* **Relational algebra:**
* IT IS A PROCEDURAL QUERY LANGUAGE WHICH GIVES A STEP BY STEP PROCESS TO OBTAIN RESULT OF QUERY.IT USES OPERATORS TO PERFORM QUERIES.
* **TYPES OF RELATIONAL ALGEBRA:-**

1. **SELECT OPERATOR**

* IT SELECTS TUPLES THAT SATISFY A GIVEN PREDICATE.
* IT IS DENOTED BY SIGMA.

NOTATION:SIGMA.P(R)

WHERE,SIGMA—SELECTION PREDICTION.

P---PROPOSITIONAL LOGIC FORMULA WHICH MAY USE AND OR

NOT OPERATOR.

R---USED FOR RELATION.

|  |  |  |
| --- | --- | --- |
| BRANCH\_NAME | LOAN\_NO | AMOUNT |
| REDWOOD | L-17 | 1000 |
| DOWNTOWN | L-20 | 2000 |
| PERRYRIDE | L-22 | 5000 |
| DOWNTOWN | L-30 | 8000 |

INPUT:SIGMA BRANCH\_NAME =”DOWNTOWN”(LOAN)

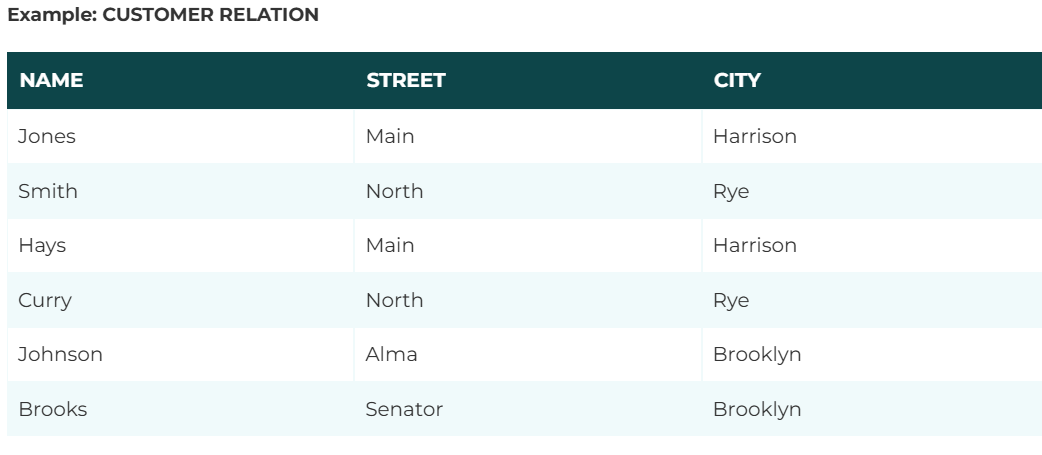
|  |  |  |
| --- | --- | --- |
| BRANCH\_NAME | LOAN\_NO | AMOUNT |
| DOWNTOWN | L-20 | 2000 |
| DOWNTOWN | L-30 | 8000 |

1. **PROJECT OPERATION**

* THIS SHOWS LIST OF THOSE ATTRIBUTES THAT WE WISH TO APPEAR IN THE RESULT & REST OF THE ATTRIBUTES ARE ELIMINATED FROM TABLE.
* IT IS DENOTED BY π

NOTATION: πA1,A2,AN(R)

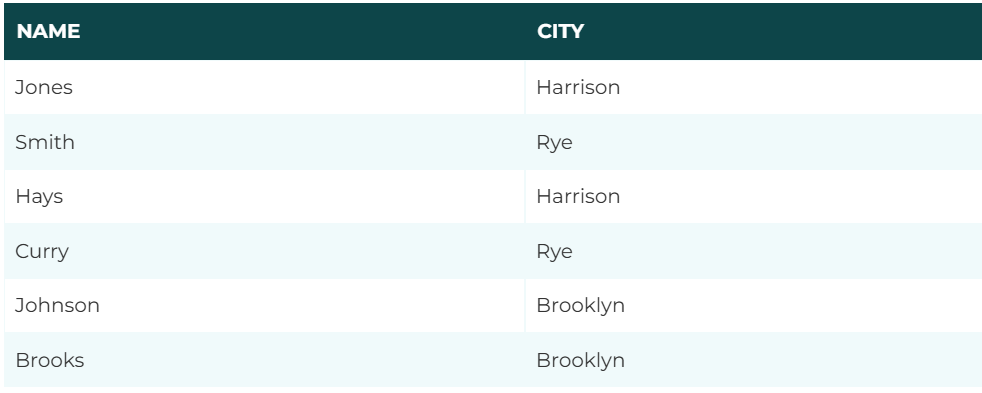
WHERE A1,A2,A3 USED AS ATTRIBUTE NAME OF RELATION R.



