1, 2)

Code:

#include <stdio.h>

#include <string.h>

#include <conio.h>

struct fd

{

    int left[8], right[8];

    int lcount, rcount;

} f[10];

int attrcount, closcount = 0, fdcount, closure[10];

char attr[10][25];

int nolnor[8], ronly[8], lonly[8], merg1n3[8], exteriors[8];

void getclosure();

void get\_nolnor();

void get\_ronly();

void get\_lonly();

void get\_merg1n3();

int iscomplete();

void getclosure()

{

    int i, j, k, l = 0, issubset, found;

    do

    {

        for (i = 0; i <= fdcount; i++)

        {

            issubset = 1;

            for (j = 0; j < f[i].lcount; j++)

            {

                found = 0;

                for (k = 0; k < closcount; k++)

                {

                    if (closure[k] == f[i].left[j])

                    {

                        found = 1;

                        break;

                    }

                }

                if (found == 0)

                {

                    issubset = 0;

                    break;

                }

            }

            if (issubset == 1)

            {

                for (k = 0; k < f[i].rcount; k++)

                {

                    found = 0;

                    for (j = 0; j < closcount; j++)

                    {

                        if (closure[j] == f[i].right[k])

                            found = 1;

                    }

                    if (found == 0)

                    {

                        closure[closcount] = f[i].right[k];

                        closcount++;

                    }

                }

            }

        }

        l++;

    } while (l < attrcount);

}

void get\_nolnor()

{

    int i, found, j, k, l = 0;

    for (i = 0; i < attrcount; i++)

    {

        found = 0;

        for (j = 0; j <= fdcount; j++)

        {

            for (k = 0; k < f[j].lcount; k++)

            {

                if (i == f[j].left[k])

                {

                    found = 1;

                    break;

                }

            }

            if (found == 1)

                break;

            for (k = 0; k < f[j].rcount; k++)

            {

                if (i == f[j].right[k])

                {

                    found = 1;

                    break;

                }

            }

            if (found == 1)

                break;

        }

        if (found == 0)

        {

            nolnor[l] = i;

            l++;

        }

    }

    nolnor[l] = 222;

}

void get\_ronly()

{

    int rpresent, lpresent, i, j, k, l = 0;

    for (i = 0; i < attrcount; i++)

    {

        rpresent = 0;

        for (j = 0; j <= fdcount; j++)

        {

            for (k = 0; k < f[j].rcount; k++)

            {

                if (i == f[j].right[k])

                {

                    rpresent = 1;

                    break;

                }

            }

            if (rpresent == 1)

                break;

        }

        lpresent = 0;

        if (rpresent == 1)

        {

            for (j = 0; j <= fdcount; j++)

            {

                for (k = 0; k < f[j].lcount; k++)

                {

                    if (i == f[j].left[k])

                    {

                        lpresent = 1;

                        break;

                    }

                }

                if (lpresent == 1)

                    break;

            }

        }

        if (lpresent == 0 && rpresent == 1)

            ronly[l++] = i;

    }

    ronly[l] = 222;

}

void get\_lonly()

{

    int rpresent, lpresent, i, j, k, l = 0;

    for (i = 0; i < attrcount; i++)

    {

        lpresent = 0;

        for (j = 0; j <= fdcount; j++)

        {

            for (k = 0; k < f[j].lcount; k++)

            {

                if (i == f[j].left[k])

                {

                    lpresent = 1;

                    break;

                }

            }

            if (lpresent == 1)

                break;

        }

        rpresent = 0;

        if (lpresent == 1)

        {

            for (j = 0; j <= fdcount; j++)

            {

                for (k = 0; k < f[j].rcount; k++)

                {

                    if (i == f[j].right[k])

                    {

                        rpresent = 1;

                        break;

                    }

                }

                if (rpresent == 1)

                    break;

            }

        }

        if (lpresent == 1 && rpresent == 0)

            lonly[l++] = i;

    }

    lonly[l] = 222;

}

void get\_merg1n3()

{

    int i, j;

    for (i = 0, j = 0; lonly[j] != 222; i++, j++)

        merg1n3[i] = lonly[j];

    for (j = 0; nolnor[j] != 222; i++, j++)

        merg1n3[i] = nolnor[j];

    merg1n3[i] = 222;

}

int compare(char temp[25])

