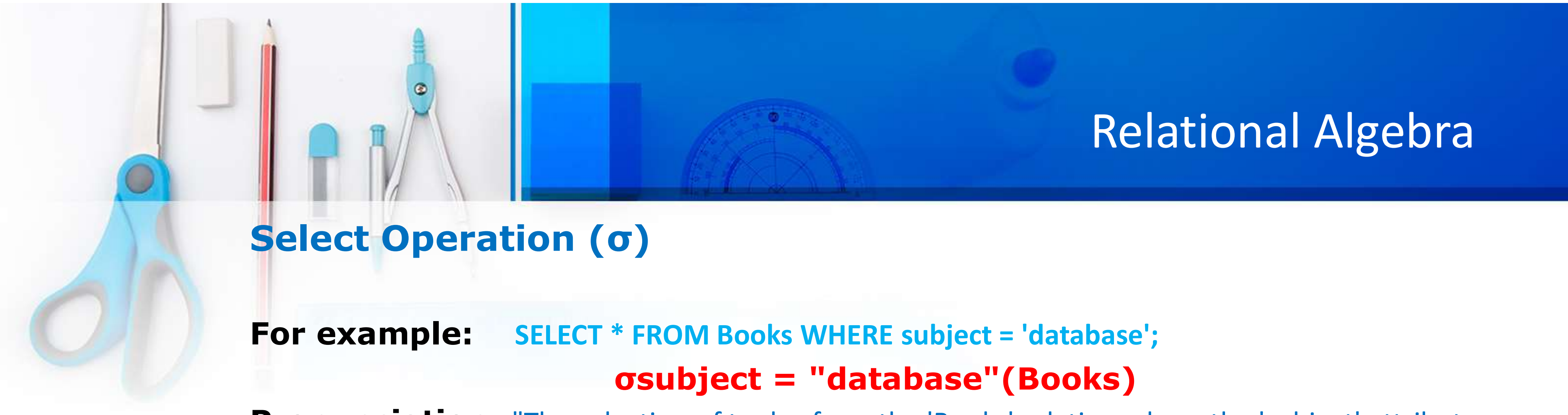
Where

σ stands for selection predicate and

r stands for relation.

p is prepositional logic formula which may use connectors like and, or, and not.

These terms may use relational operators like − =, ≠, ≥, < , >, ≤.



Relational Algebra

Select Operation (σ)

For example: SELECT * FROM Books WHERE subject = 'database';

σsubject = "database"(Books)

Pronunciation: "The selection of tuples from the 'Books' relation where the 'subject' attribute is equal to 'database'."

Output: Selects tuples from books where subject is 'database'.

SELECT * FROM Books WHERE subject = 'database' AND price = 450;

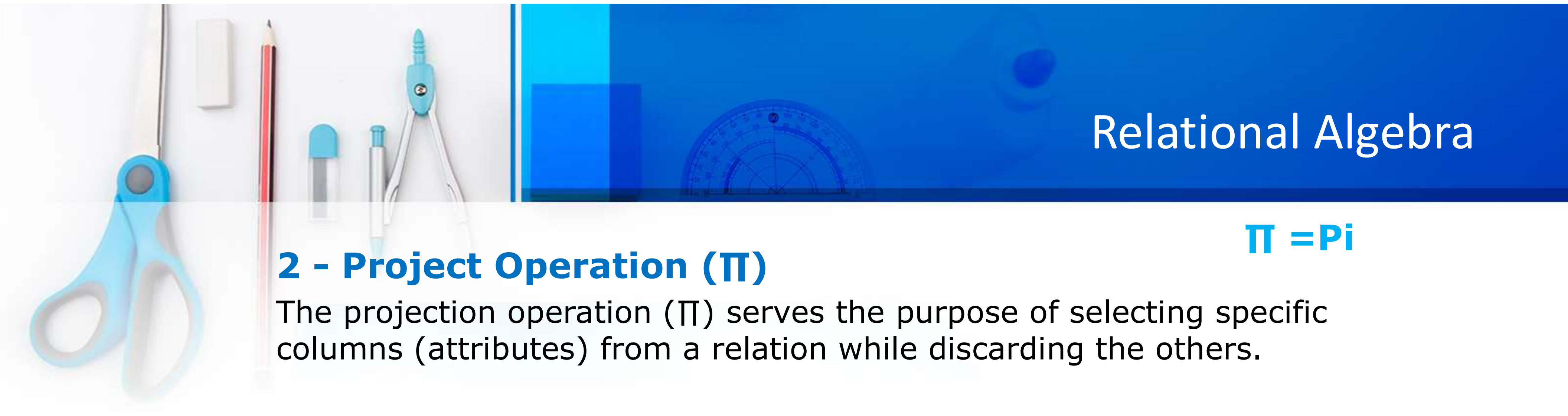
σsubject = "database" and price = "450"(Books)