**Input:**

∏ NAME, CITY (CUSTOMER)

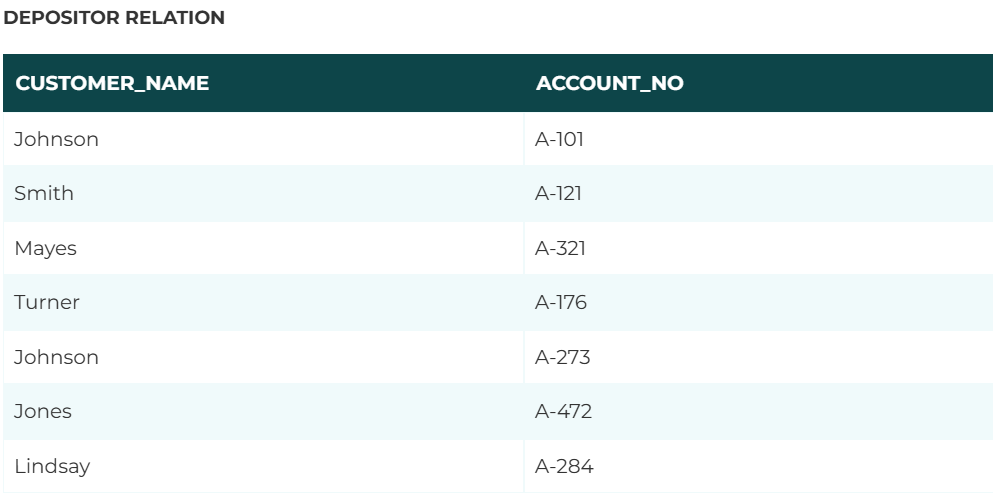
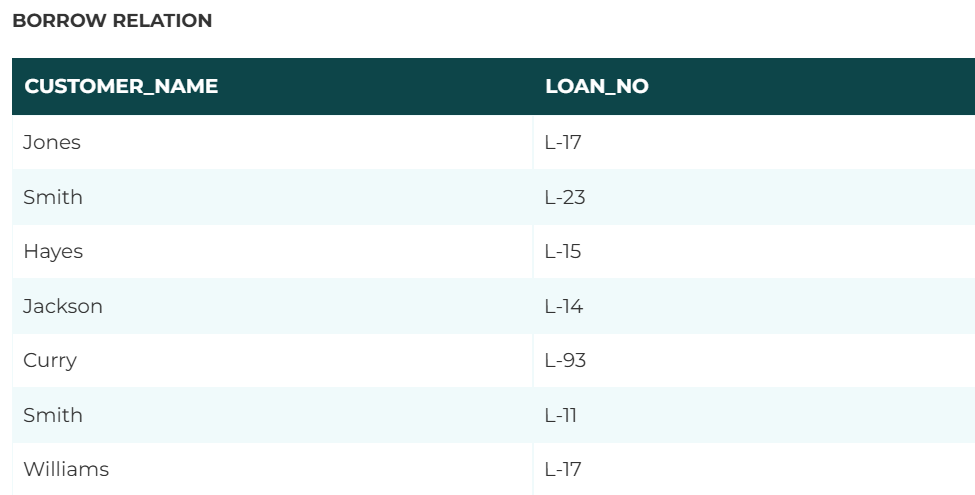
**Output:**



1. **UNION OPERATION**

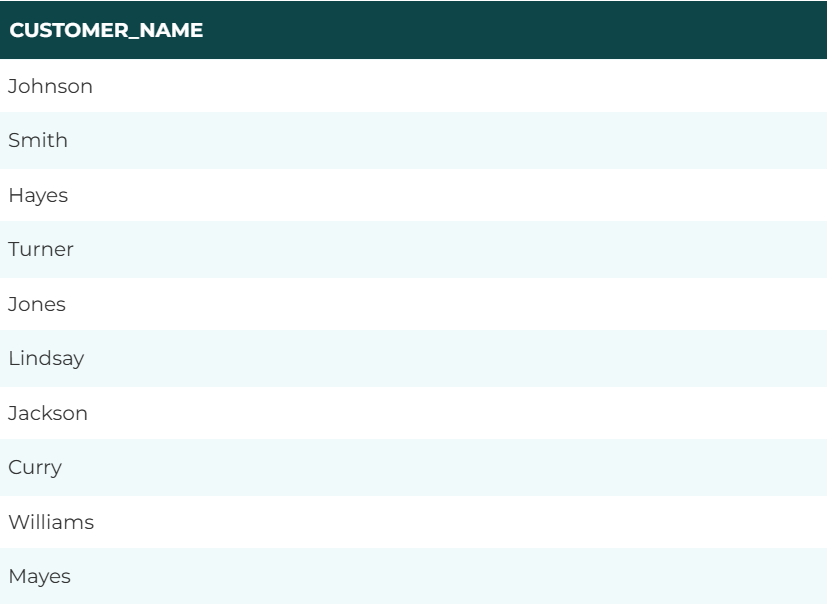
* IT IS AN OPERATION WHICH CONTAINS ALL THE TUPLES THAT ARE EITHER IN R OR S OR BOTH IN R & S.IT ELIMINATES DUPLICATE TUPLES.
* IT IS DENOTED BY U.