{

    int i;

    for (i = 0; i < attrcount; i++)

        if (strcmp(temp, attr[i]) == 0)

            return i;

    return 0;

}

int iscomplete()

{

    if (closcount != attrcount)

        return 0;

    else

        return 1;

}

void main()

{

    int i, j, k, attcode, found;

    char schema[100], temp[45], temp1[50];

    for (i = 0; i < 10; i++)

    {

        f[i].lcount = 0;

        f[i].rcount = 0;

    }

    printf("\nEnter the schema\n");

    scanf("%s", schema);

    attrcount = 0;

    for (i = 0; schema[i] != '('; i++)

        ;

    do

    {

        j = 0;

        i++;

        while (schema[i] != ',' && schema[i] != ')')

        {

            temp[j] = schema[i];

            j++;

            i++;

        }

        temp[j] = '\0';

        strcpy(attr[attrcount], temp);

        attrcount++;

    } while (schema[i] == ',');

    fdcount = -1;

    printf("\nEnter the functional dependancies\nEnter 0 to stop\n");

    for (i = 0; i < 10; i++)

    {

        scanf("%s", temp1);

        if (strcmp(temp1, "0") == 0)

            break;

        fdcount++;

        j = 0;

        if (temp1[0] == '{' || temp1[0] == '(')

            j++;

        do

        {

            if (temp1[j] == ',')

                j++;

            k = 0;

            while (temp1[j] != ',' && temp1[j] != ')' && temp1[j] != '}' && temp1[j] != '-')

            {

                temp[k] = temp1[j];

                k++;

                j++;

            }

            temp[k] = '\0';

            attcode = compare(temp);

            f[fdcount].left[f[fdcount].lcount] = attcode;

            f[fdcount].lcount++;

        } while (temp1[j] == ',');

        if (temp1[j] == ')' || temp1[j] == '}')

            j += 3;

        else if (temp1[j] == '-')

            j += 2;

        if (temp1[j] == '{' || temp1[j] == '(')

            j++;

        do

        {

            if (temp1[j] == ',')

                j++;

            k = 0;

            while (temp1[j] != ',' && temp1[j] != ')' && temp1[j] != '}' && temp1[j] != '\0')

            {

                temp[k] = temp1[j];

                k++;

                j++;

            }

            temp[k] = '\0';

            attcode = compare(temp);

            f[fdcount].right[f[fdcount].rcount] = attcode;

            f[fdcount].rcount++;

        } while (temp1[j] == ',');

    }

    get\_nolnor();

    get\_ronly();

    get\_lonly();

    get\_merg1n3();

    closcount = 0;

    for (i = 0; merg1n3[i] != 222; i++)

    {

        closure[closcount++] = merg1n3[i];

    }

    getclosure();

    i = iscomplete();

    if (i == 1)

    {

        printf("\nThe candidate key is:\n{");

        for (i = 0; merg1n3[i] != 222; i++)

        {

            printf("%s,", attr[merg1n3[i]]);

        }

        printf("\b ");

        printf("}");

    }

    else

    {

        k = 0;

        for (i = 0; i < attrcount; i++)

        {

            found = 0;

            for (j = 0; ronly[j] != 222; j++)

            {

                if (i == ronly[j])

                {

                    found = 1;

                    break;

                }

            }

            if (found == 0)

            {

                for (j = 0; merg1n3[j] != 222; j++)

                {

                    if (i == merg1n3[j])

                    {

                        found = 1;

                        break;

                    }

                }

            }

            if (found == 0)

                exteriors[k++] = i;

        }

        exteriors[k] = 222;

        printf("Candidate Keys:");

        for (k = 0; exteriors[k] != 222; k++)

        {

            closcount = 0;

            for (i = 0; merg1n3[i] != 222; i++)

                closure[closcount++] = merg1n3[i];

            closure[closcount++] = exteriors[k];

            getclosure();

            i = iscomplete();

            if (i == 1)

            {

                printf("\n{");

                for (i = 0; merg1n3[i] != 222; i++)

                    printf("%s,", attr[merg1n3[i]]);

                printf("%s},{B,E,H},{D,E,H}", attr[exteriors[k]]);

            }

        }

    }

    getch();

}

Inputs & Outputs::

1:

Text

Description automatically generated

2:

Text

Description automatically generated