B⋈Tree

A mechanism to drive existing B⁺Trees to do Join Internally.

B⋈TREE Index

To understand how B⋈TREE Index works let see what happens when we insert a new Row R_m from Table T_i into the database.

Suppose that table T_i is in Direct Join with a table T_k, we have to look for all the Rows $R_n \ldots R_z$ in T_k that satisfy the join condition with R_m and insert Rows references to $R_m R_n \ldots R_m R_z$ in the virtual table T_{ik}.

The process should be repeated for $R_m R_n \ldots R_m R_z$ with a table in join at least with one of the base tables constituting the Virtual Join Table T_{ij} and so on until we scan a path in the sequence of tables in join.

Transformation of Existing B+Tree

- The internal definition for the creation of a B+Tree take in consideration the following:
  - Name of B+Tree index follow by an index
  - Number of base tables constituting the virtual table indexed by the B+Tree
  - Length and type of Keys
  - Length and type of Inherited Keys (They are supplementary fields inserted in the B+Tree but they are not part of the key and they are not used for comparison)

- Declare the page of B+Tree as a buffer of bytes and divide it as needed. Many existing B+Tree follow this technique to support different type of multiple columns Key.

- The Leaf Page structure consists of:
  - Pointer to the previous sibling page
  - number of elements in which everyone consists of:
    - Space for the columns forming the keys
    - Space for the Data Pointers (Row Ids) to reference the Row in every table
    - Space for the columns forming the Inherited Keys
  - Pointer to the next sibling Page

Transformation of Existing B+Tree (continue)

- The Non Leaf Page structure consists of:
  - Pointer to a child page which key values are smaller than all the keys in the page
  - number of elements in which everyone consists of:
    - Space for the columns forming the keys
    - Pointer to a child page which key values are bigger than the keys in the Element

- Due to the fact that many join keys are duplicates, change has been made for the duplicates in the sense when 2 keys are equals, we consider the data references for them. The B+Tree keeps these possibly duplicated keys separate internally by combining the unique sequence of data references with each key. The process of combination is done logically, and requires no additional space for key storage.

Many advanced B+Tree in the market use (Key, Data Reference) combination to refer to unique Row eliminating duplicates internally and use additional fields others than the one forming the key to avoid access to the table.

So for those B+Trees, the only modification is instead of space of one Data Reference is a space for multiple Data Reference Space.

Definitions

- Base Table:

  Base tables are database objects whose structure and the data they contain are both on disk.

- Virtual Table:

  Virtual tables are tables whose contents are derived from base tables. Only its definition (base tables Names constituting it) is stored on disk.

Definitions

Direct Join:

Two tables are in Direct Join if there is a link between them (in other sense if there is common columns between them).

Join Graph:

A graph representing direct join between tables.

Adjacency List:

List for every table T_i in the database all those tables in direct Join with it.

Generating Join Graph

- Base Tables represent the vertexes of the Join Graph.
- Due to the fact that join is commutative, for every pair of tables in direct join between them as defined by DBA create an undirected edge to link them.
- It is very easy to knows which tables are in direct join with others tables from the definition of common columns between them.

The algorithm for generating the Linked List representation of the join Graph is the following:

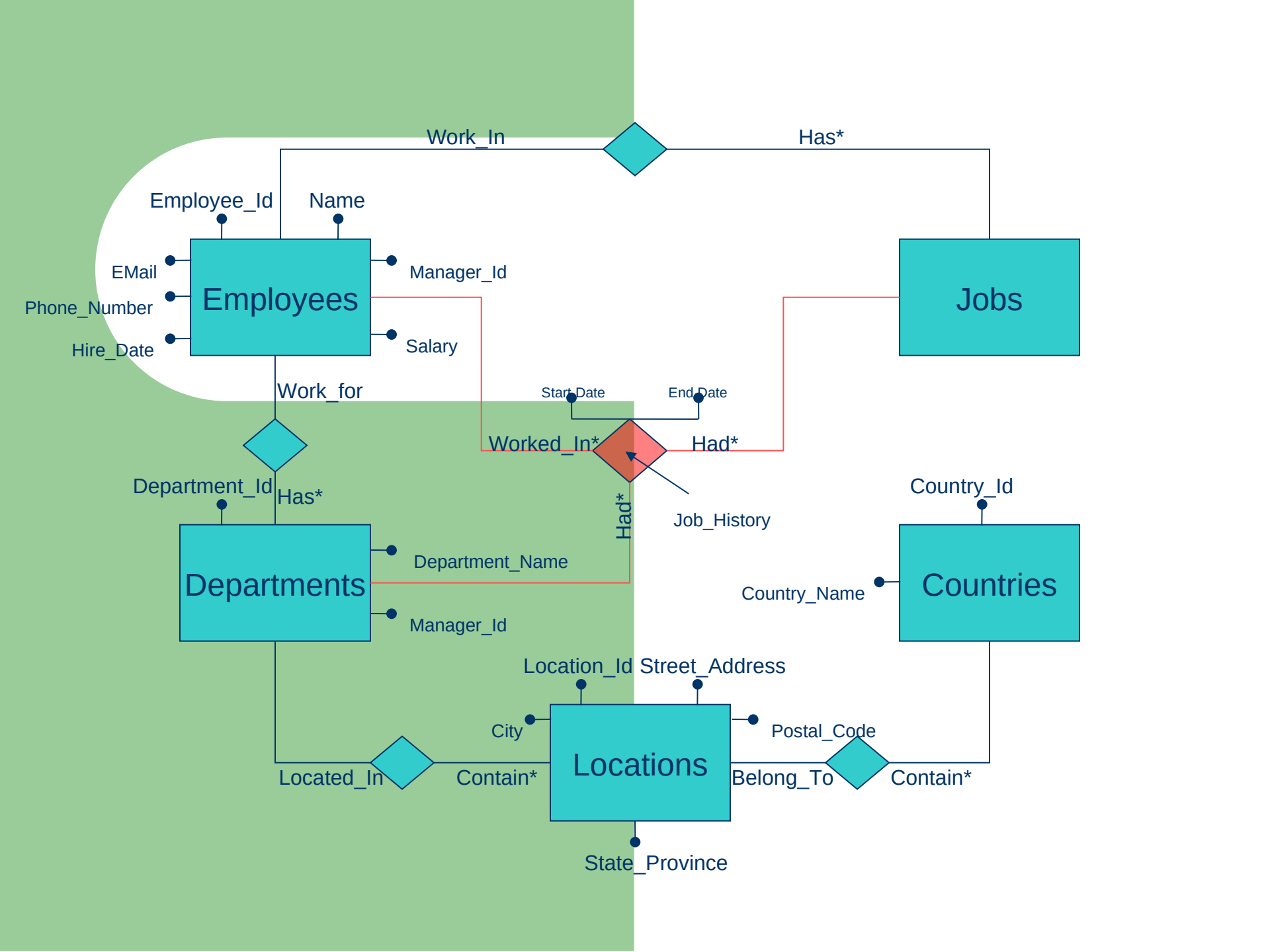
generateJoinGraph (in BaseTables; out JoinGraph)

insert the base tables as vertexes of the graph

for every direct join between 2 tables T_i and T_k where T_i is the table of order i and T_k is the table of order k as defined by the DBA do

AdjacentList[T_i] += T_k follow by the common key

AdjacentList[T_k] += T_i follow by the common key


Work_In
Has*
Employee_Id
Name
EMail
Phone_Number
Hire_Date
Employees
Manager_Id
Salary
Jobs
Work_for
Start_Date
End_Date
Worked_In*
Had*
Department_Id
Has*
Had*
Job_History
Country_Id
Department_Name
Departments
Countries
Country_Name
Manager_Id
Location_Id
Street_Address
City
Postal_Code
Located_In
Contain*
Locations
Belong_To
Contain*
State_Province

generateJoinGraph (in BaseTables; out JoinGraph)

insert the base tables as vertexes of the graph

for every direct join between 2 tables of the form T_i and T_k where T_i is the table of order i and T_k is the table of order k as defined by the DBA do

AdjacentList[T_i] += T_k follow by the common key

AdjacentList[T_k] += T_i follow by the common key

Base Tables

Employees	Job_History	Jobs	Departments	Locations	Countries
0	1	2	3	4	5

generateJoinGraph (in BaseTables; out JoinGraph)

insert the base tables as vertexes of the graph

for every direct join between 2 tables of the form T_i and T_k where T_i is the table of order i and T_k is the table of order k as defined by the DBA do

AdjacentList[T_i] += T_k follow by the common key

AdjacentList[T_k] += T_i follow by the common key

Base Tables

Employees	Job_History	Jobs	Departments	Locations	Countries
0	1	2	3	4	5

generateJoinGraph (in BaseTables; out JoinGraph)

insert the base tables as vertexes of the graph

for every direct join between 2 tables of the form T_i and T_k where T_i is the table of order i and T_k is the table of order k as defined by the DBA do

AdjacentList[T_i] += T_k follow by the common key

AdjacentList[T_k] += T_i follow by the common key

Base Tables

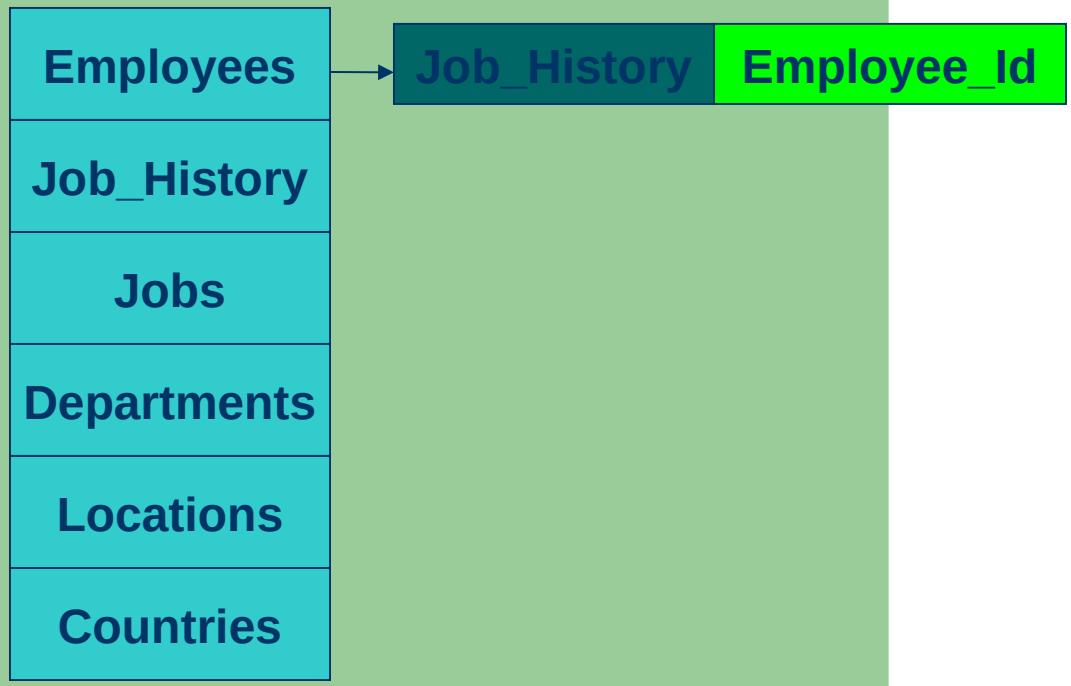
Employees	Job_History	Jobs	Departments	Locations	Countries
0	1	2	3	4	5

Employees
Job_History
Jobs
Departments
Locations
Countries

generateJoinGraph (in BaseTables; out JoinGraph)
insert the base tables as vertexes of the graph
for every direct join between 2 tables of the form T_i and T_k where T_i is the table of order i and T_k is the table of order k as defined by the DBA do

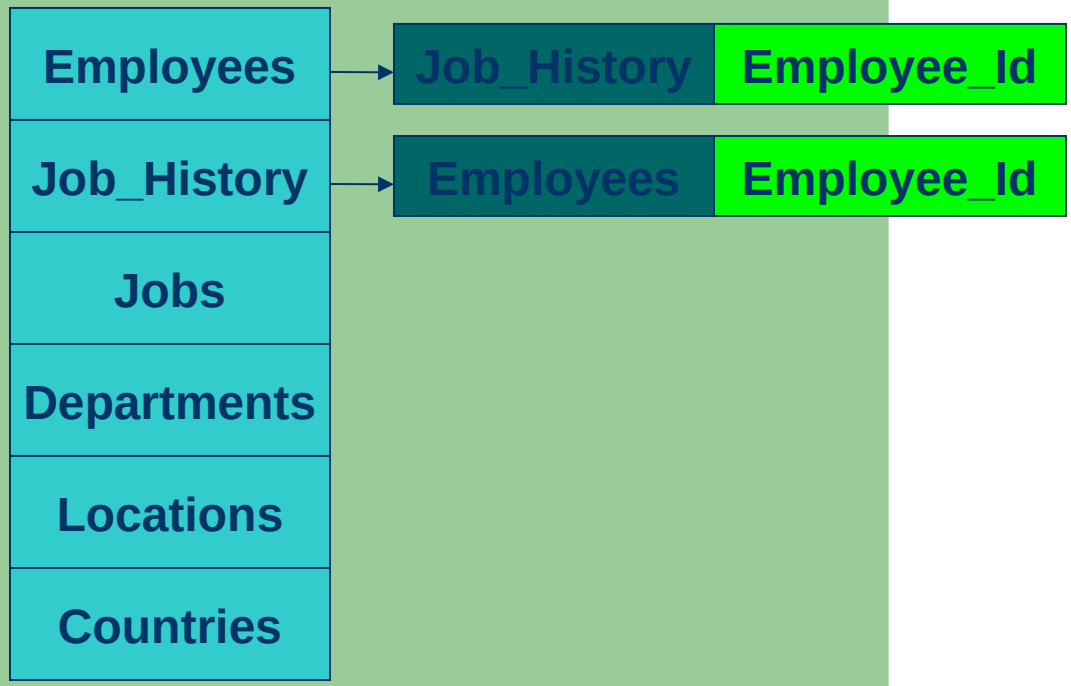
- **AdjacentList[T_i] += T_k follow by the common key**
- **AdjacentList[T_k] += T_i follow by the common key**

Base Tables



generateJoinGraph (in BaseTables; out JoinGraph)
insert the base tables as vertexes of the graph
for every direct join between 2 tables of the form T_i and T_k where T_i is the table of order i and T_k is the table of order k as defined by the DBA do
AdjacentList[T_i] += T_k follow by the common key
AdjacentList[T_k] += T_i follow by the common key