Output: Selects tuples from books where subject is 'database' and 'price' is 450.

SELECT * FROM Books WHERE subject = 'database' AND price = 450 or year = 2010;

σsubject = "database" and price = "450" or year > "2010"(Books)

Output: Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.



Relational Algebra

Π =Pi

2 - Project Operation (Π)

The projection operation (Π) serves the purpose of selecting specific columns (attributes) from a relation while discarding the others.

Notation: **ΠA1, A2, An (r)**

Pronunciation: "Projection of the 'A1, A2, An' attribute from the 'r' relation

Where A1, A2 , An are attribute names of relation r.

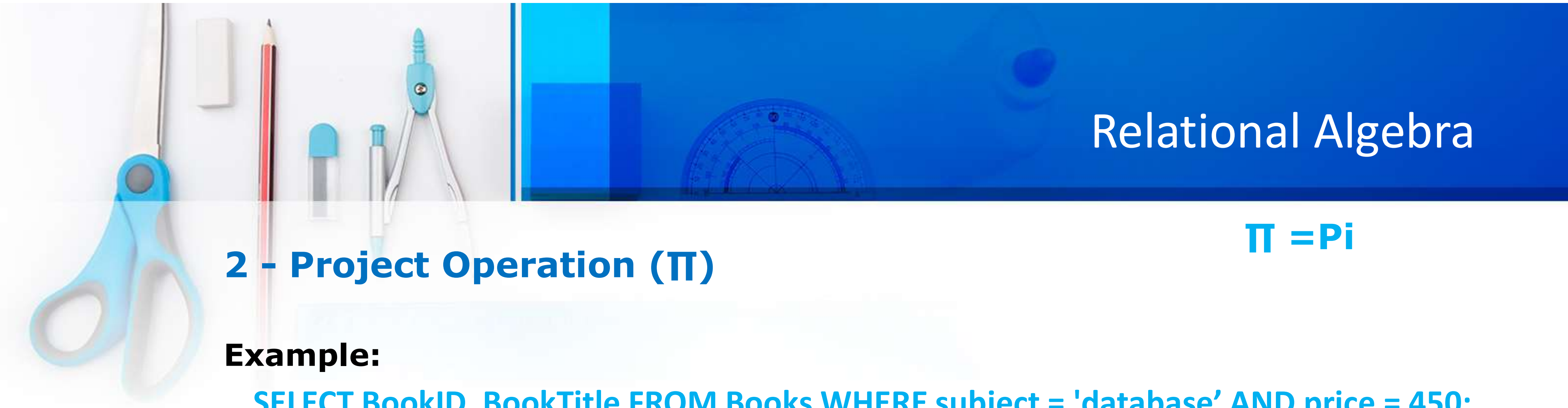
Duplicate rows are automatically eliminated, as relation is a set.

Example:

Notation: **Πsubject, author (Books)**

Pronunciation: "Projection of the 'subject, author' attributes from the 'Books' relation

Output: Selects and projects columns named as subject and author from the relation Books.



Π =Pi

2 - Project Operation (Π)

Example:

SELECT BookID, BookTitle FROM Books WHERE subject = 'database’ AND price = 450;

Notation:

πBookID, BookTitle(σsubject = ’database’ AND price = 450(Books))

Pronunciation: "Projection of the ‘BookID, BookTitle' attributes from the ‘Books' relation where the 'subject' attribute is equal to 'database’ and price attribute is equal to 450."

Note: Duplicate rows are automatically eliminated, as relation is a set.