NOTATION:R U S.

* 
* 
* **Input:**

∏ CUSTOMER\_NAME (BORROW) ∪ ∏ CUSTOMER\_NAME (DEPOSITOR)

* Output:



1. **SET INTERSECTION**

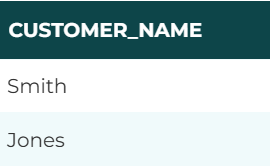
* IT CONTAINS ALL TUPLES THAT ARE IN BOTH R & S.
* IT IS DENOTED BY INTERSECTION

**∩.**NOTATION:R **∩**  S.

* **Input:**

∏ CUSTOMER\_NAME (BORROW) ∩ ∏ CUSTOMER\_NAME (DEPOSITOR)

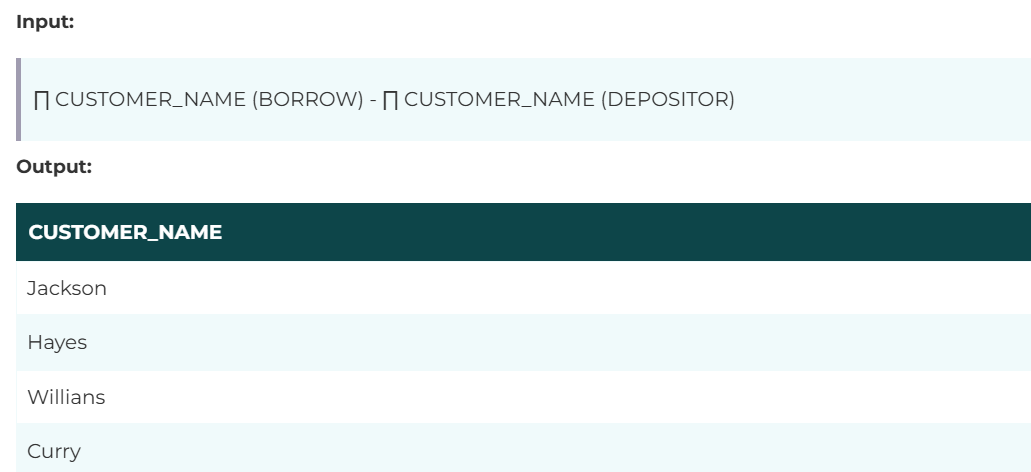
* **Output:**



1. **SET DIFFERENCE**

* THE SET INTERSECTION CONTAINS ALL TUPLES THAT ARE IN R BUT NOT IN S.
* IT IS DENOTED BY INTERSECTION(-).

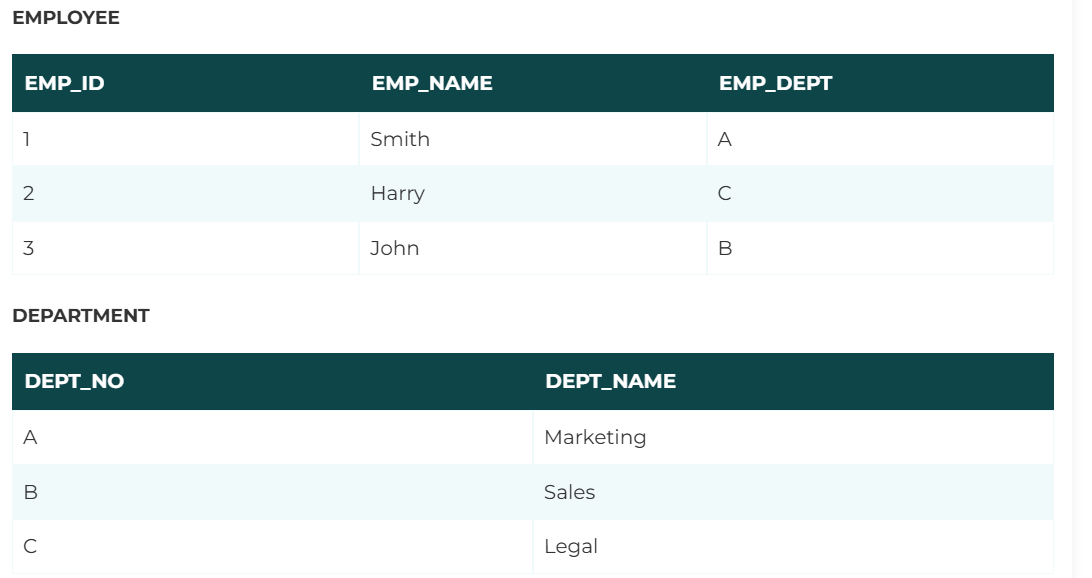
NOTATION:R-S.