Base Tables



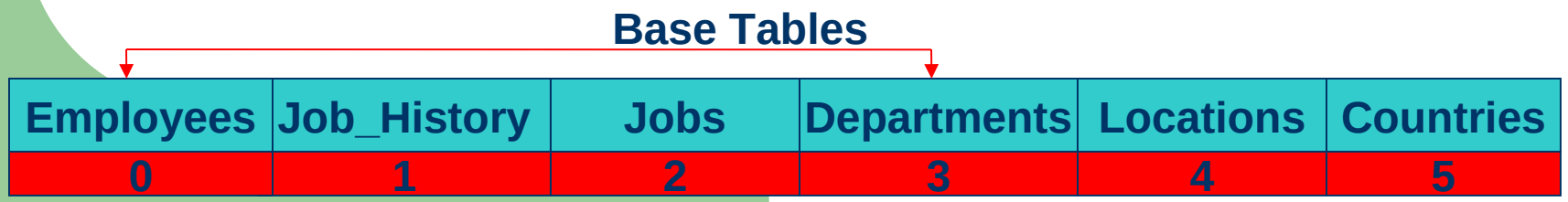
generateJoinGraph (in BaseTables; out JoinGraph)

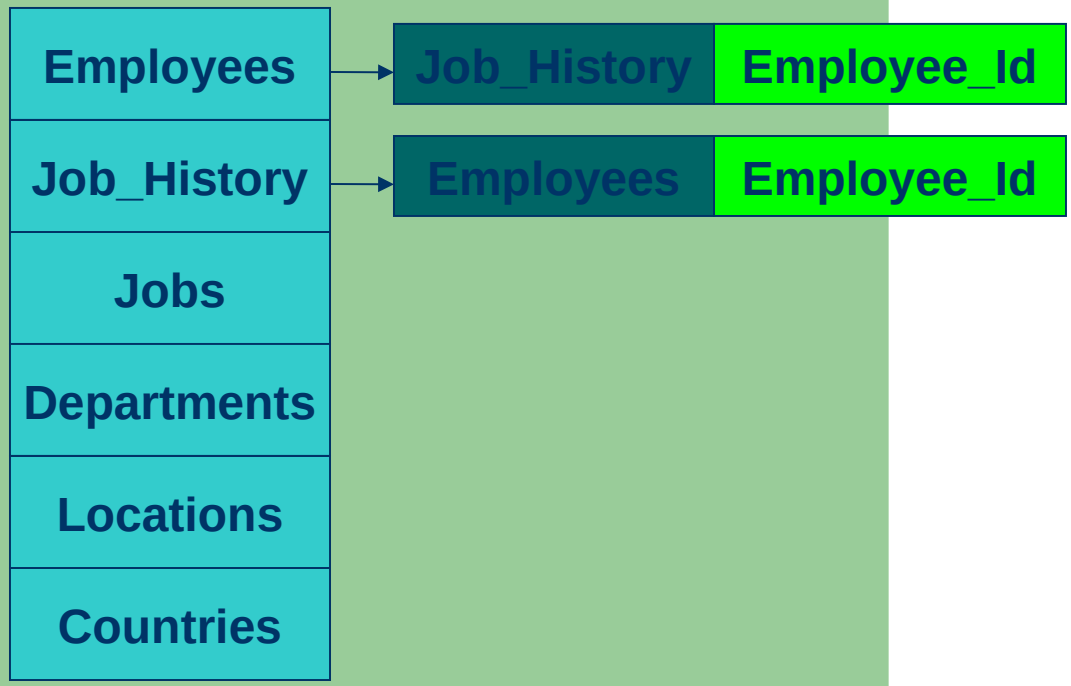
insert the base tables as vertexes of the graph

for every direct join between 2 tables of the form T_i and T_k where T_i is the table of order i and T_k is the table of order k as defined by the DBA do

AdjacentList[T_i] += T_k follow by the common key

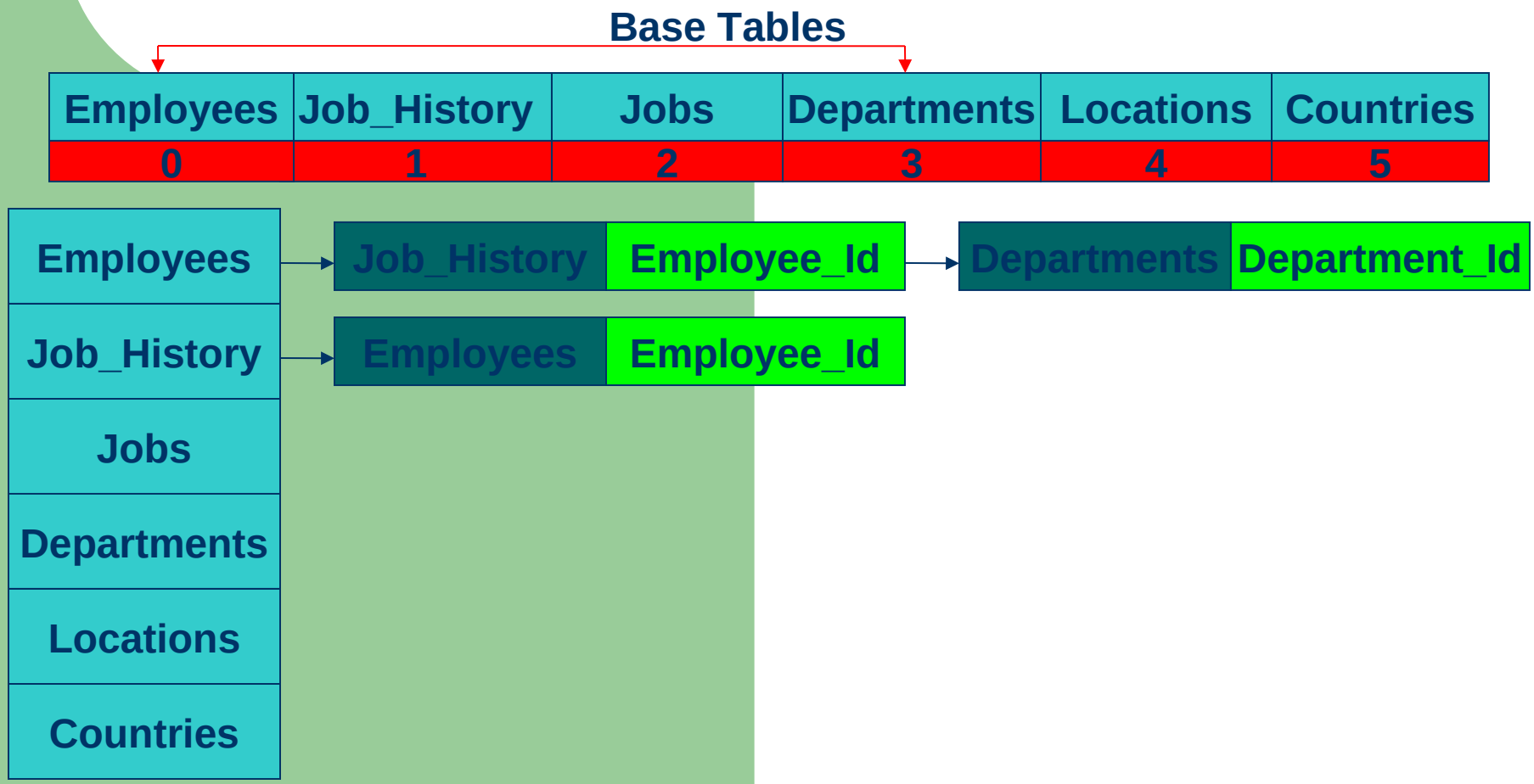
AdjacentList[T_k] += T_i follow by the common key





generateJoinGraph (in BaseTables; out JoinGraph)
insert the base tables as vertexes of the graph
for every direct join between 2 tables of the form T_i and T_k where T_i is the table of order i and T_k is the table of order k as defined by the DBA do

- **AdjacentList[T_i] += T_k follow by the common key**
- **AdjacentList[T_k] += T_i follow by the common key**



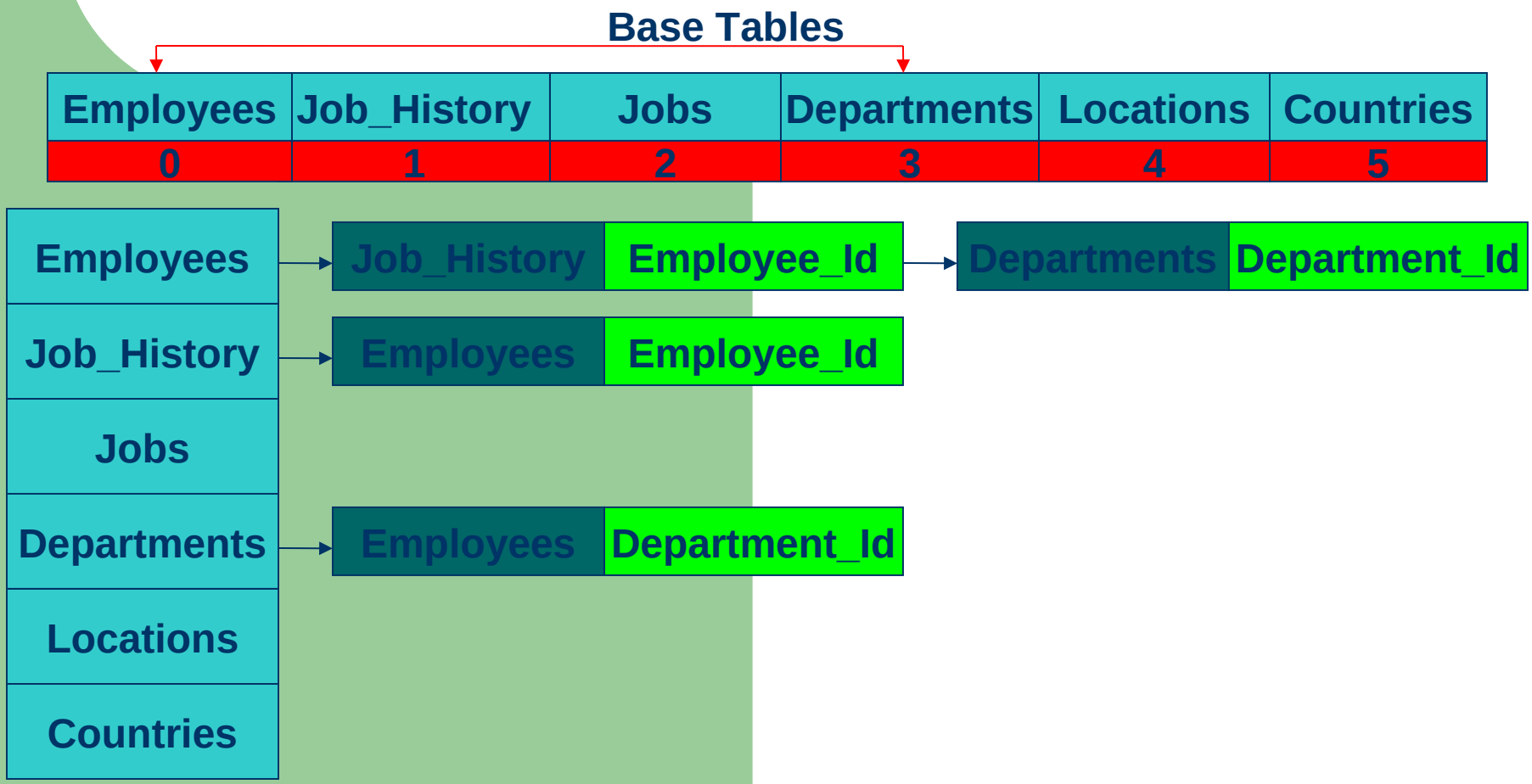
generateJoinGraph (in BaseTables; out JoinGraph)

insert the base tables as vertexes of the graph

for every direct join between 2 tables of the form T_i and T_k where T_i is the table of order i and T_k is the table of order k as defined by the DBA do

→ **AdjacentList[T_i] += T_k follow by the common key**

AdjacentList[T_k] += T_i follow by the common key



generateJoinGraph (in BaseTables; out JoinGraph)

insert the base tables as vertexes of the graph

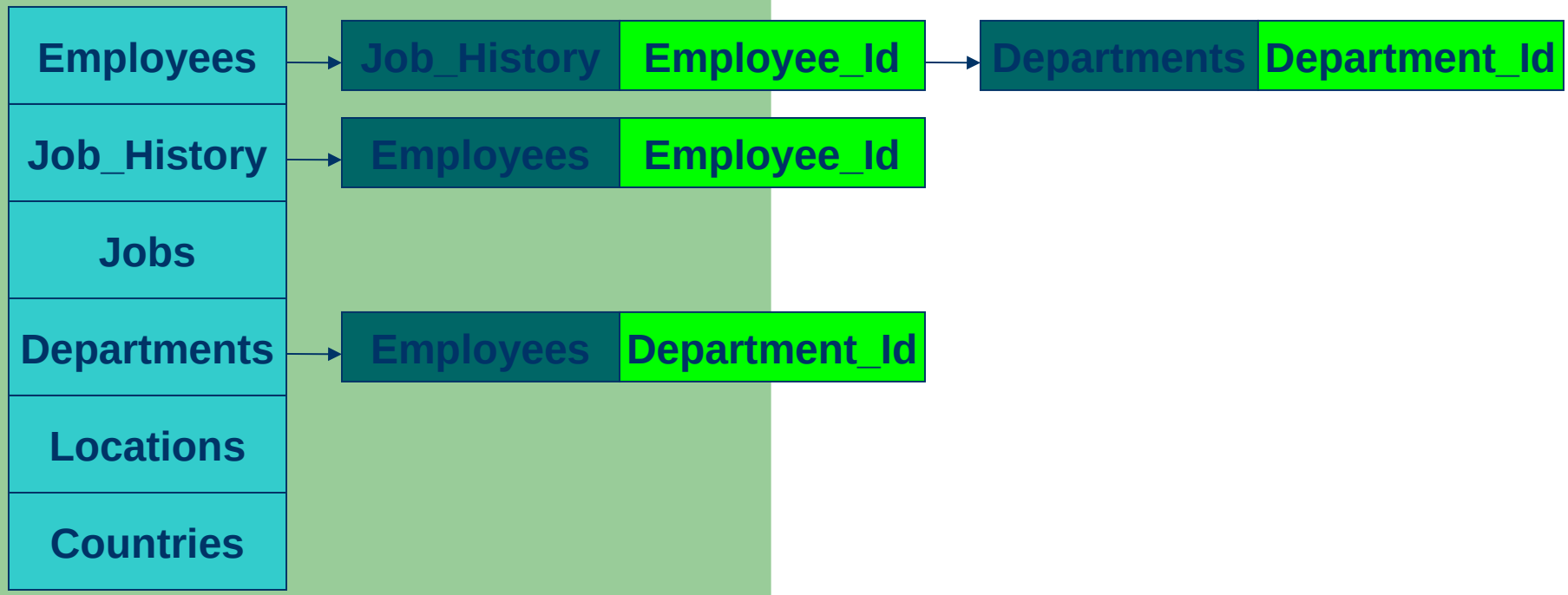
for every direct join between 2 tables of the form T_i and T_k where T_i is the table of order i and T_k is the table of order k as defined by the DBA do