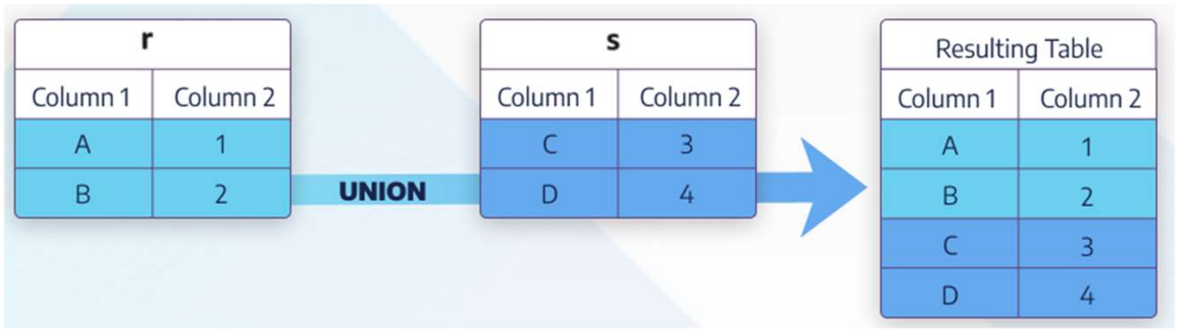
2- Project Operation (∏)

(a) Union Operation (∪)

It performs binary union between two given relations and is defined as:

Notation: **r U s**

Note: Where r and s are either database relations or relation result set (temporary relation).



r

Column 1	Column 2
A	1
B	2

UNION

s

Column 1	Column 2
C	3
D	4

Resulting Table

Column 1	Column 2
A	1
B	2
C	3
D	4

2- Project Operation (Π)

(a) Union Operation (∪)

Example:

Notation: **π author (Books) ∪ π author (Articles)**

Pronunciation: "Projection of the 'author' attribute from the 'Books' relation, union, projection of the 'author' attribute from the 'Articles' relation."

Output: Projects the names of the authors who have either written a book or an article or both.

For a union operation to be valid, the following conditions must hold −

- **r, and s must have the same number of attributes.**
- **Attribute domains must be compatible.**
- **Duplicate tuples are automatically eliminated.**

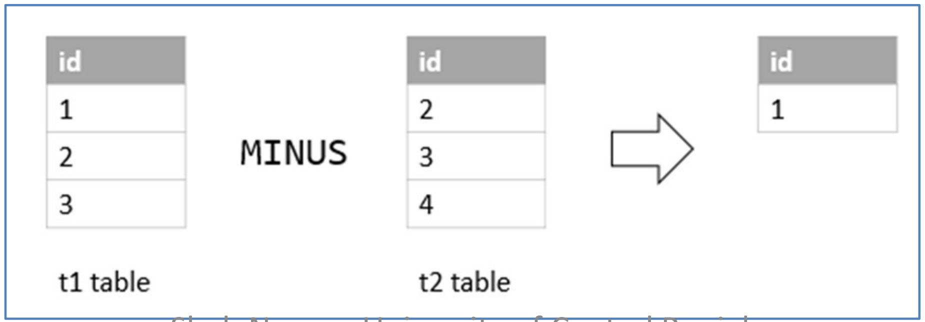
Relational Algebra

2- Project Operation (∏)

(b) Set Difference (−)