* 

1. **CARTESIAN PRODUCT**

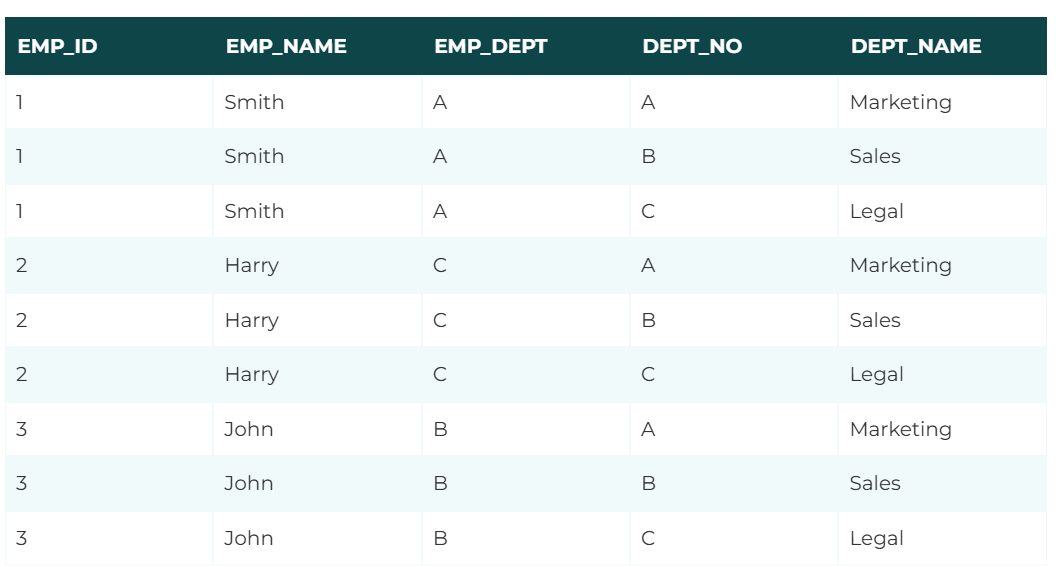
* IT IS USED TO COMBINE EACH ROW IN ONE TABLE WITH EACH ROW IN OTHER TABLE.
* IT IS ALSO CALLED CROSS PRODUCT.
* IT IS DENOTED BY X.

NOTATION:E X D.

* 
* **Input:**

EMPLOYEE X DEPARTMENT

* Output:



1. **Rename Operation:**

* The rename operation is used to rename the output relation.
* It is denoted by **rho** (ρ).
* **Example:** We can use the rename operator to rename STUDENT relation to STUDENT1.

ρ(STUDENT1, STUDENT)  

* **Relational calculus:**
* THERE IS AN ALTERNATE WAY OF FORMULATING QUERIES ARE CALLED RELATIONAL CALCULUS.
* IT IS NON-PROCEDURAL QUERY LANGUAGE.IN THIS USER IS CONCERNED WITH DTAILS OF HOW TO OBTAIN THE END RESULT.
* **WHY IT IS CALLED R.C.:**

1. IT IS BASED ON PREDICATE CALCULUS,A NAME DERIVED FROM BRANCH OF SYMBOLIC LANGUAGE.
2. MANY OF THE CALCULUS EXPRESSION INVOLVE THE USE OF QUANTIFIERS. THERE ARE TWO TYPES OF QUANTIFIERS:

**A)UNIVERSAL QUANTIFIERS.**

🡺IT IS DENOTED BY  ∀ WHICH MEANS IN A GIVEN SET OF TUPLES EXACTLY ALL TUPLES SATISFY A GIVEN CONDITION.

**B)EXISTENTIAL QUANTIFIERS.**

🡺IT IS DENOTED BY **Ǝ** WHICH MEANS IN A GIVEN SET OF TUPLES THERE IS ATLEAST ONE OCCURRENCE WHOSE VALUE SATISFY A GIVEN CONDITION.

* **TYPES OF RELATIONAL CALCULUS.:**

1. **TUPLE RELATIONAL MODEL**

* IT IS NON PROCEDURAL QUERY LANGUAGE WHICH IS BASED ON FINDING A NUMBER OF TUPLE VARIABLES ALSO CALLED RANGE VARIABLES FOR WHICH PREDICATE HOLDS TRUE.