AdjacentList[T_i] += T_k follow by the common key

AdjacentList[T_k] += T_i follow by the common key

Base Tables

Employees	Job_History	Jobs	Departments	Locations	Countries
0	1	2	3	4	5

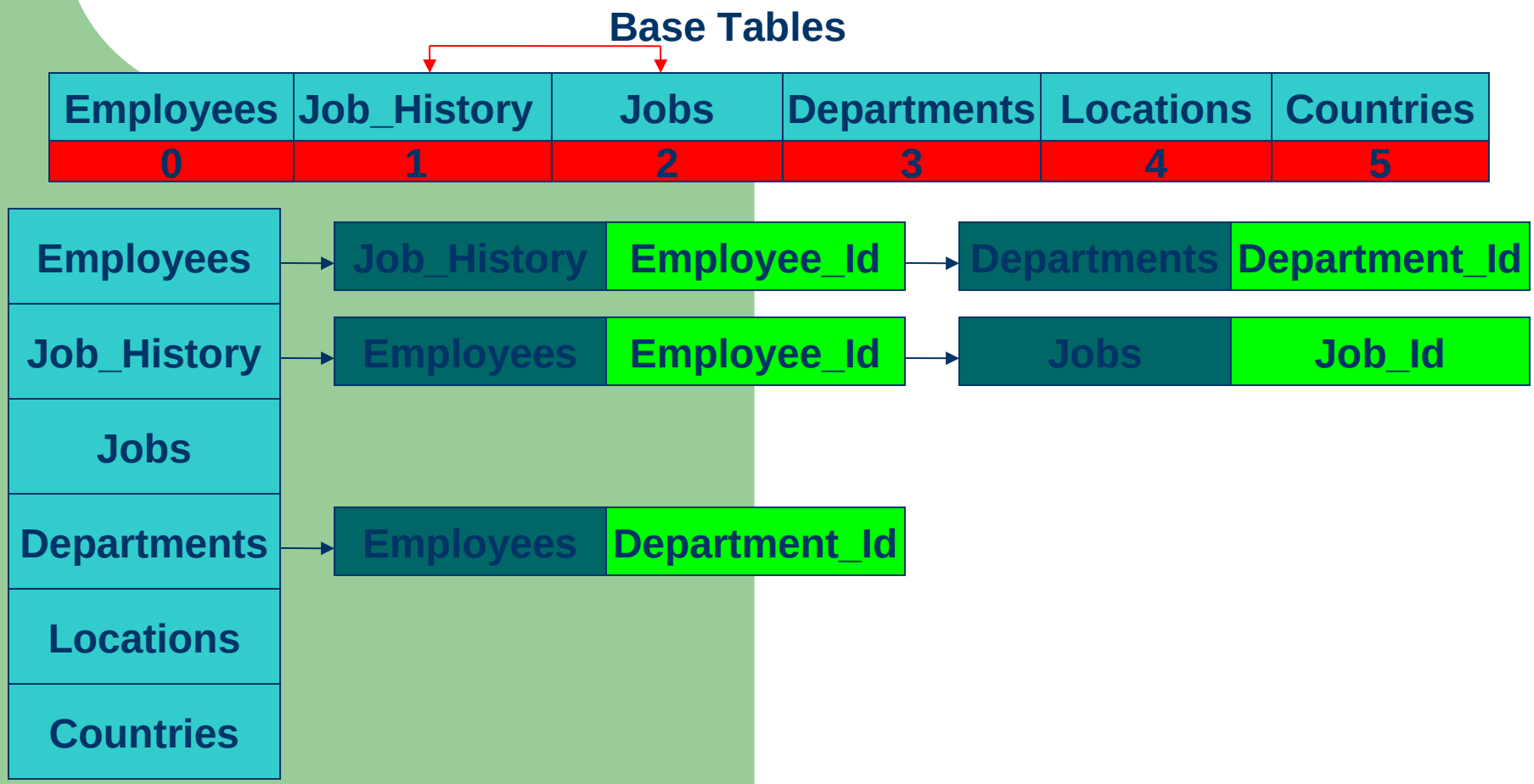


generateJoinGraph (in BaseTables; out JoinGraph)

insert the base tables as vertexes of the graph

for every direct join between 2 tables of the form T_i and T_k where T_i is the table of order i and T_k is the table of order k as defined by the DBA do

- **AdjacentList[T_i] += T_k follow by the common key**
- **AdjacentList[T_k] += T_i follow by the common key**



generateJoinGraph (in BaseTables; out JoinGraph)

insert the base tables as vertexes of the graph

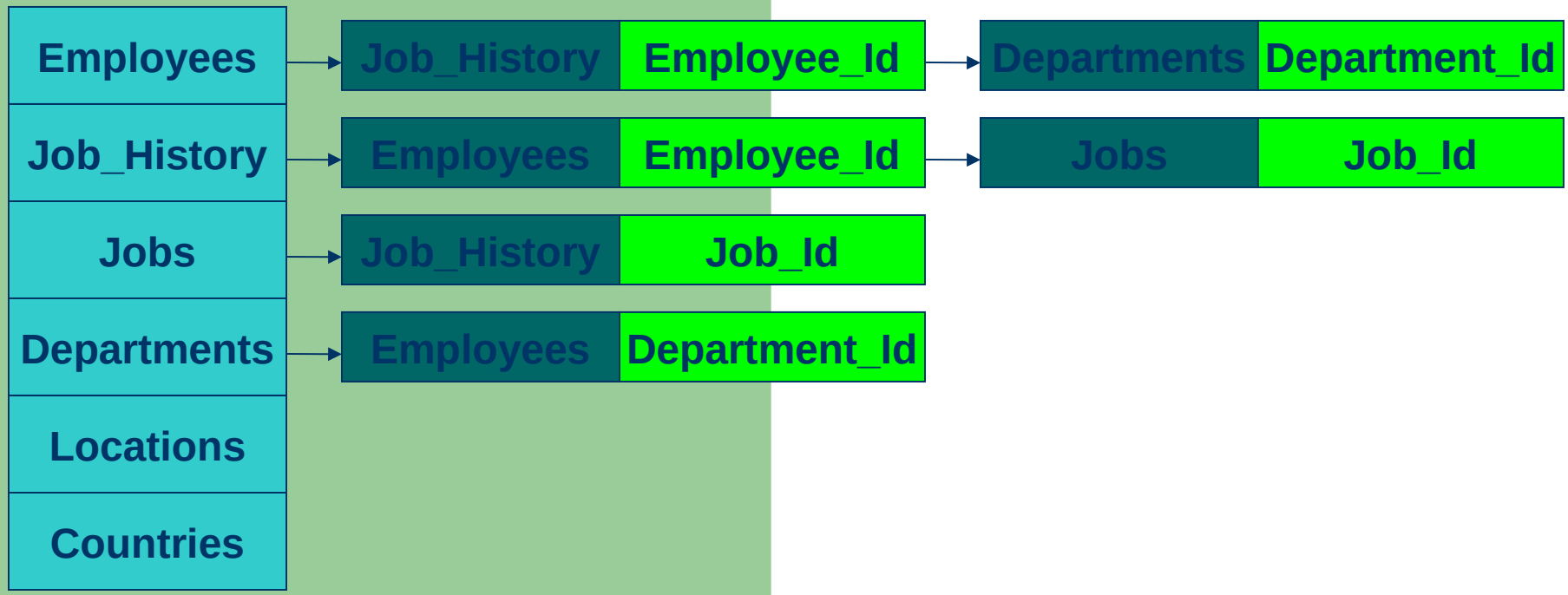
for every direct join between 2 tables of the form T_i and T_k where T_i is the table of order i and T_k is the table of order k as defined by the DBA do

→ **AdjacentList[T_i] += T_k follow by the common key**

AdjacentList[T_k] += T_i follow by the common key

Base Tables

Employees	Job_History	Jobs	Departments	Locations	Countries
0	1	2	3	4	5



generateJoinGraph (in BaseTables; out JoinGraph)

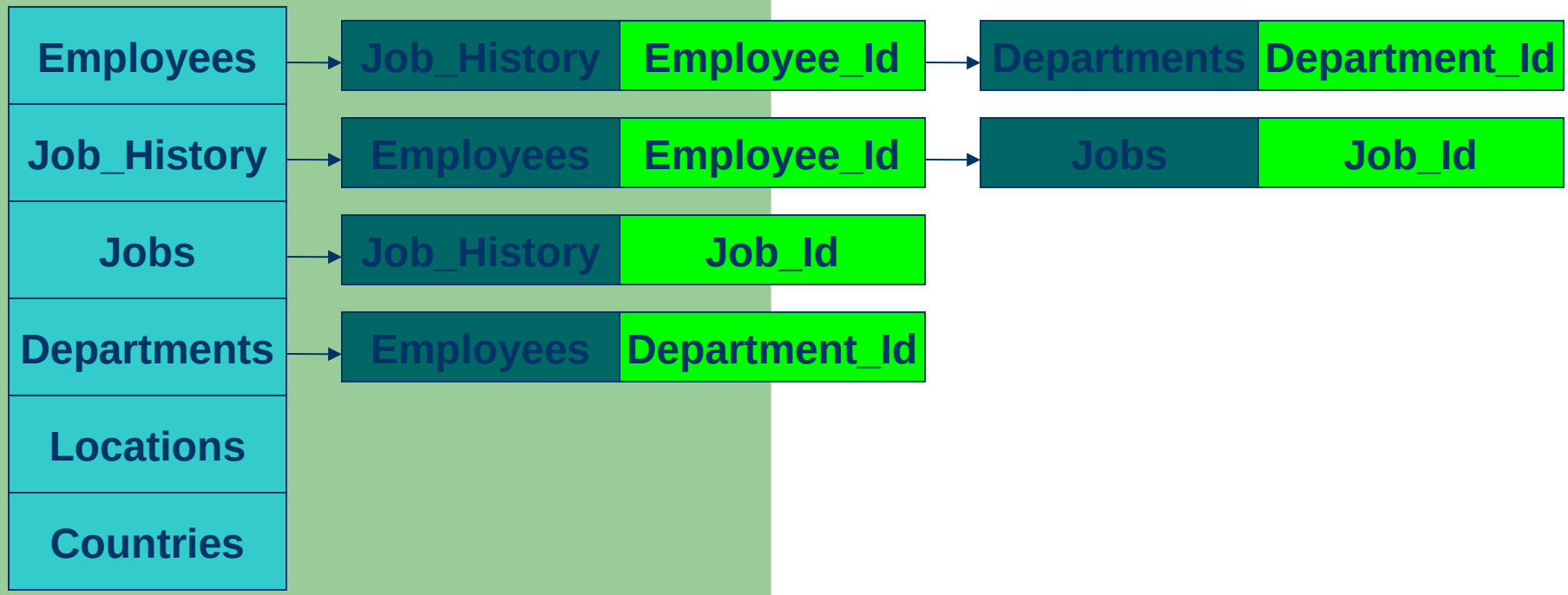
insert the base tables as vertexes of the graph

for every direct join between 2 tables of the form T_i and T_k where T_i is the table of order i and T_k is the table of order k as defined by the DBA do

AdjacentList[T_i] += T_k follow by the common key

AdjacentList[T_k] += T_i follow by the common key

Base Tables



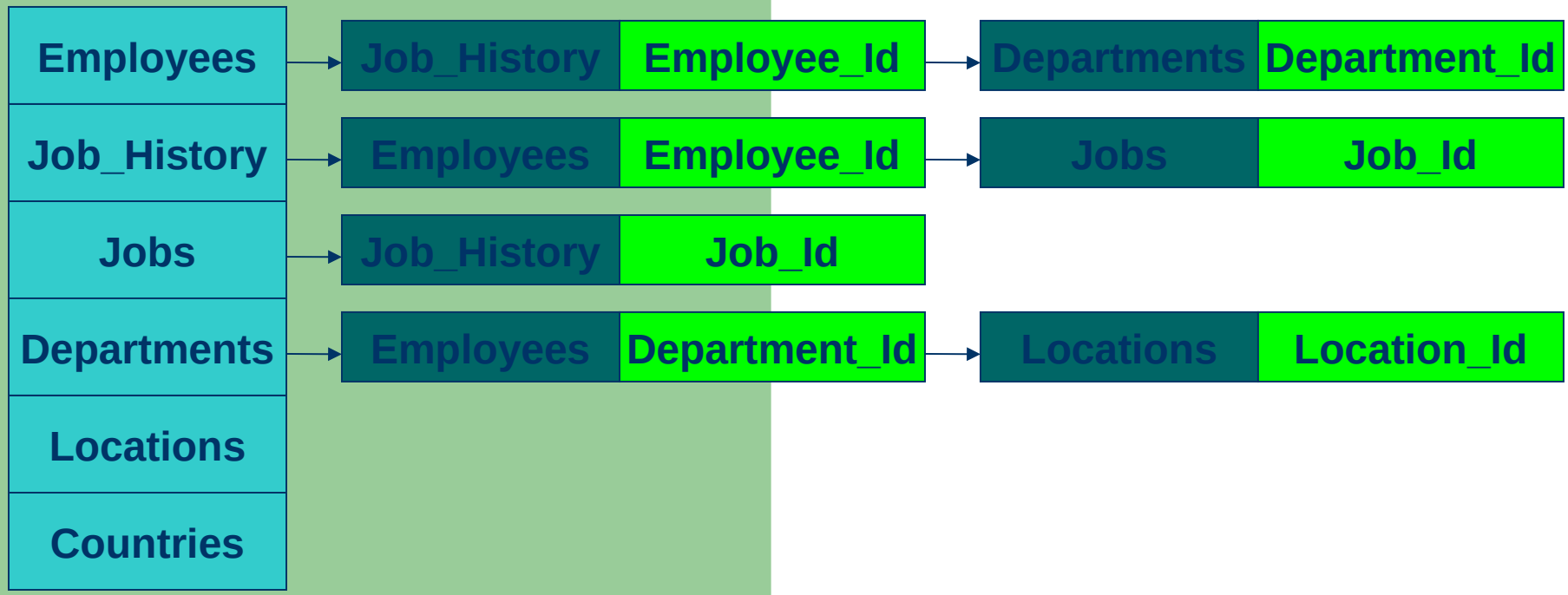
generateJoinGraph (in BaseTables; out JoinGraph)

insert the base tables as vertexes of the graph

for every direct join between 2 tables of the form T_i and T_k where T_i is the table of order i and T_k is the table of order k as defined by the DBA do

- **AdjacentList[T_i] += T_k follow by the common key**
- **AdjacentList[T_k] += T_i follow by the common key**