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

Notation: **r − s**



id
1
2
3

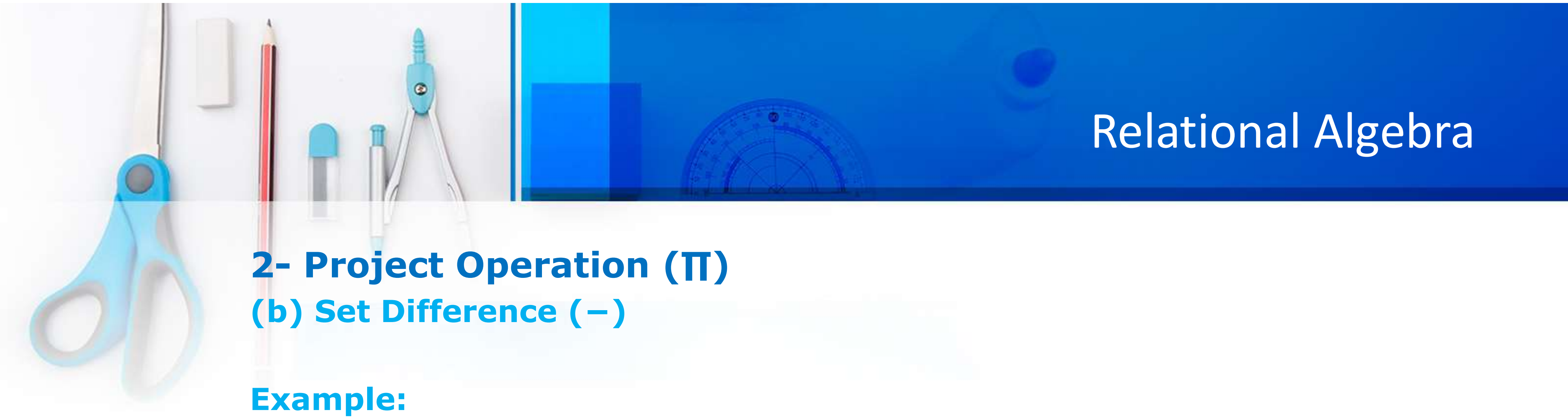
t1 table

MINUS

id
2
3
4

t2 table

id
1



2- Project Operation (Π)

(b) Set Difference (−)

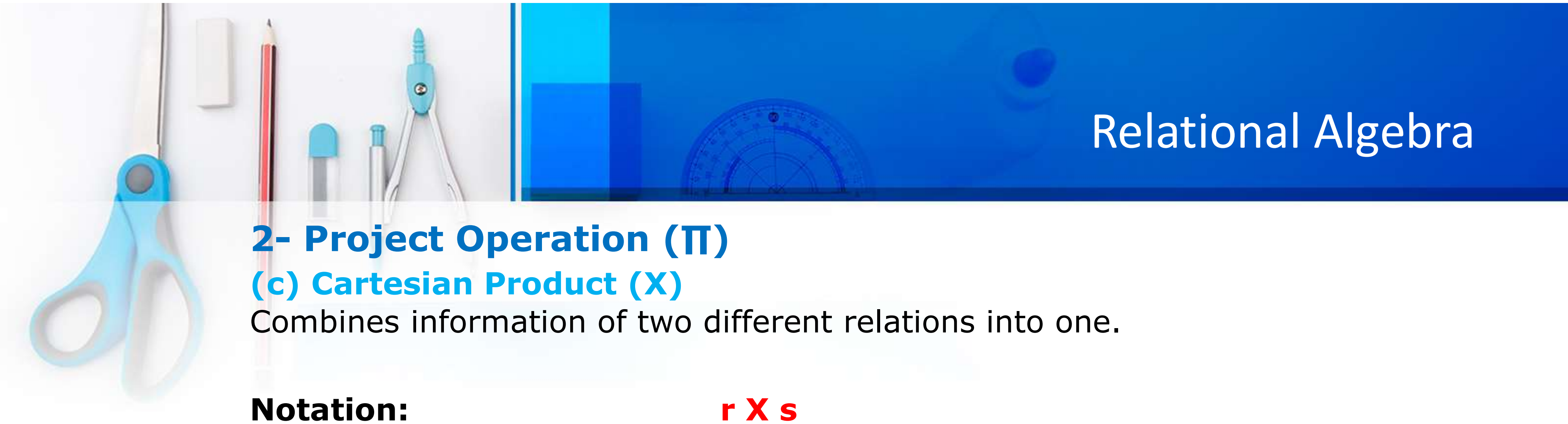
Example:

Finds all the tuples that are present in r but not in s.

Notation: π **author (Books) − π author (Articles)**

Pronunciation: "Projection of the 'author' attribute from the 'Books' relation minus projection of the 'author' attribute from the 'Articles' relation."

Output: Provides the name of authors who have written books but not articles.