NOTATION:{T/P(T)} OR {T/CONDITION(T)}

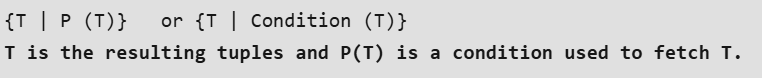
WHERE,T=RESULTING TUPLES.

P(T)=CONDITION USED TO FETCH T.

1. **DOMAIN RELATIONAL MODEL**

* IN THIS FILTERING VARIABLE USES DOMAIN OF ATTRIBUTES.IT USES LOGICAL CONNECTIVES AND,OR,NOT.IT USES EXISTENTIAL & UNIVERSAL QUANTIFIERS TO BIND VARIABLES.
* **Tuple and domain calculus:**

**(Tuple relational calculus)**

* A [Tuple Relational Calculus](https://www.geeksforgeeks.org/tuple-relational-calculus-trc-in-dbms/) is a non-procedural query language that specifies the selection of the tuples in a relation.
* It can select the tuples with a range of values or tuples for certain attribute values etc. The resulting relation can have one or more tuples.
* **Notation:**   
   
* **Example:**

****

### Advantages of Tuple Relational Calculus (TRC)

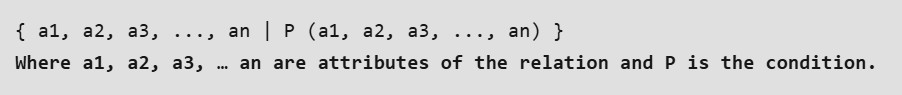
1. TRC serves as the foundation for many Query Language like [SQL](https://www.geeksforgeeks.org/sql-concepts-and-queries/).
2. You can easily write information with TRC.
3. TRC describes the data rather than the process to get that data.

### Disadvantages of Tuple Relational Calculus (TRC)

1. The practical use of TRC is very much less.
2. If we use the TRC, there might be some optimization issues.
3. You can only use TRC for some simple problems.

## **(Domain Relational Calculus):**

* A [Domain Relational Calculus](https://www.geeksforgeeks.org/domain-relational-calculus-in-dbms/) uses a list of attributes to be selected from the relation based on the condition.
* It is the same as TRC but differs by selecting the attributes rather than selecting whole tuples.
* Notation:



### ****Example:****

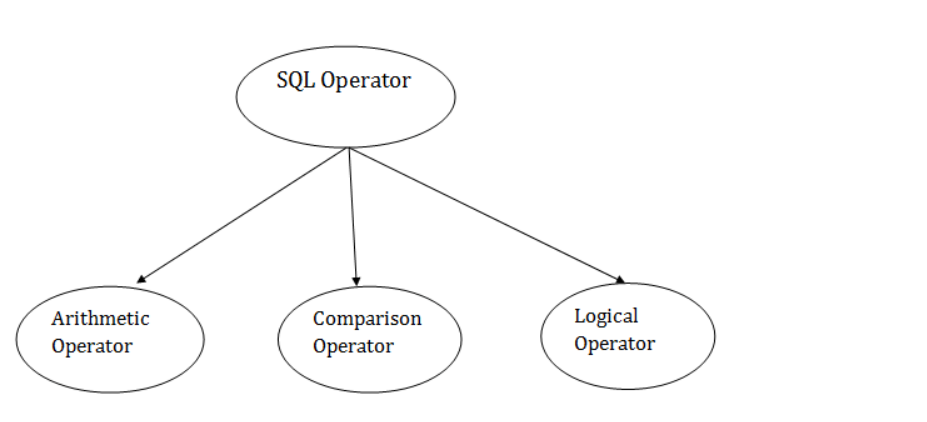
### 

### Advantages of Domain Relational Calculus (DRC)

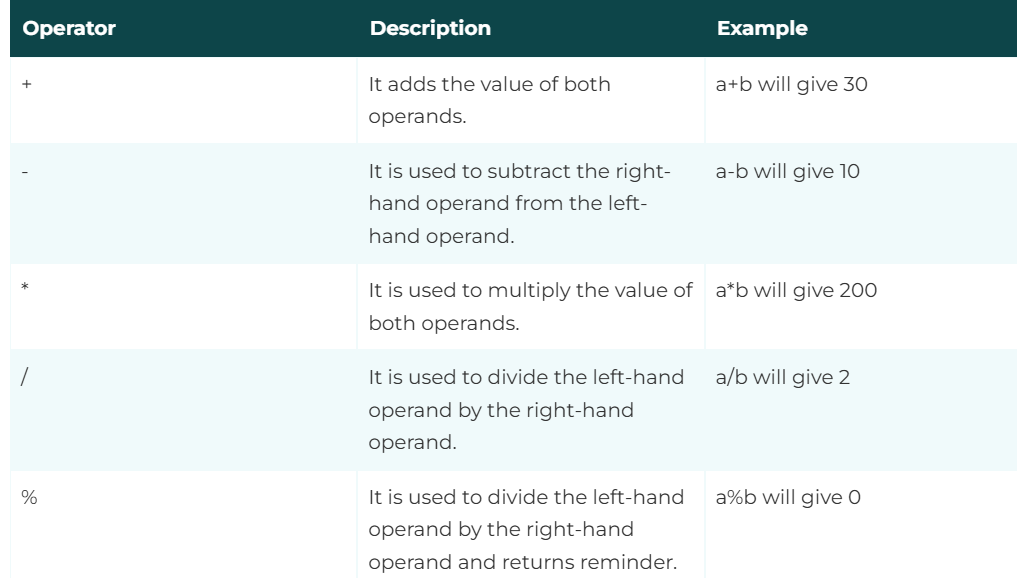
1. DRC helps work on the individual attribute values.
2. DRC helps to generate the data getting process than the definition of the data.