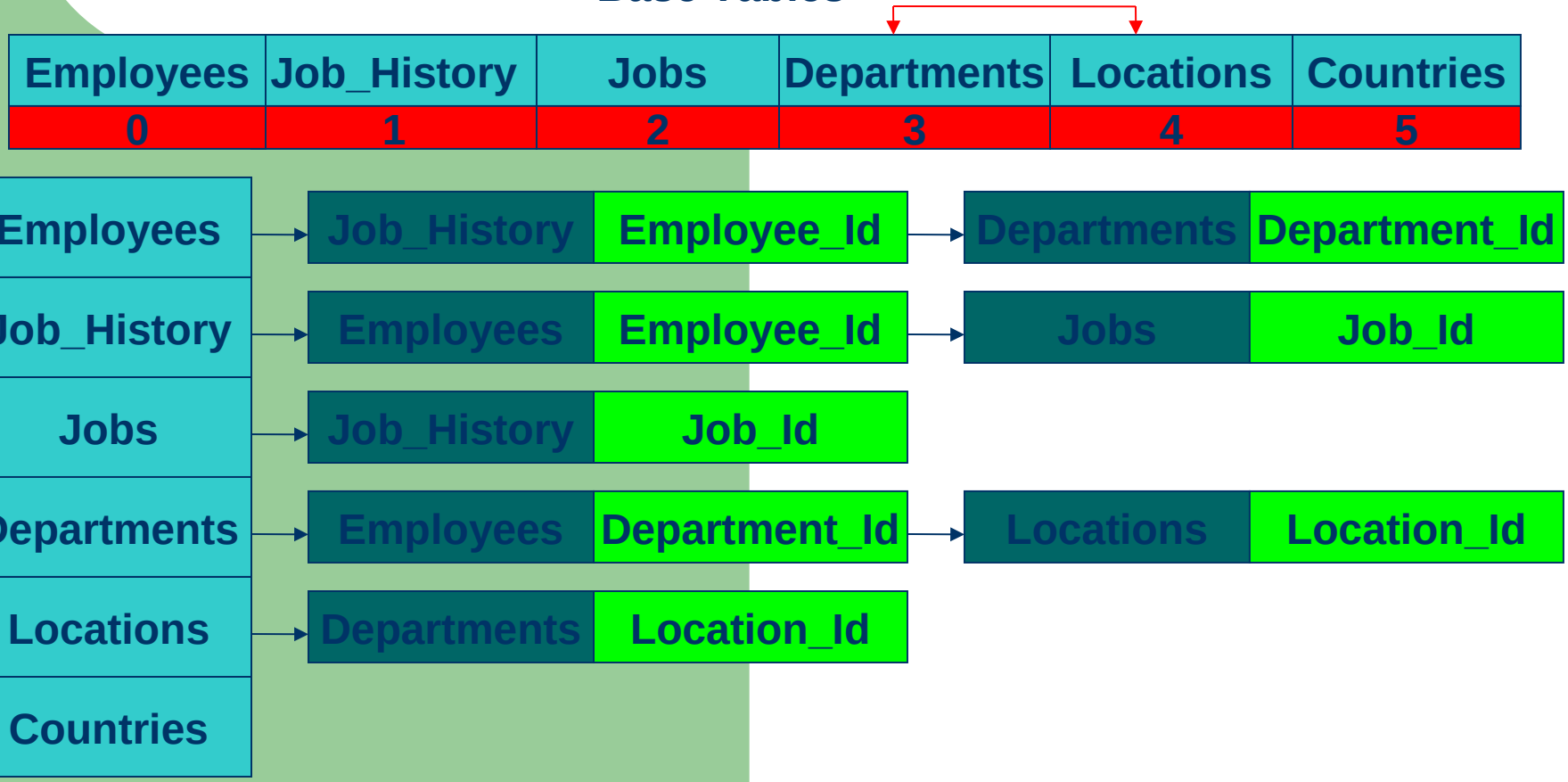
Base Tables



generateJoinGraph (in BaseTables; out JoinGraph)
insert the base tables as vertexes of the graph
for every direct join between 2 tables of the form T_i and T_k where T_i is the table of order i and T_k is the table of order k as defined by the DBA do

→ **AdjacentList[T_i] += T_k follow by the common key**
AdjacentList[T_k] += T_i follow by the common key

Base Tables



generateJoinGraph (in BaseTables; out JoinGraph)

insert the base tables as vertexes of the graph

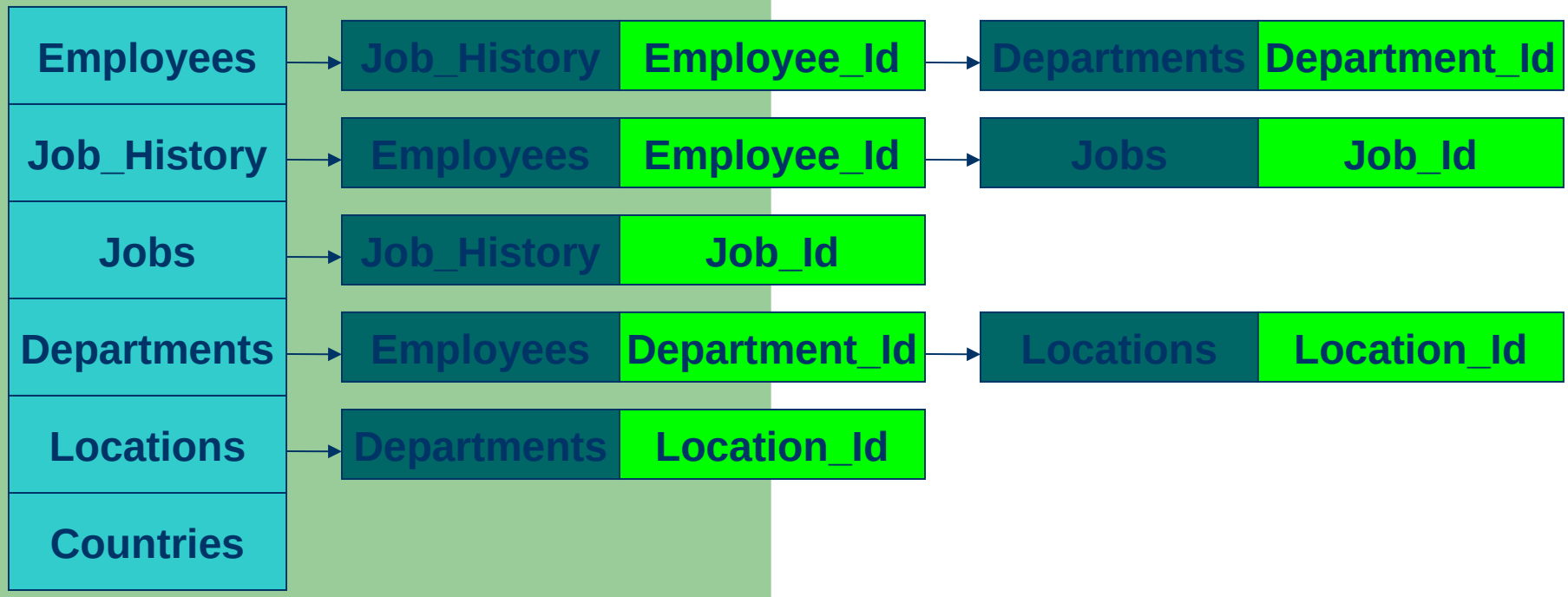
for every direct join between 2 tables of the form T_i and T_k where T_i is the table of order i and T_k is the table of order k as defined by the DBA do

AdjacentList[T_i] += T_k follow by the common key

AdjacentList[T_k] += T_i follow by the common key

Base Tables

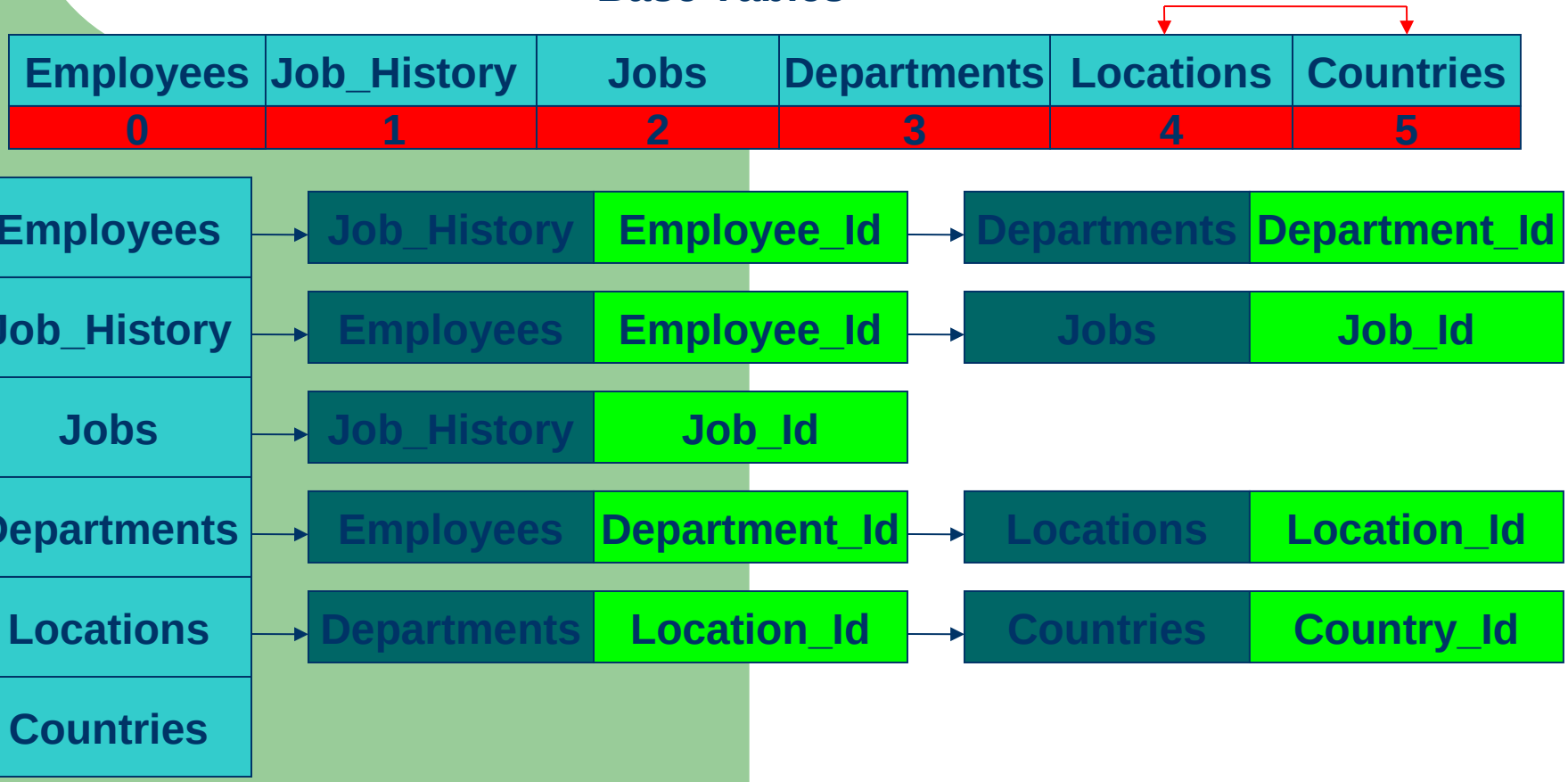
Employees	Job_History	Jobs	Departments	Locations	Countries
0	1	2	3	4	5



generateJoinGraph (in BaseTables; out JoinGraph)
insert the base tables as vertexes of the graph
for every direct join between 2 tables of the form T_i and T_k where T_i is the table of order i and T_k is the table of order k as defined by the DBA do

- **AdjacentList[T_i] += T_k follow by the common key**
- **AdjacentList[T_k] += T_i follow by the common key**

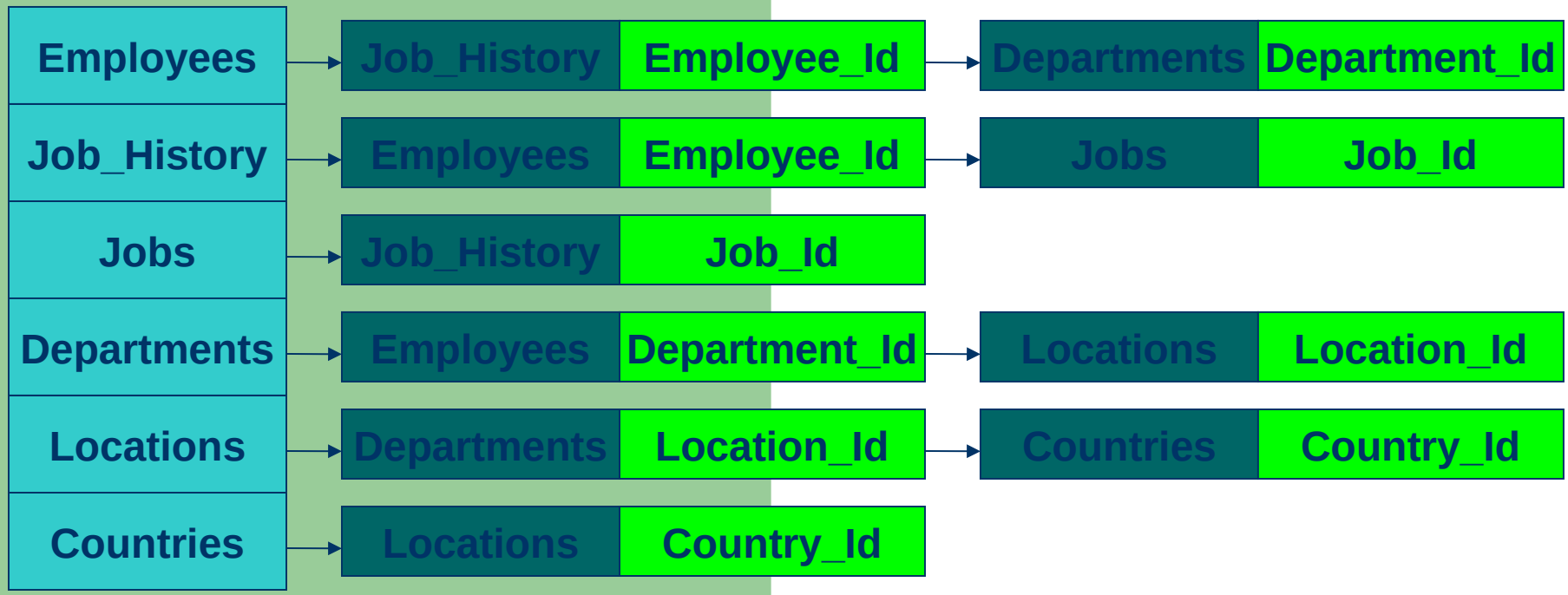
Base Tables

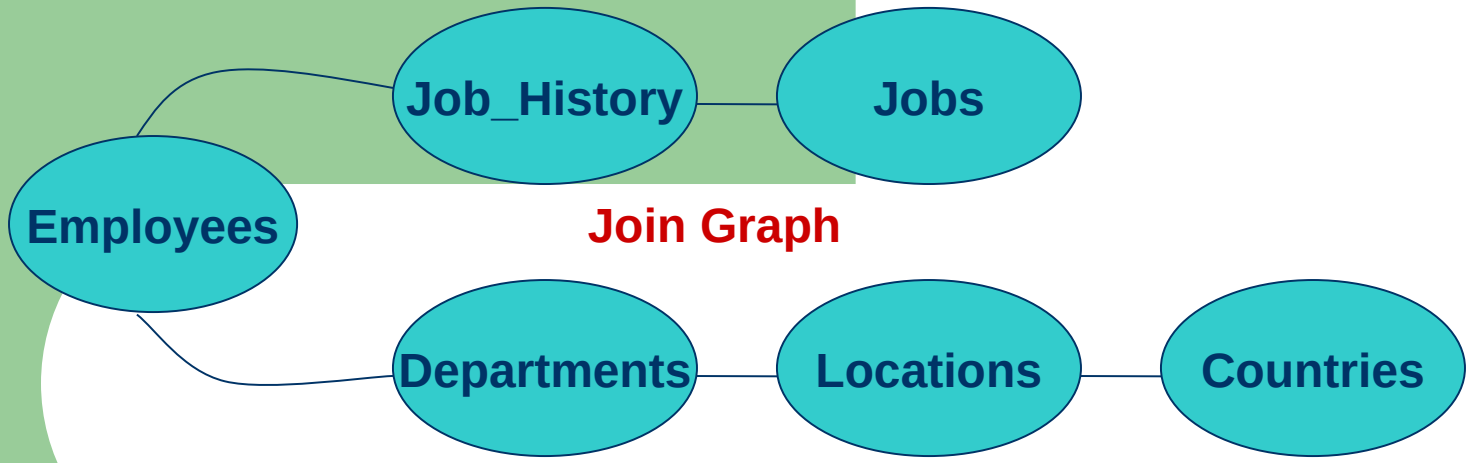


generateJoinGraph (in BaseTables; out JoinGraph)
insert the base tables as vertexes of the graph
for every direct join between 2 tables of the form T_i and T_k where T_i is the table of order i and T_k is the table of order k as defined by the DBA do

