EXPERIMENT - 9

AIM: To study and implement Woolman Algorithm to detect Lossy Decomposition.

THEORY: Lossless Decomposition is a decomposition technique where we ensure that we can always recover the original relation from the smaller relations produced by the decomposition. In order words, no information is lost during the decomposition process. This ensures that we maintain all the functional dependencies present in the original schema.

Woolman's Algorithm:
It is used to check whether a decomposition of a relation is lossy or lossless. It is a test for lossless (non-additive) join property. Here, a universal relation R, a decomposition $D = \{R_1, R_2, R_3, R_4 -- R_n\}$ of R, and a set F of functional dependencies are defined, where $R = \{A_1, A_2, A_3 \cdots A_n\}$

Algorithm:
Initialization: A table is created with the attributes of the original relation as the columns and the smaller relation as the rows. The table is then filled with the value of the smaller

relation, with a (column) if attribute is present and b (row, column) if attribute is absent.

Crete an initial matrix S with one row i for each relation in Ri in D and

Create an initial matrix S with one row i for each rotation in Ri in D, and one column j for each attribute Aj in R. Set S(i,j) = bij for all matrix entries, if relation Ri includes attribute Aj then set S(i,j) = aj

ex: Let R = (A, B, C, D), FD = (A→B, B→C, C→D) and R is decomposed into R_1(A,B), R_2(B,C), R_3(C,D)

		A	B	C	D
1	R_1	a_1	a_2	b_{13}	b_{14}
2	R_2	b_{21}	a_2	a_3	b_{24}
3	R_3	b_{31}	b_{32}	a_3	a_4

2) <u>Processing</u>: For every functional dependency in the original relation, use check if the smaller relations commonly contain the attributes on the left hand side of the functional dependency. If they do, we mark the right hand side with the common value in the table, preferring a over b. If they do not, we leave it as it is.

For each functional dependency $X \to Y$ in F

i) If any of rows have an "a" symbol for the column, set the other rows to that same "a" symbol in the column

ii) If no "a" symbol exists for the attribute in any of the rows, choose one of the "b" symbols that appear in one of the rows ~~of~~ for the attribute and set the other rows to that same "b" symbol in the column.

Repeat the following loop until a complete loop execution results in no change to S.

3) Termination: The algorithm terminates when either one of the rows is completely filled with a or, when the table is not changed in a pass. If there is a row that contains all a, then the decomposition is lossless. If not, then the decomposition is lossy.

		1	2	3	4
		A	B	C	D
1	R_1	a_1	a_2	b_{13} a_3	b_{14}
2	R_2	b_{21}	a_2	a_3	b_{24}
3	R_3	b_{31}	b_{31}	a_3	a_4

$\Longrightarrow$

CODE (WOOLMAN'S ALGORITHM) :

```
def initialize_table(relation, decomp_relation):
    table = []
    for decomp_rel in decomp_relation:
        row = []
        for attr in relation:
            if attr in decomp_rel:
                row.append('a' + str(relation.index(attr) + 1))
            else:
                row.append('b')
        table.append(row)
    return table


def check_lossy_lossless(relation, decomp_relation, functional_dependencies):
    table = initialize_table(relation, decomp_relation)
    print("Initial Table:")
    print_table(table, relation)

    repeat = 0
    while repeat < len(relation):
        for fd in functional_dependencies:
            deriving_attrs = fd[0]
            derived_attrs = fd[1]
            flag = 0
            for attr in derived_attrs:
                col_index = relation.index(attr)
                col_values = [row[col_index] for row in table]
                if 'a' + str(col_index + 1) in col_values and 'b' in
col_values:
                    flag = 1
                    break
            if flag == 1:
                for attr in derived_attrs:
                    col_index = relation.index(attr)
                    for row in table:
                        if row[col_index] == 'b':
                            row[col_index] = 'a' + str(col_index + 1)
                repeat = 0
                print("\nChanged Table after checking dependency ",fd[0],"-
>",fd[1],":")
                print_table(table,relation)
                # break
            else:
                repeat += 1
```

```
                print("\nNo change in Table after checking dependency
",fd[0],"->",fd[1])

    lossless = any(all(attr.startswith('a') for attr in row) for row in table)

    print("\nFinal Table:")
    print_table(table, relation)

    if lossless:
        print("\nThe given decomposed relation is LOSSLESS.")
    else:
        print("\nThe given decomposed relation is LOSSY.")


def print_table(table, relation):
    num_rows = len(table)
    num_cols = len(relation)

    for row_num, row in enumerate(table):
        row_vals = []
        for col_num, attr in enumerate(row):
            if attr.startswith('a'):
                row_vals.append(attr)
            else:
                row_vals.append('b' + str(row_num + 1) + str(col_num + 1))
        print('\t'.join(row_vals))


# Example usage
if __name__ == "__main__":
    relation = ["A", "B", "C", "D", "E"]
    decomp_relation = ["AD", "AB", "BE", "CDE", "AE"]
    functional_dependencies = [["A", "C"], ["B", "C"], ["C", "D"], ["DE",
"C"], ["CE", "A"]]

    check_lossy_lossless(relation, decomp_relation, functional_dependencies)
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

b41     b42     a3      a4      a5
a1      b52     a3      b54     a5

No change in Table after checking dependency  B -> C

Changed Table after checking dependency  C -> D :
a1      b12     a3      a4      b15
a1      a2      a3      a4      b25
b31     a2      a3      a4      a5
b41     b42     a3      a4      a5
a1      b52     a3      a4      a5

No change in Table after checking dependency  DE -> C

Changed Table after checking dependency  CE -> A :
a1      b12     a3      a4      b15
a1      a2      a3      a4      b25
a1      a2      a3      a4      a5
a1      b42     a3      a4      a5
a1      b52     a3      a4      a5

No change in Table after checking dependency  A -> C

No change in Table after checking dependency  B -> C

No change in Table after checking dependency  C -> D

No change in Table after checking dependency  DE -> C

No change in Table after checking dependency  CE -> A

Final Table:
a1      b12     a3      a4      b15
a1      a2      a3      a4      b25
a1      a2      a3      a4      a5
a1      b42     a3      a4      a5
a1      b52     a3      a4      a5

The given decomposed relation is LOSSLESS.
PS C:\Users\ATHARVA>
```

		1	2	3	4
		A	B	C	D
1	R_1	a_1	a_2	a_3	b_{14} a_4
2	R_2	b_{21}	a_2	a_3	b_{24} a_4
3	R_3	b_{31}	b_{32}	a_3	a_4

Row 1 contain all a's
∴ the decomposition relation are lossless.

CONCLUSION: A python program to detect lossy decomposition by Woolman's algorithm is successfully implemented and studied.