### Disadvantages of Domain Relational Calculus (DRC)

1. The system developed with DRC is hard to maintain.
2. DRC can’t work with multiple domains at the same time.

* **SQL COMMANDS:**
* **SQL OPERATORS:**
* SQL operators are symbols or keywords used in SQL queries to perform operations on data. They are used to filter, modify, or compare data in a database. SQL operators can be divided into several types, each serving different purposes.
* 

1. Arithmetic operator:

* Let's assume 'variable a' and 'variable b'. Here, 'a' contains 20 and 'b' contains 10.
* 

1. Comparison operator:

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| = | It checks if two operands values are equal or not, if the values are queal then condition becomes true. | (a=b) is not true |
| != | It checks if two operands values are equal or not, if values are not equal, then condition becomes true. | (a!=b) is true |
| <> | It checks if two operands values are equal or not, if values are not equal then condition becomes true. | (a<>b) is true |
| > | It checks if the left operand value is greater than right operand value, if yes then condition becomes true. | (a>b) is not true |
| < | It checks if the left operand value is less than right operand value, if yes then condition becomes true. | (a<b) is true |
| >= | It checks if the left operand value is greater than or equal to the right operand value, if yes then condition becomes true. | (a>=b) is not true |
| <= | It checks if the left operand value is less than or equal to the right operand value, if yes then condition becomes true. | (a<=b) is true |
| !< | It checks if the left operand value is not less than the right operand value, if yes then condition becomes true. | (a!=b) is not true |
| !> | It checks if the left operand value is not greater than the right operand value, if yes then condition becomes true. | (a!>b) is true |

1. Logical operator:

|  |  |
| --- | --- |
| Operators | Description |
| All | It compares a value to all values in another value set. |
| And | It allows the existence of multiple conditions in an SQL statement. |
| Any | It compares the values in the list according to the condition. |
| Between | It is used to search for values that are within a set of values. |
| In | It compares a value to that specified list value. |
| Not | It reverses the meaning of any logical operator. |
| Or | It combines multiple conditions in SQL statements. |
| Exists | It is used to search for the presence of a row in a specified table. |
| Like | It compares a value to similar values using wildcard operator. |

* **Finding candidate keys in functional dependency:**
* **Canonical cover of fd:**
* A canonical cover is a set of functional dependencies that is equivalent to a given set of functional dependencies but is minimal in terms of the number of dependencies.
* **The process of finding the canonical cover of a set of functional dependencies involves three main steps:**

1. **Reduction:**

* The first step is to reduce the original set of functional dependencies to an equivalent set that has the same closure as the original set, but with fewer dependencies. This is done by removing redundant dependencies and combining dependencies that have common attributes on the left-hand side.

1. **Elimination:**

* The second step is to eliminate any extraneous attributes from the left-hand side of the dependencies. An attribute is considered extraneous if it can be removed from the left-hand side without changing the closure of the dependencies.

1. **Minimization:**

* The final step is to minimize the number of dependencies by removing any dependencies that are implied by other dependencies in the set.

## **How to Find Canonical Cover**

* To compute the canonical cover for set F, follow this algorithm.

1. Use the union rule to replace any dependencies in α1 → β1 and α2 → β2 with α1 → β1β2.
2. Find a functional dependency α → β with an extraneous attribute either in α or in β.
3. If an extraneous attribute is found, delete it from α → β.
4. Repeat until F does not change.

## **Features of the Canonical Cover**

## **Minimal:**

## The canonical cover is the smallest set of dependencies that can be derived from a given set of dependencies, i.e., it has the minimum number of dependencies required to represent the same set of constraints.

## **Lossless:**

## The canonical cover preserves all the functional dependencies of the original set of dependencies, i.e., it does not lose any information.

1. **Deterministic:**

* The canonical cover is deterministic, i.e., it does not contain any redundant or extraneous dependencies.

1. **Reduces Data Redundancy:**

* The canonical cover helps to reduce data redundancy by eliminating unnecessary dependencies that can be inferred from other dependencies.