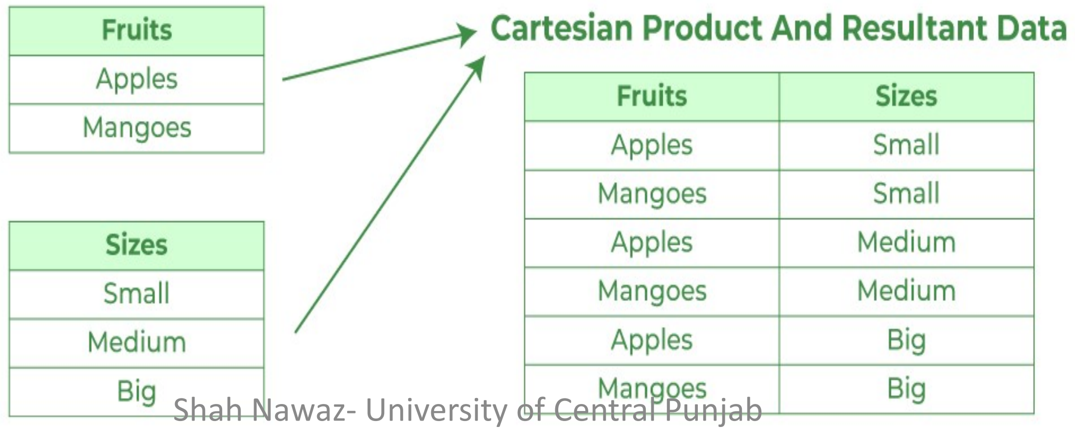
2- Project Operation (Π)

(c) Cartesian Product (X)

Combines information of two different relations into one.

Notation: r X s

Where r and s are relations.



Fruits
Apples
Mangoes

Sizes
Small
Medium
Big

Cartesian Product And Resultant Data

Fruits	Sizes
Apples	Small
Mangoes	Small
Apples	Medium
Mangoes	Medium
Apples	Big
Mangoes	Big

2- Project Operation (Π)

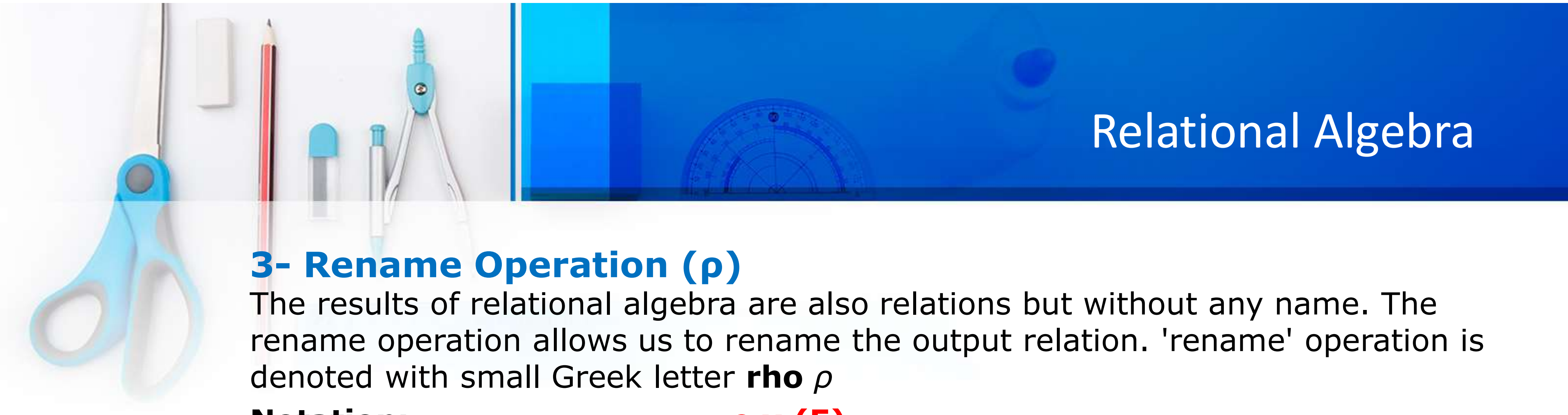
(c) Cartesian Product (X)

Example:

Notation: **π author = ‘Kiran'(Books X Articles)**

Pronunciation: "The selection of tuples from the join of 'Books' and 'Articles' relations where the 'author' attribute is equal to 'Kiran'."

Output: Yields a relation, which shows all the books and articles written by Kiran.



Relational Algebra

3- Rename Operation (ρ)

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** *ρ*

Notation: **ρ x (E)**

Pronunciation: "The rename operation on the relation E, renaming it to x."

Example 1: Rename the attributes Name, Age of table Department to A,B.