→ **AdjacentList[T_i] += T_k follow by the common key**
AdjacentList[T_k] += T_i follow by the common key

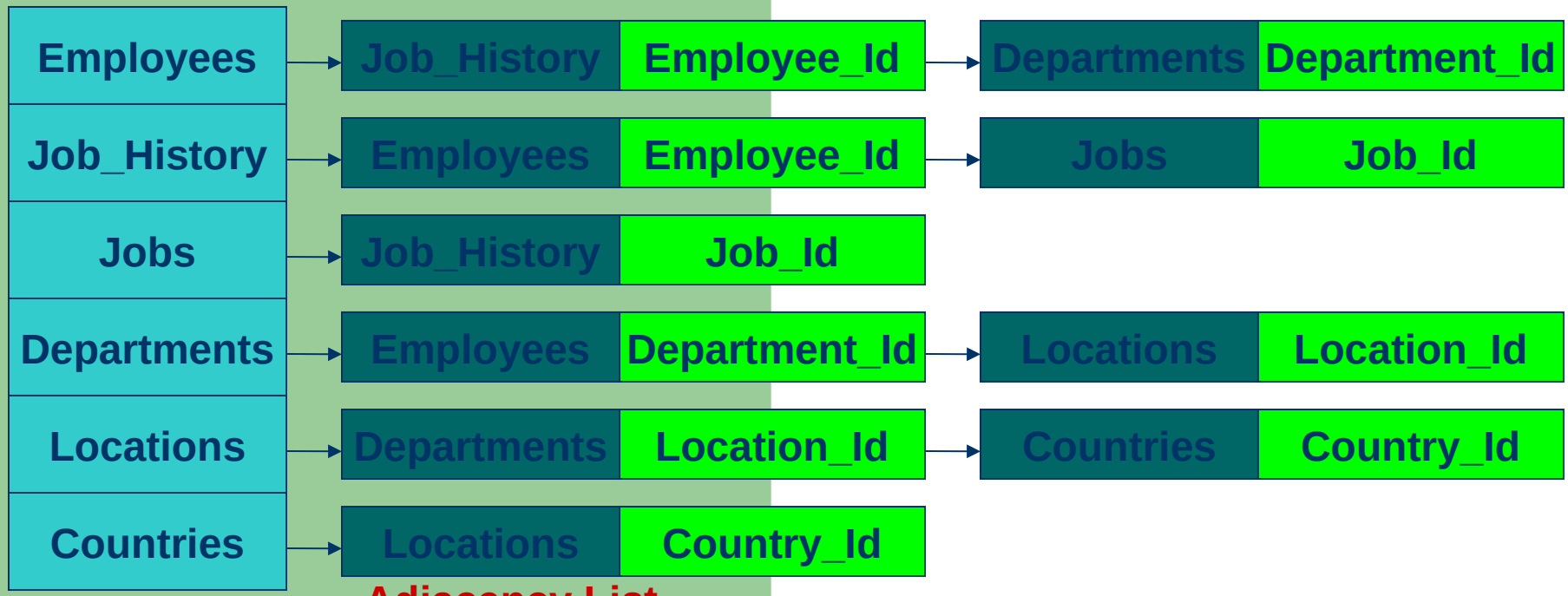
Base Tables





Join Graph

Linked List representation of the Join Graph



Adjacency List

Definitions

Join Path List:

A sequence of tables $T_0 \ldots T_{n-1}$ is in the Join Path List if every T_i of them is at least in direct join with another table in the sequence.

Notation

- When index i is not between brackets like in T_i, it represent a base table T_i.

When index i is between brackets like in $T_{[i]}$, it represent a base table T_i or a virtual table in which index i represent a set of indexes for the base tables forming the virtual table.

Steps to generate function: Key($T_{[j]}$) **getFirstAdjacentListKey($T_{[j]}$, $T_{[k]}$)**

for every Base Table T_l in $T_{[j]}$ do

Take one at a time

for every $T_{Link(l)}$ do

Take one at a time

if $T_{Link(l)}$ in $T_{[k]}$ then

return(key(T_l,$T_{Link(l)}$))

Normally one of the 2 tables $T_{[j]}$ or $T_{[k]}$ is a base table this is why we stop after founding the key.

Key could be a one column key or multicolumn key that satisfy the join condition.

Steps to generate Join Path List for the join sequence $T_0 \ldots T_m$

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty

Steps to generate Join Path List for the join sequence $T_0 \ldots T_m$ (continue)

insert all the names of base tables from path as vertexes in the JoinPathList

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

- **take one T_i at a time**
- **PathJoinAdjacentList(T_i) = $T_{[buf]}$**
- **Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)**
- **PathJoinAdjacentList($T_{[buf]}$) = T_i**
- **Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)**
- **$T_{[buf]}$ + = T_i**
- **Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$**

Steps to generate Join Path List for the join sequence $T_0 \ldots T_m$ (continue)

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

- **take one $T_{[i]}$ at a time**
- **for all Base Tables in$T_{[i]}$ do**
  - **take one T_k at a time**
  - **for every buf.Table = T_k do**
    - **if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then**
      InheritedKey($T_{[i]}$) += buf.key
- **if T_l is the table from which comes Key($T_{[i]}$) then**
  - **buf.Table = T_l**
  - **buf.key = Key($T_{[i]}$)**

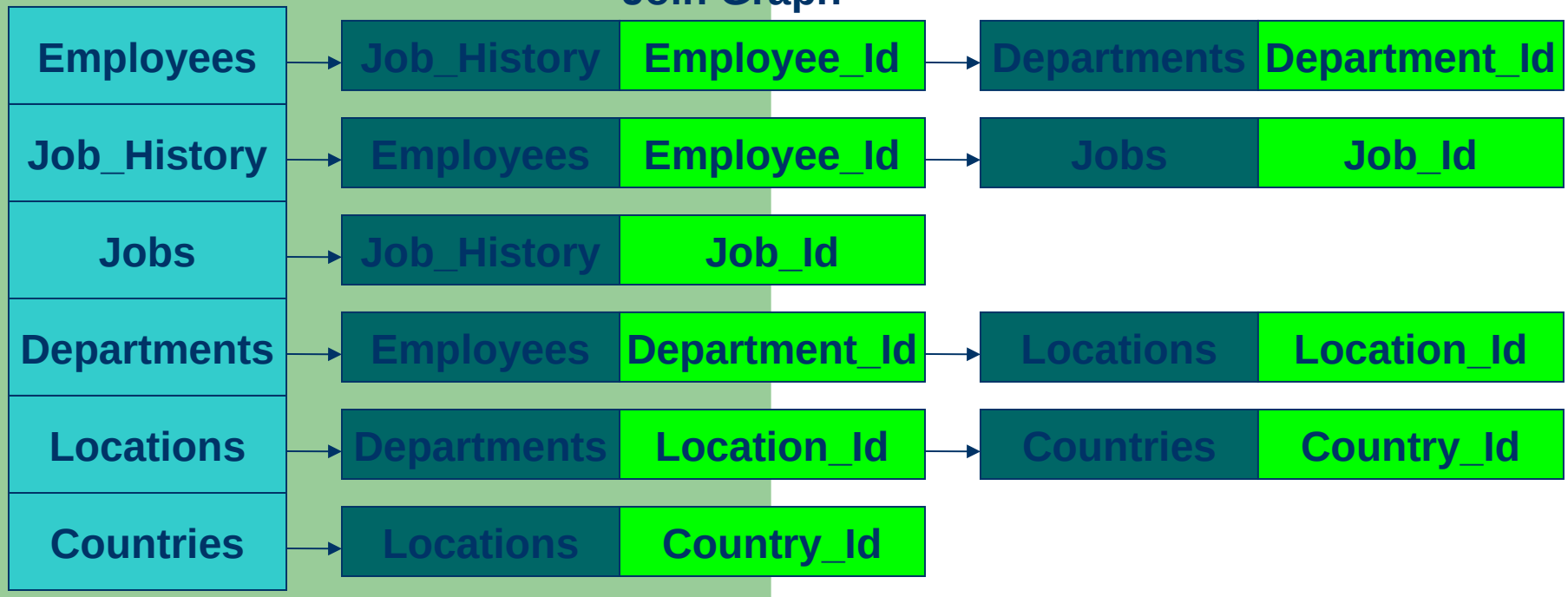
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty

Join Base Tables

Employees	Job_History	Jobs	Departments	Locations	Countries

Join Graph



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty

Join Base Tables

Employees	Job_History	Jobs	Departments	Locations	Countries
T_0	T_1	T_2	T_3	T_4	T_5

generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables

create 2 dynamic arrays queue and path

insert T_0 into path

insert T_0 into queue

repeat

    $T_{Element}$ = First Table in queue

    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do

        if the Link Item is in the join sequence then

            if path doesn't contain the Link Item then

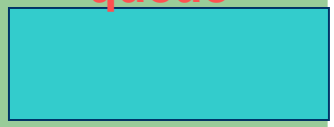
                insert Link Item into path

                insert Link Item into queue

    remove $T_{Element}$ from queue

until queue is empty

queue



path

generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables

create 2 dynamic arrays queue and path

insert T_0 into path

insert T_0 into queue

repeat

    $T_{Element}$ = First Table in queue

    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do

        if the Link Item is in the join sequence then

            if path doesn't contain the Link Item then

                insert Link Item into path

                insert Link Item into queue

    remove $T_{Element}$ from queue

until queue is empty

queue

path

Employees

generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables

create 2 dynamic arrays queue and path

insert T_0 into path

insert T_0 into queue

repeat

    $T_{Element}$ = First Table in queue

    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do

        if the Link Item is in the join sequence then

            if path doesn't contain the Link Item then

                insert Link Item into path

                insert Link Item into queue

    remove $T_{Element}$ from queue

until queue is empty

queue

Employees

path

Employees

generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty

queue

Employees

path

Employees

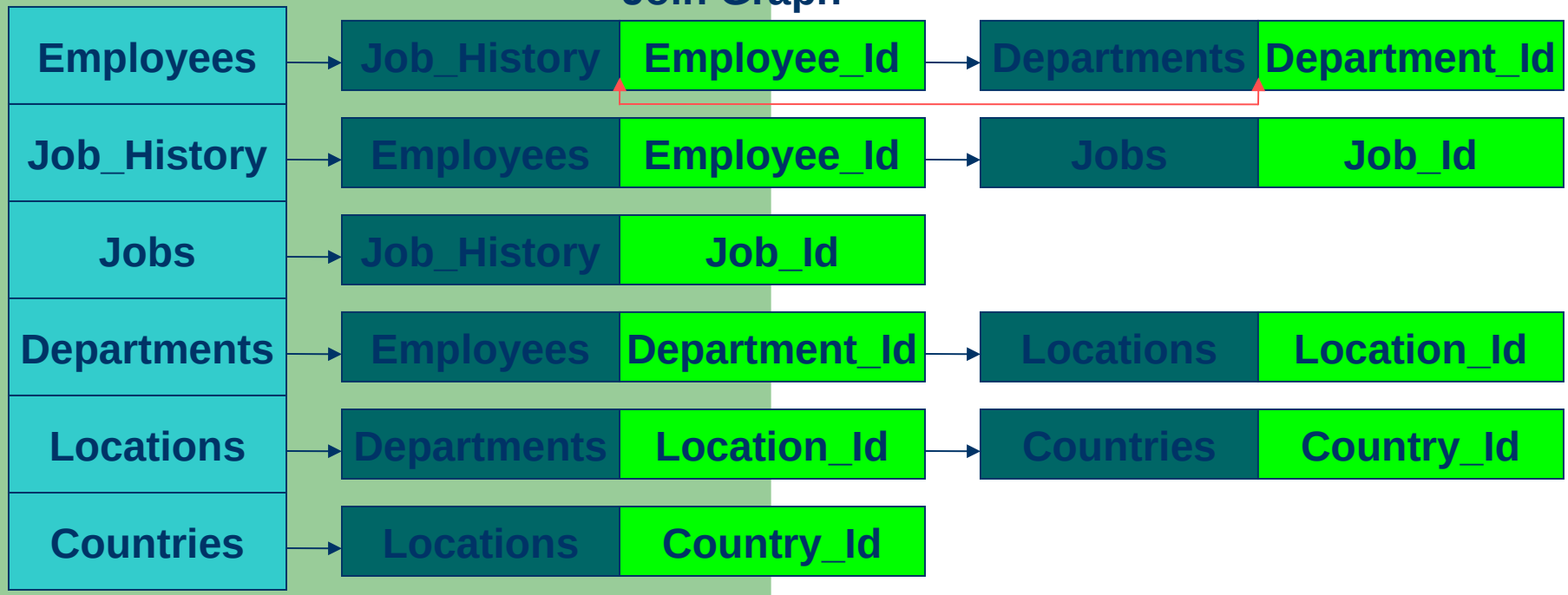
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty

generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty

Join Graph

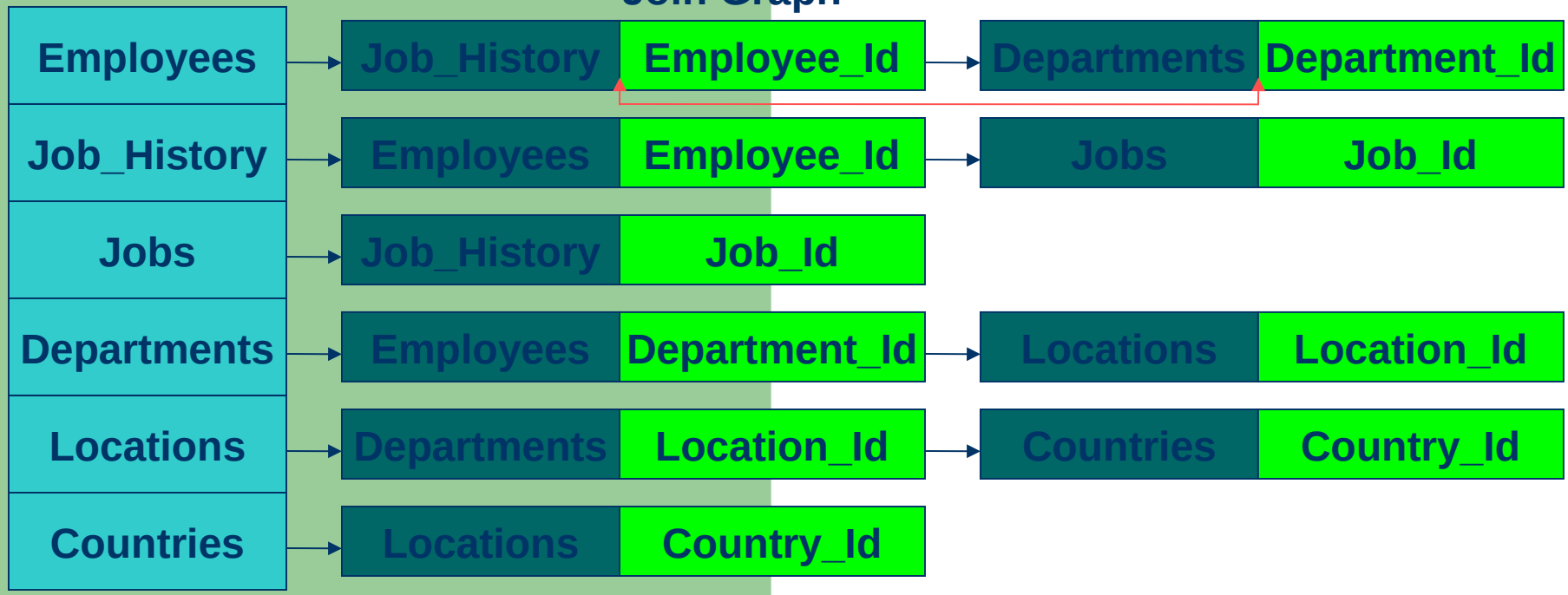


generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty

Join Base Tables

Join Graph



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables

create 2 dynamic arrays queue and path

insert T_0 into path

insert T_0 into queue

repeat

$T_{Element}$ = First Table in queue

for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do

if the Link Item is in the join sequence then

if path doesn't contain the Link Item then

insert Link Item into path

insert Link Item into queue

remove $T_{Element}$ from queue

until queue is empty

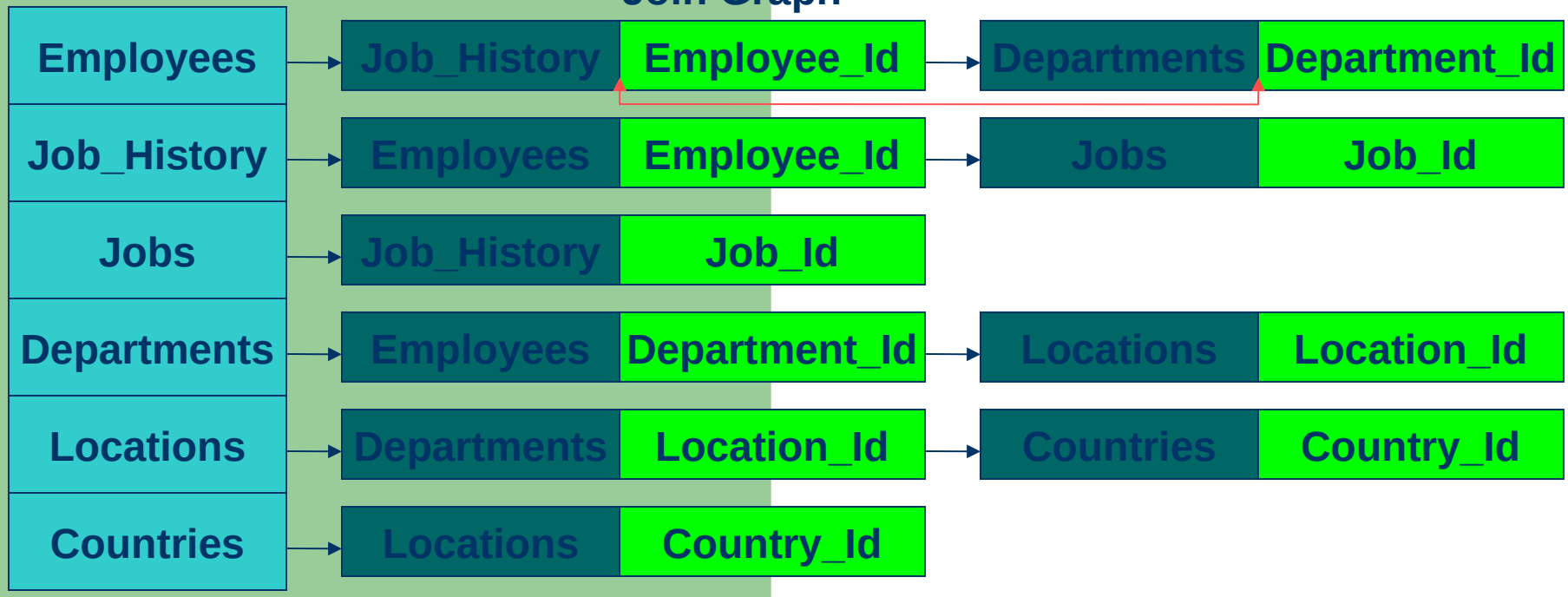
path

Employees

Join Base Tables

Employees	Job_History	Jobs	Departments	Locations	Countries

Join Graph



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables

create 2 dynamic arrays queue and path

insert T_0 into path

insert T_0 into queue

repeat

    $T_{Element}$ = First Table in queue

    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do

        if the Link Item is in the join sequence then

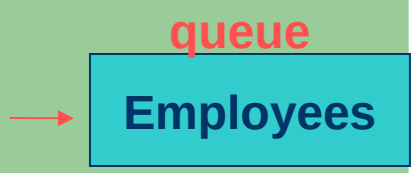
            if path doesn't contain the Link Item then

                insert Link Item into path

                insert Link Item into queue

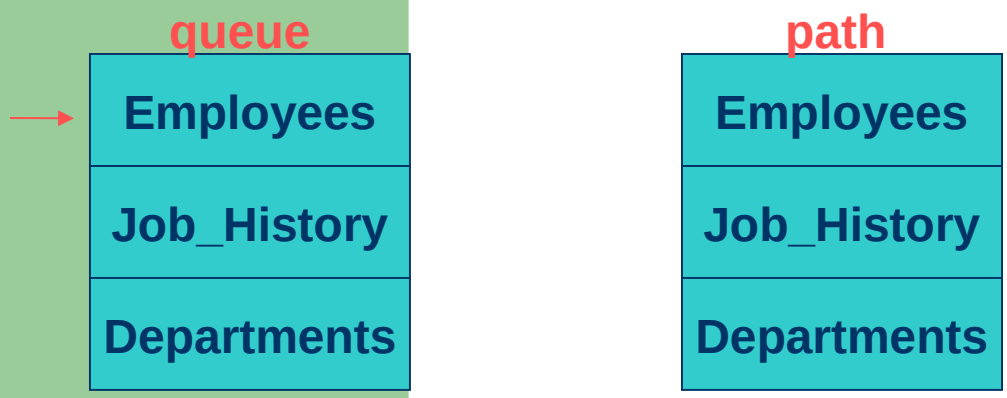
    remove $T_{Element}$ from queue

until queue is empty



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables

create 2 dynamic arrays queue and path

insert T_0 into path

insert T_0 into queue

repeat

    $T_{Element}$ = First Table in queue

    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do

        if the Link Item is in the join sequence then

            if path doesn't contain the Link Item then

                insert Link Item into path

                insert Link Item into queue

    remove $T_{Element}$ from queue

until queue is empty

queue

Job_History
Departments

path

Employees
Job_History
Departments

generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables

create 2 dynamic arrays queue and path

insert T_0 into path

insert T_0 into queue

repeat

    $T_{Element}$ = First Table in queue

    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do

        if the Link Item is in the join sequence then

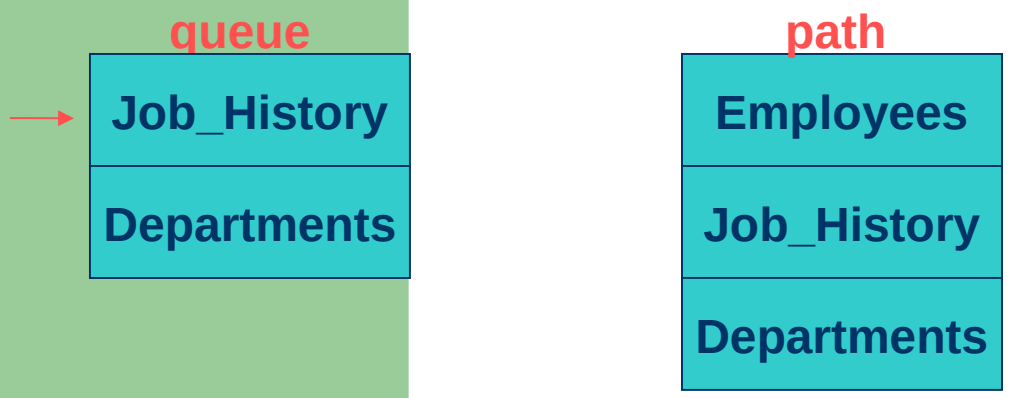
            if path doesn't contain the Link Item then