1. **Improves Query Performance:**

* The canonical cover helps to improve query performance by reducing the number of [joins](https://www.geeksforgeeks.org/sql-join-set-1-inner-left-right-and-full-joins) and redundant data in the database.

1. **Facilitates Database Maintenance:**

* The canonical cover makes it easier to modify, update, and delete data in the database by reducing the number of [dependencies](https://www.geeksforgeeks.org/types-of-functional-dependencies-in-dbms) that need to be considered.
* **Example 1**

**Q) Given F = { A → BC, B → C, A → B, AB → C }**

**ANS:**

* **Step 1 Reduction:** There are two functional dependencies with the same attributes on the left: A → BC, A → B are already in their simplest form.
* **Step 2 Elimination:** In A → BC, C is extraneous because A → C can be derived from A → B and B → C. Thus, we reduce it to A → B.
* **Step 3 Minimization:** No redundant dependencies remain.

Hence, the canonical cover is Fc = { A → B, B → C }

* **Example 2**

**Q)Given F = { A → BC, CD → E, B → D, E → A }**

**ANS:**

* **Step 1 Reduction:**Each left-hand side of the functional dependencies is unique and cannot be combined further.
* **Step 2 Elimination:**None of the attributes on the left or right sides of any functional dependency are extraneous.
* **Step 3 Minimization:**No dependencies are redundant.

Hence, the canonical cover is F = { A → BC, CD → E, B → D, E → A }

* **Lossless and lossy join decomposition:**
* The process of breaking up a relation into smaller sub-relations is called Decomposition.
* Decomposition is required in DBMS to convert a relation into a specific normal form which further reduces redundancy, anomalies, and inconsistency in the relation.
* **There are mainly two types of decompositions in DBMS**

1. **Lossless join Decomposition**

* [Lossless join decomposition](https://www.geeksforgeeks.org/lossless-decomposition-in-dbms/) is a process in which a relation is decomposed into smaller relations without losing any information. When we rejoin the decomposed relations, the original relation is perfectly reconstructed without losing data.
* **Advantages of Lossless Join Decomposition**

1. **Data Integrity:**

* On decomposed tables no loss of any data or informationwhen re-join them together, it becomes the original table before decomposition.

1. **Consistency:**

* This decomposition ensures that data will remain accurate and consistent across the database.

1. **Normalization:**

* This helps in achieving higher normal forms like [3NF](https://www.geeksforgeeks.org/third-normal-form-3nf/)or [BCNF](https://www.geeksforgeeks.org/boyce-codd-normal-form-bcnf/) and improving efficiency.

### Disadvantage of Lossless Join Decomposition

1. **Storage Overhead:**

* Storage usage is increased, as sometimes additional tables and columns are needed.

1. **Complex Queries:**

* To rejoin the table we decomposed may require complex [SQL queries](https://www.geeksforgeeks.org/sql-concepts-and-queries/), and these queries may impact the performance.

1. **Lossy join Decomposition**

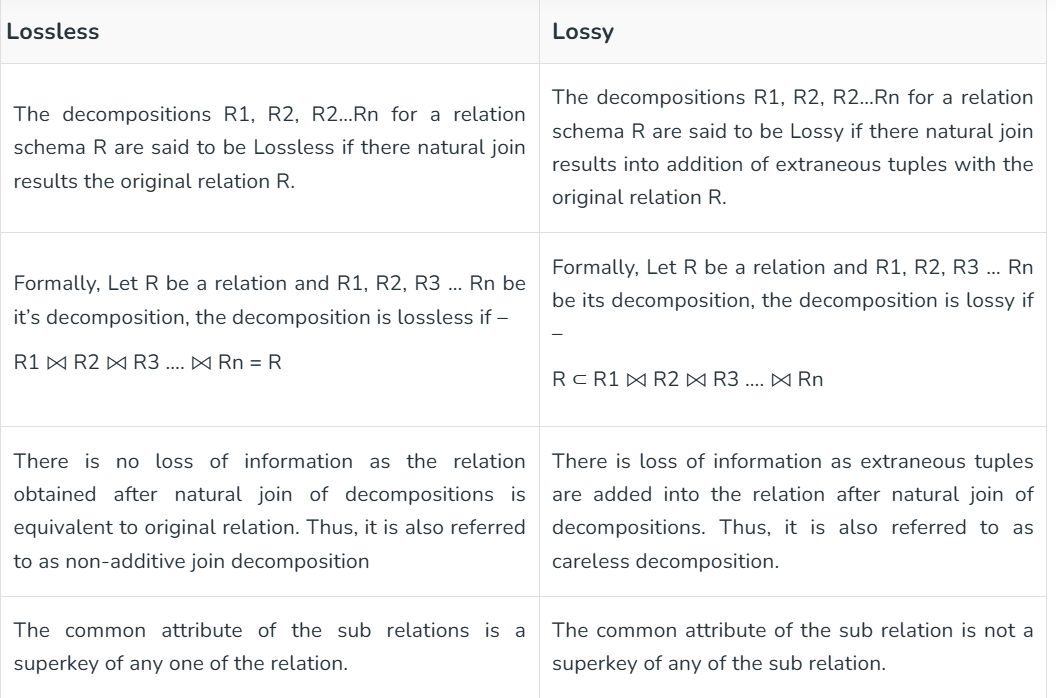
* In this type of decomposition the information will be lost if the relations are decomposed into smaller parts. This means that when the original relation is decomposed and then later we try to rejoin them back together then some data from the original relation is not lost and not recoverable, which leads to data inconsistencies.