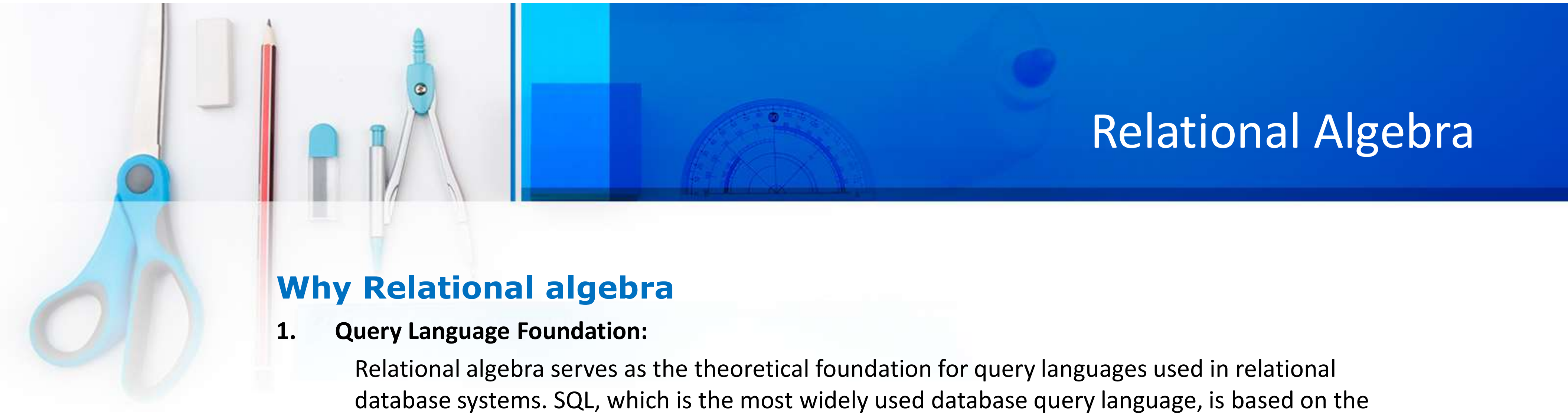
Notation: **ρ (A, B) (Department)**

Example 2: Rename the table name Project to Pro and its attributes to P, Q, R.

Notation: **ρ Pro(P, Q, R) (Project)**

Example 3: Rename the first attribute of the table Student with attributes A, B, C to P.

Notation: **ρ (P, B, C) (Student)**



Relational Algebra

Why Relational algebra

1. **Query Language Foundation:**

   Relational algebra serves as the theoretical foundation for query languages used in relational database systems. SQL, which is the most widely used database query language, is based on the principles of relational algebra.

2. **Database Querying:**

   It provides a set of operations for retrieving, filtering, and combining data from relational databases. Common operations include selection (σ), projection (π), union (∪), intersection (∩), difference (-), and join (⋈).

3. **Query Optimization:**

   Relational algebra is used by database management systems (DBMS) to optimize query execution. DBMS can analyze and transform a high-level SQL query into an equivalent, more efficient series of relational algebra operations.

4. **Formal Specification:**

   It offers a formal and mathematical way to specify queries and operations on relational databases. This precision is crucial for ensuring consistency and correctness in the handling of data.

Relational Algebra

Why Relational algebra

5. **Query Planning:**

   Relational algebra is used in the process of query planning, where the DBMS determines the most efficient way to execute a given query. This involves choosing the best order of operations and selecting appropriate indexes.