                insert Link Item into path

                insert Link Item into queue

    remove $T_{Element}$ from queue

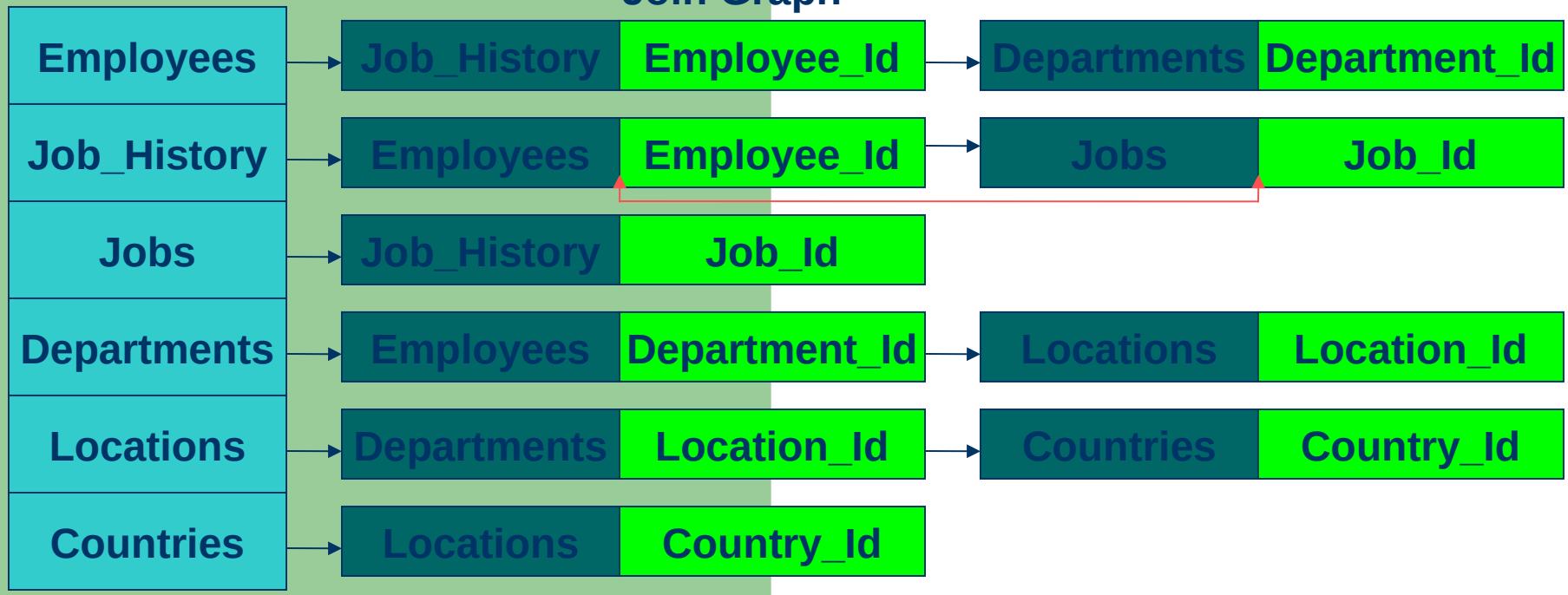
until queue is empty



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty

Join Graph

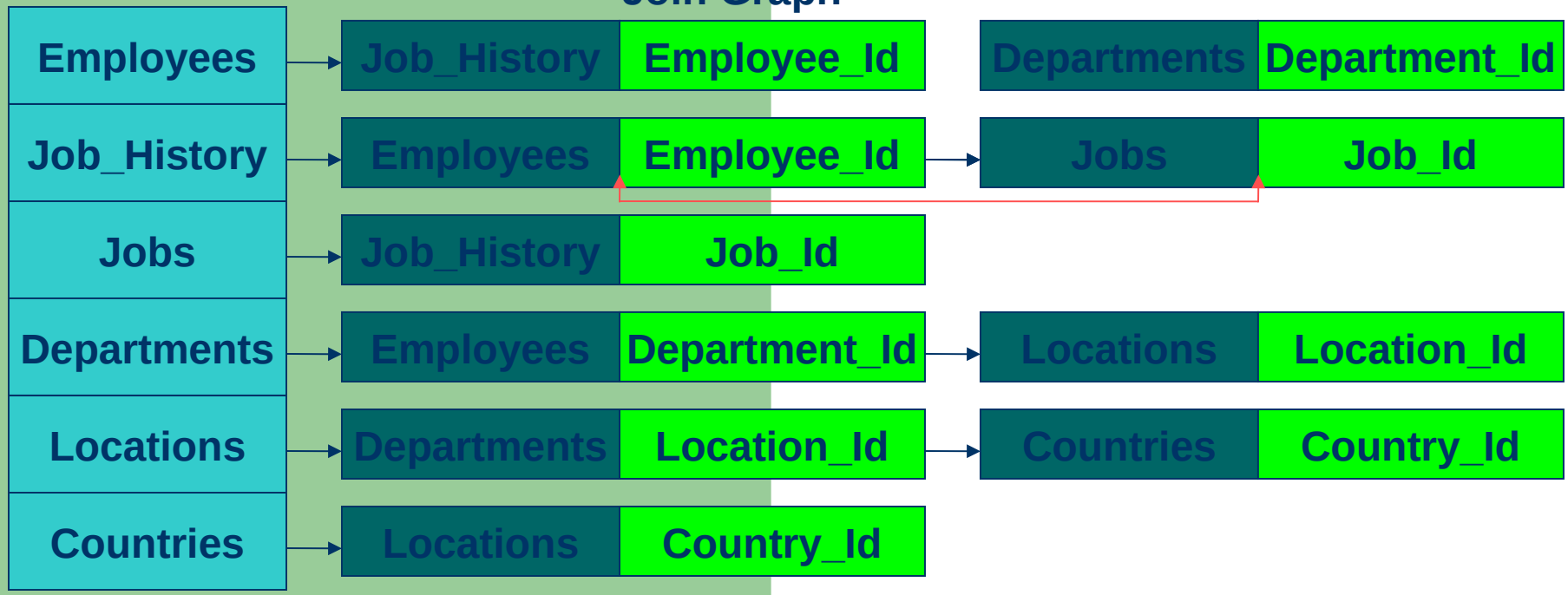


generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty

Join Base Tables

Join Graph



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty

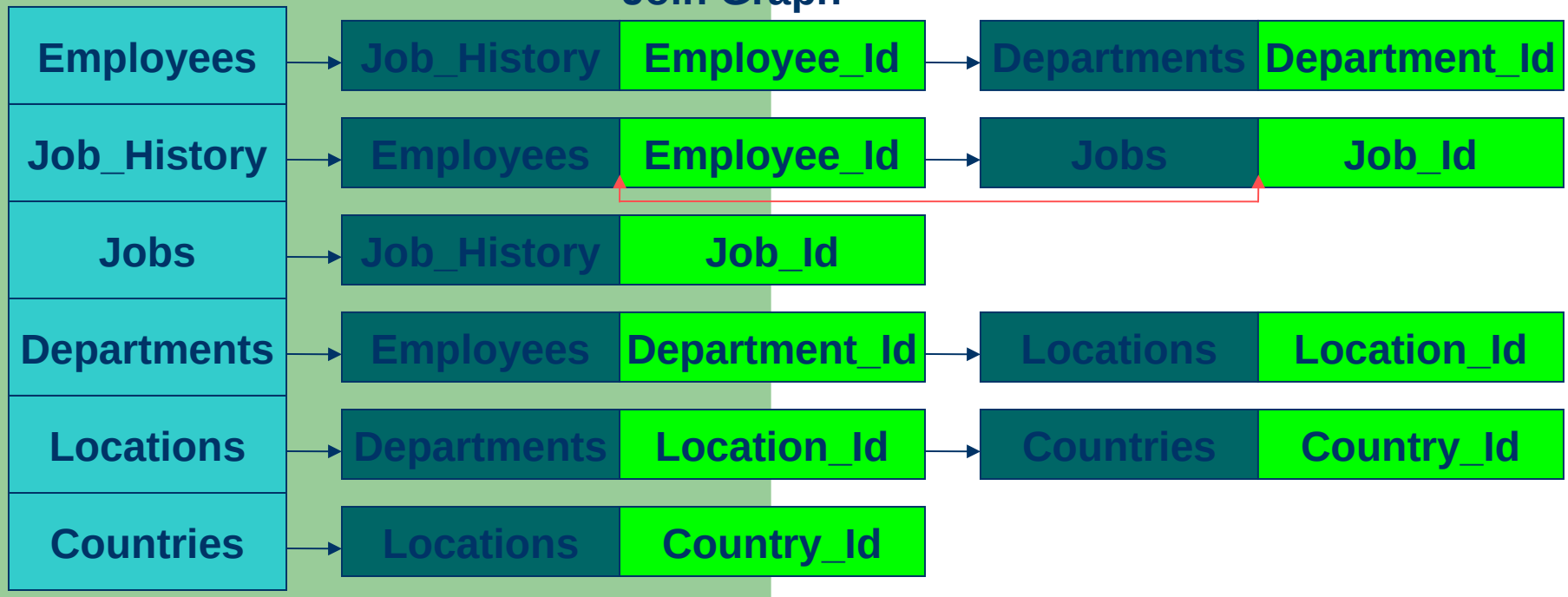
path

- Employees
- Job_History
- Departments

Join Base Tables

Employees | Job_History | Jobs | Departments | Locations | Countries

Join Graph



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables

create 2 dynamic arrays queue and path

insert T_0 into path

insert T_0 into queue

repeat

    $T_{Element}$ = First Table in queue

    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do

        if the Link Item is in the join sequence then

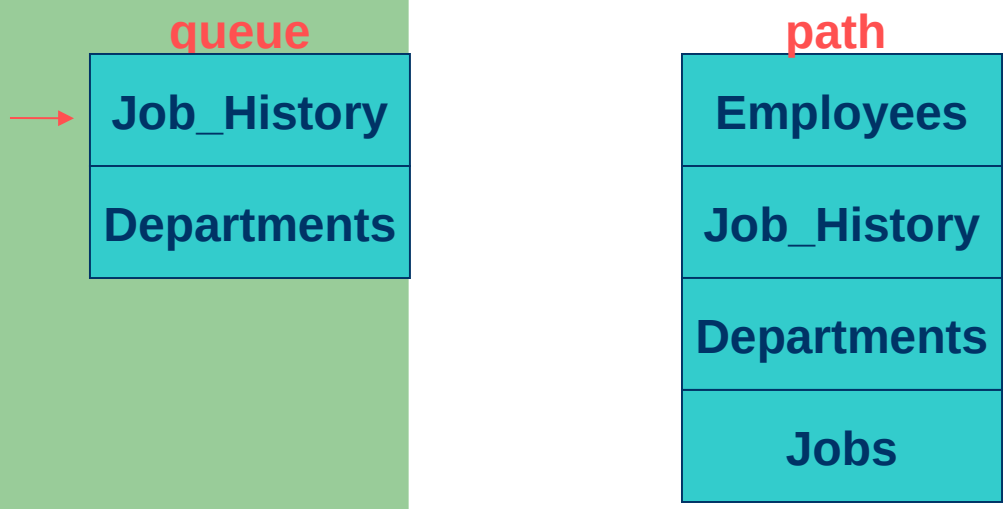
            if path doesn't contain the Link Item then

                insert Link Item into path

                insert Link Item into queue

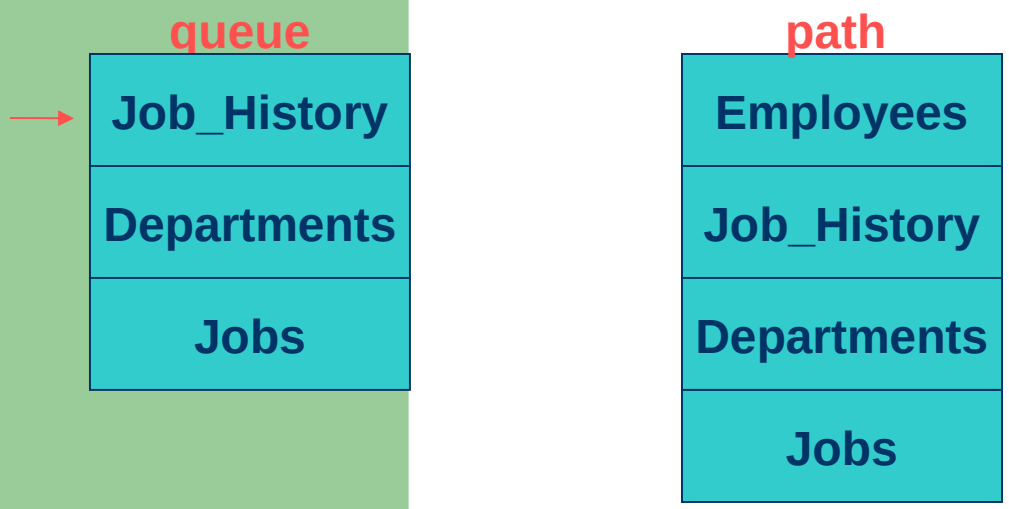
    remove $T_{Element}$ from queue

until queue is empty



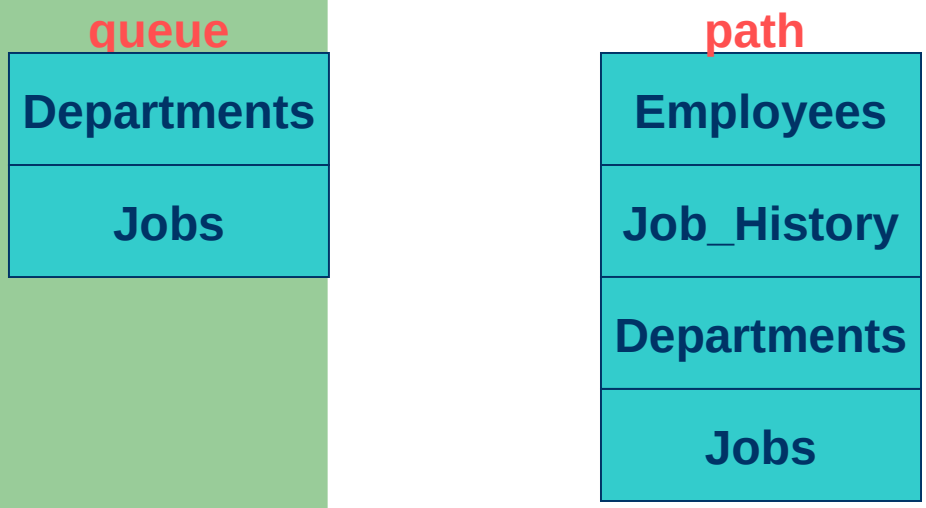
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables

create 2 dynamic arrays queue and path

insert T_0 into path

insert T_0 into queue

repeat

    $T_{Element}$ = First Table in queue

    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do

        if the Link Item is in the join sequence then

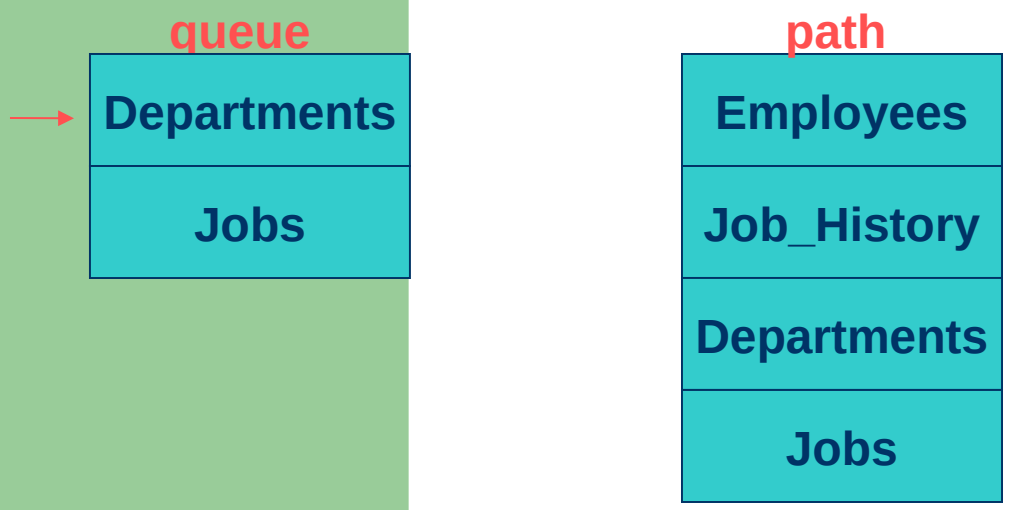
            if path doesn't contain the Link Item then