### Advantages of Lossy Join Decomposition

1. **Structure is Simple,**the result of decomposing the relations will give simple and smaller sub tables and this helps in reducing the complexity in some cases.
2. **Redundancy is Less:**In some cases where loss of information is acceptable will help in reducing redundancy in certain cases.

### Disadvantage of Lossy Join Decomposition

1. **Loss of Data:**When tables are joined back together then some of the information will be loosed permanently which can cause problem.
2. **Inconsistency:**as discussed above, due to information loss the data integrity problem will arise.
3. **Hard to manage:**It can be difficult to maintain data consistency as some information may loss, which is making harder to manage the database.

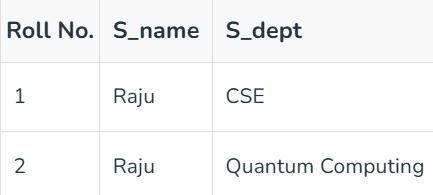


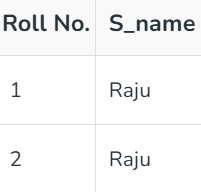
* **Example-1:**

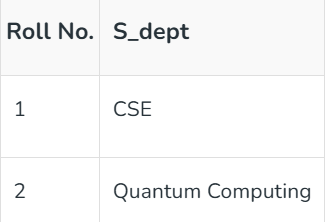
1. **Example to check whether given Decomposition Lossless Join Decomposition.**

**ANS:**

* Let there be a relational schema Student(Roll No., S\_name, S\_dept). StudentDetails(Roll No., S\_name) and Dept(Roll No., S\_dept) be it’s decompositions.

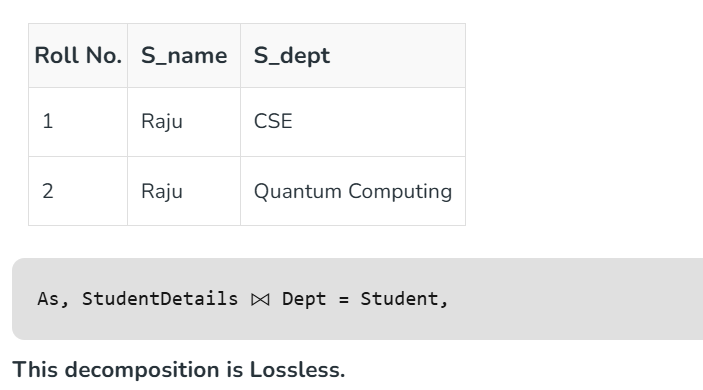






* Now for the decomposition to be lossless,

StudentDetails ⨝ Dept = Student then, StudentDetails ⨝ Dept is



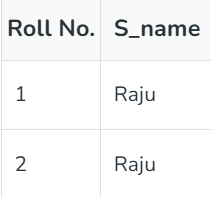
* **Example-2:**

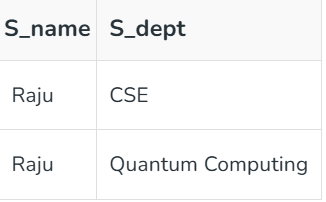
1. **Example to check whether given Decomposition Lossy Join Decomposition.**

**ANS:**

* Let there be a relational schema Student(Roll No., S\_name, S\_dept). StudentDetails(Roll No., S\_name) and Dept(S\_name, S\_dept) be it’s decompositions.

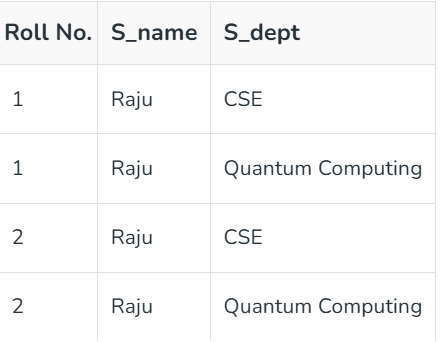






* Now for the decomposition to be lossy,







* This decomposition is Lossy.