6. **Database Design:**

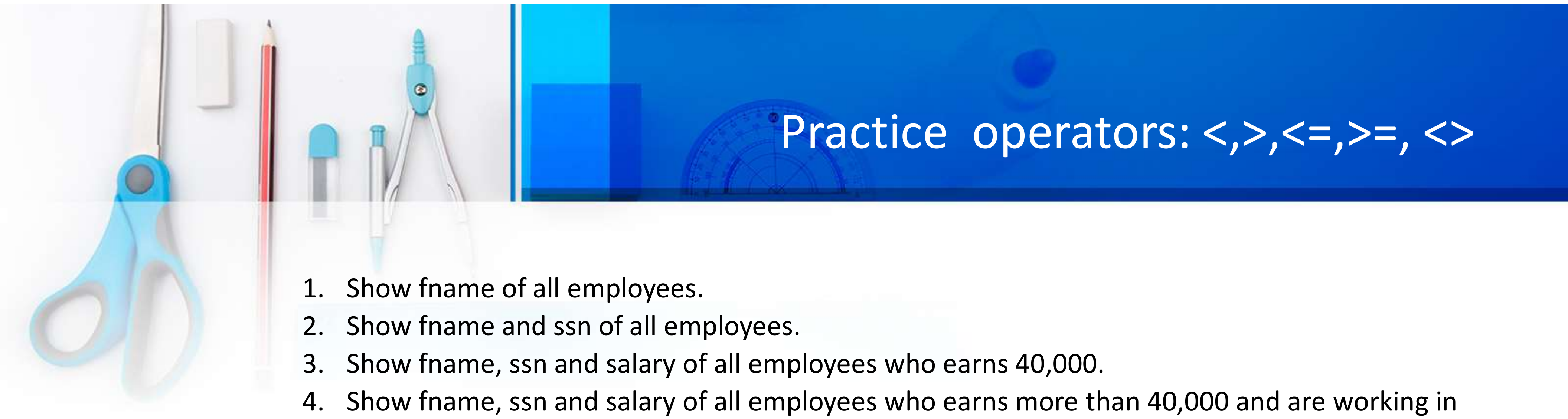
   Relational algebra concepts are valuable in the design and normalization of relational database schemas. They help in organizing data to minimize redundancy and maintain data integrity.

7. **Theoretical Understanding:**

   It provides a theoretical framework for understanding the fundamental operations and properties of relational databases. This understanding is important for both database designers and developers.

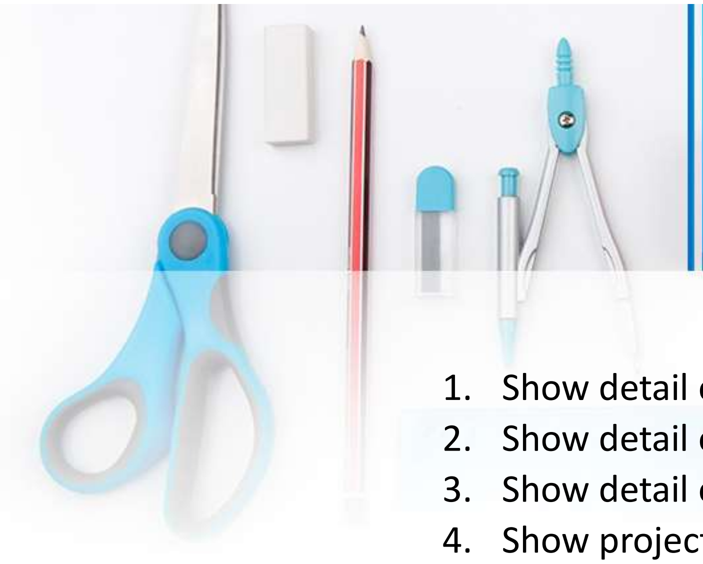
8. **Relational Database Theory:**

   Relational algebra is a fundamental part of the broader relational database theory. It defines the mathematical foundation for the relational model, which is the basis for most modern database systems.



Practice operators: <,>,<=,>=, <>

1. Show fname of all employees.
2. Show fname and ssn of all employees.
3. Show fname, ssn and salary of all employees who earns 40,000.
4. Show fname, ssn and salary of all employees who earns more than 40,000 and are working in dnumber 5.
5. Show fname of emloyees who earns 30,000 or they are female.
6. Show information of employees who have no supervisor.
7. Show fname of employees whose address is saved in our database.
8. Show uniques fname of all employees.
9. Show employees who are working in department # 5 or earn between 20,000 and 40,000
10. Show information of all employees in descending order by their salaries.



Practice Queries : LIKE / NOT LIKE

1. Show detail of employee whose first name starts with K and born in 1980.
2. Show detail of employee whose first name starts with J and last name does not end with d.
3. Show detail of department which has at least 3 alphabets in its name.
4. Show projects which has r in their name and it is on second position.
5. Show projects which does not has H in their name.
6. Show projects which does not has H on second position of their name.
7. Show name of dependent from dependent table, where dependent name starts with D and has at least 3 characters.
8. Show detail of employee whose first name starts with A and ends with A.
9. Show name of employee who have 5 digit salary.
10. Shaw name of employees who live in “Texas”.

Thank You all!

Greek Symbols: https://skylinecollege.edu/boo/greek.php