                insert Link Item into path

                insert Link Item into queue

    remove $T_{Element}$ from queue

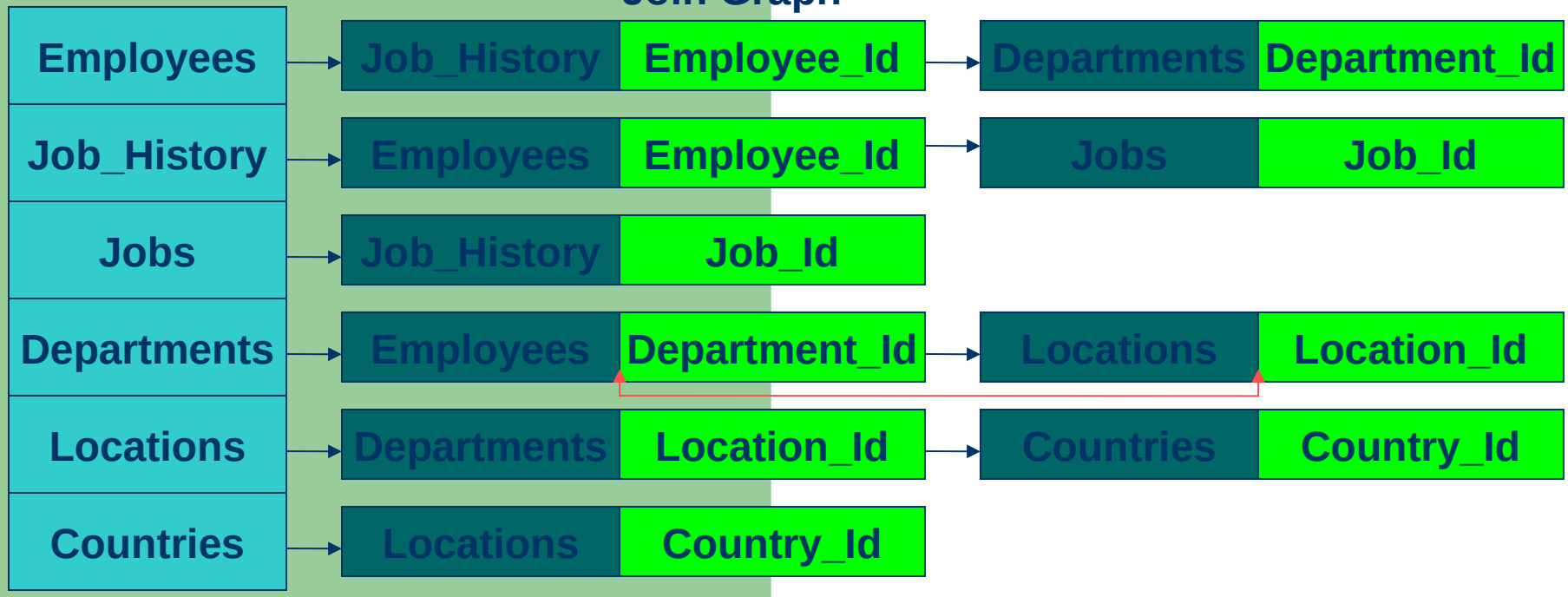
until queue is empty



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty

Join Graph

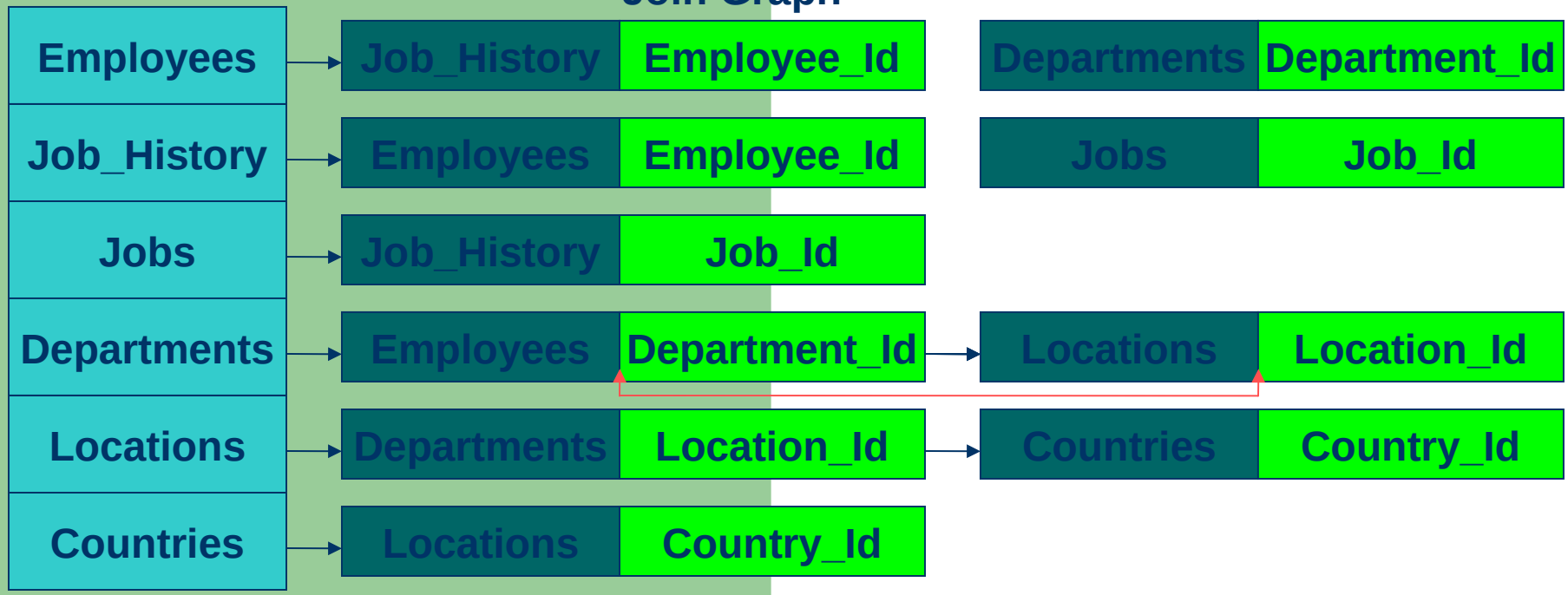


generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty

Join Base Tables

Join Graph



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty

path

Employees	Jobs
Job_History	
Departments	

Join Base Tables

Employees	Job_History	Jobs	Departments	Locations	Countries

Join Graph

Employees	Job_History	Employee_Id	Departments	Department_Id
Job_History	Employees	Employee_Id	Jobs	Job_Id
Jobs	Job_History	Job_Id		
Departments	Employees	Department_Id	Locations	Location_Id
Locations	Departments	Location_Id	Countries	Country_Id
Countries	Locations	Country_Id		

generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables

create 2 dynamic arrays queue and path

insert T_0 into path

insert T_0 into queue

repeat

    $T_{Element}$ = First Table in queue

    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do

        if the Link Item is in the join sequence then

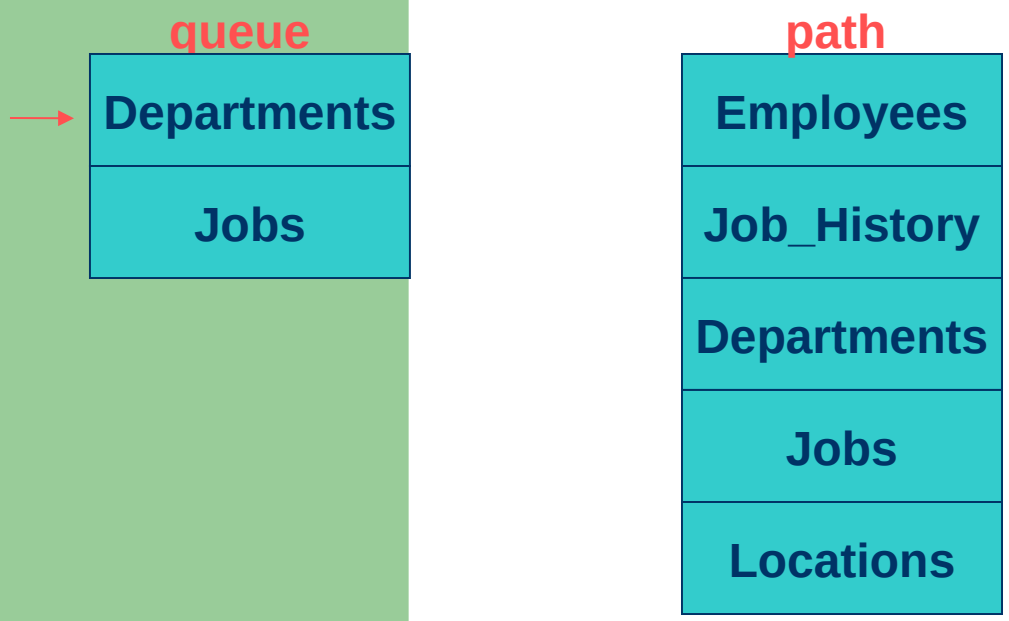
            if path doesn't contain the Link Item then

                insert Link Item into path

                insert Link Item into queue

    remove $T_{Element}$ from queue

until queue is empty



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables

create 2 dynamic arrays queue and path

insert T_0 into path

insert T_0 into queue

repeat

    $T_{Element}$ = First Table in queue

    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do

        if the Link Item is in the join sequence then

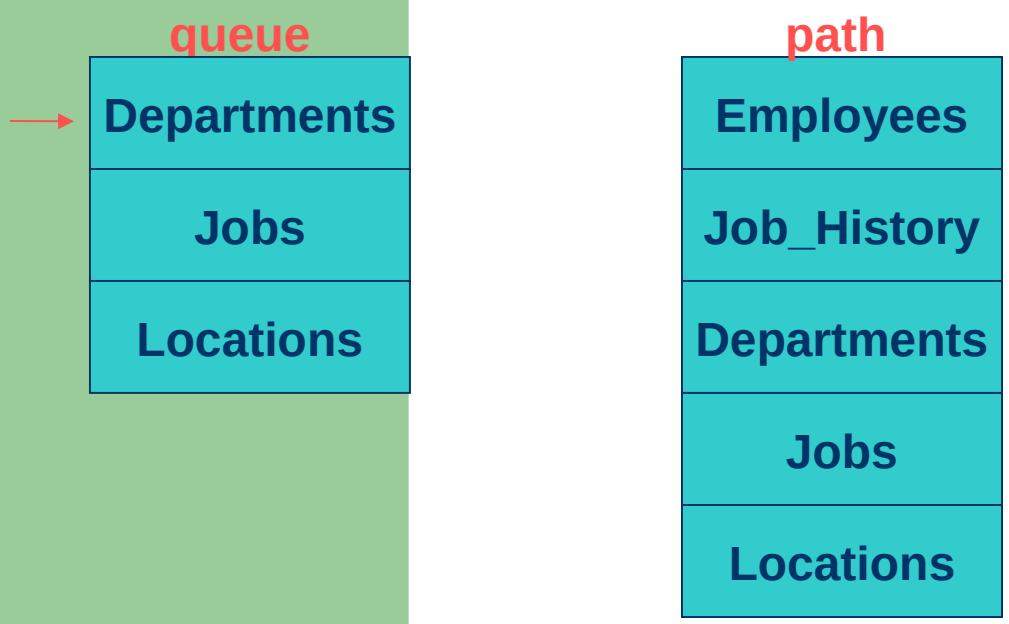
            if path doesn't contain the Link Item then

                insert Link Item into path

                insert Link Item into queue

    remove $T_{Element}$ from queue

until queue is empty



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables

create 2 dynamic arrays queue and path

insert T_0 into path

insert T_0 into queue

repeat

    $T_{Element}$ = First Table in queue

    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do

        if the Link Item is in the join sequence then

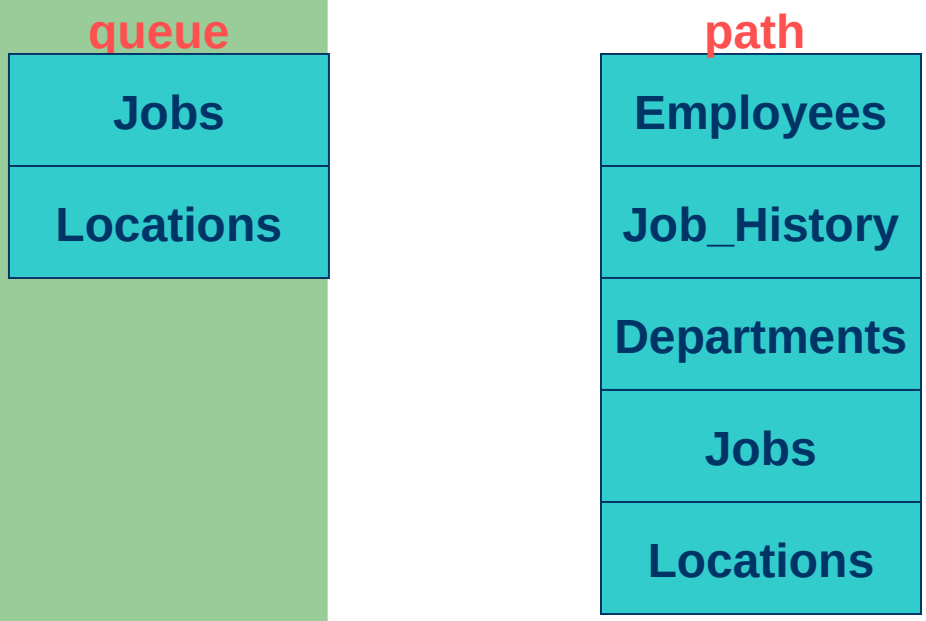
            if path doesn't contain the Link Item then

                insert Link Item into path

                insert Link Item into queue

    remove $T_{Element}$ from queue

until queue is empty



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables

create 2 dynamic arrays queue and path

insert T_0 into path

insert T_0 into queue

repeat

    $T_{Element}$ = First Table in queue

    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do

        if the Link Item is in the join sequence then

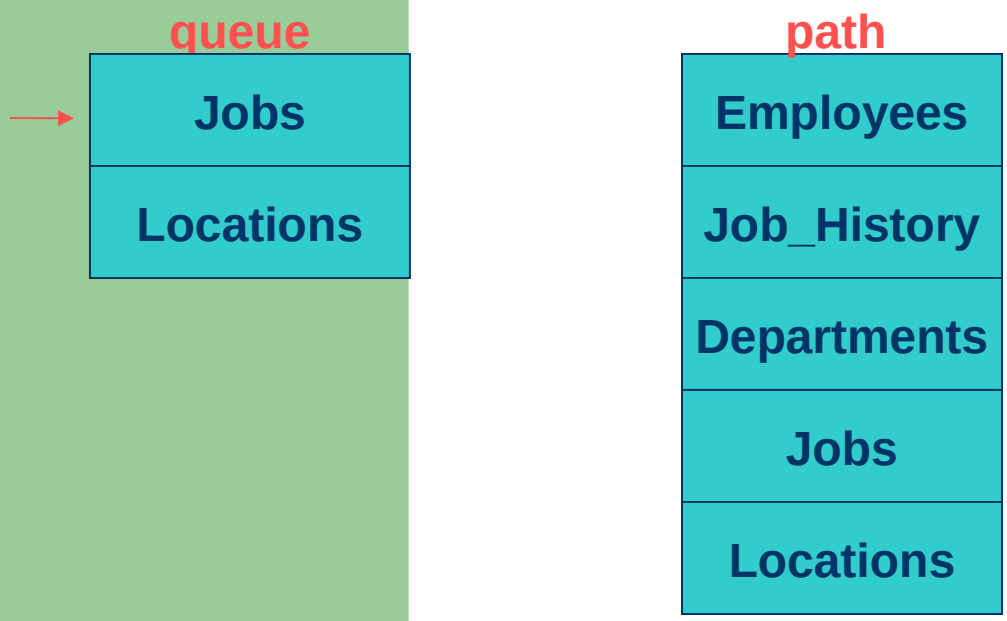
            if path doesn't contain the Link Item then

                insert Link Item into path

                insert Link Item into queue

    remove $T_{Element}$ from queue

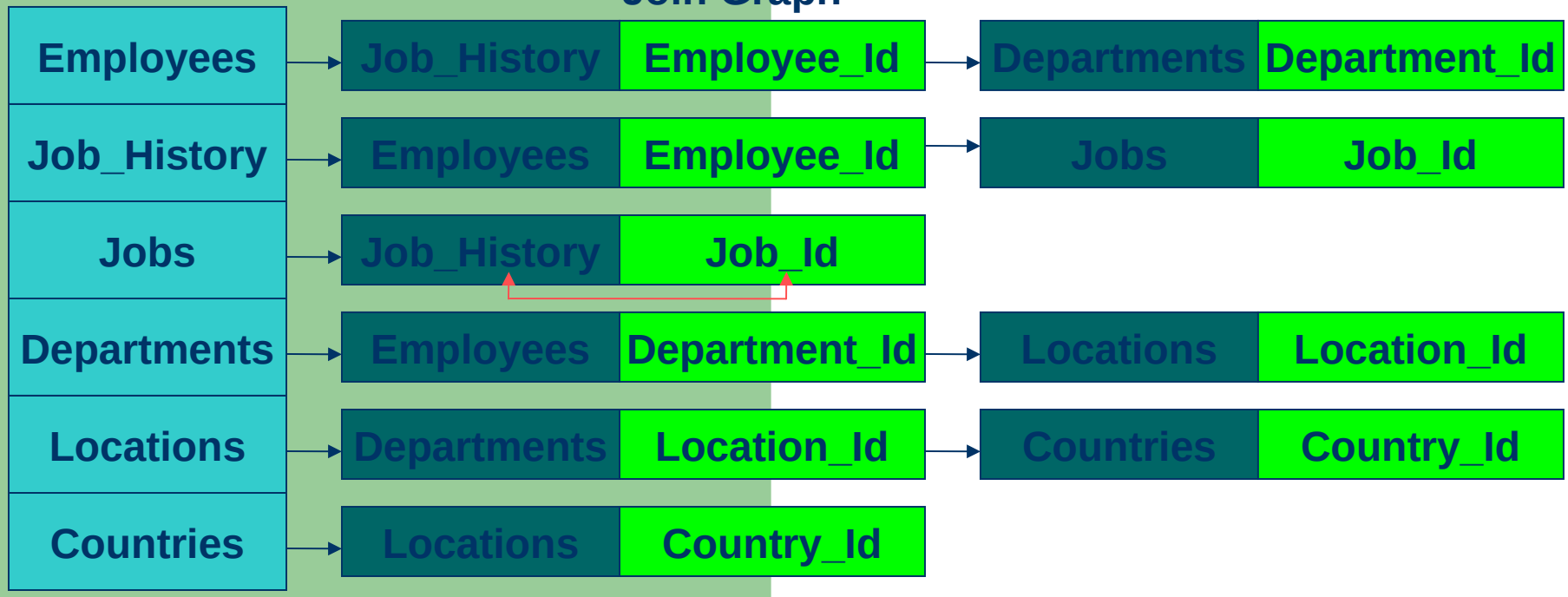
until queue is empty



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty

Join Graph

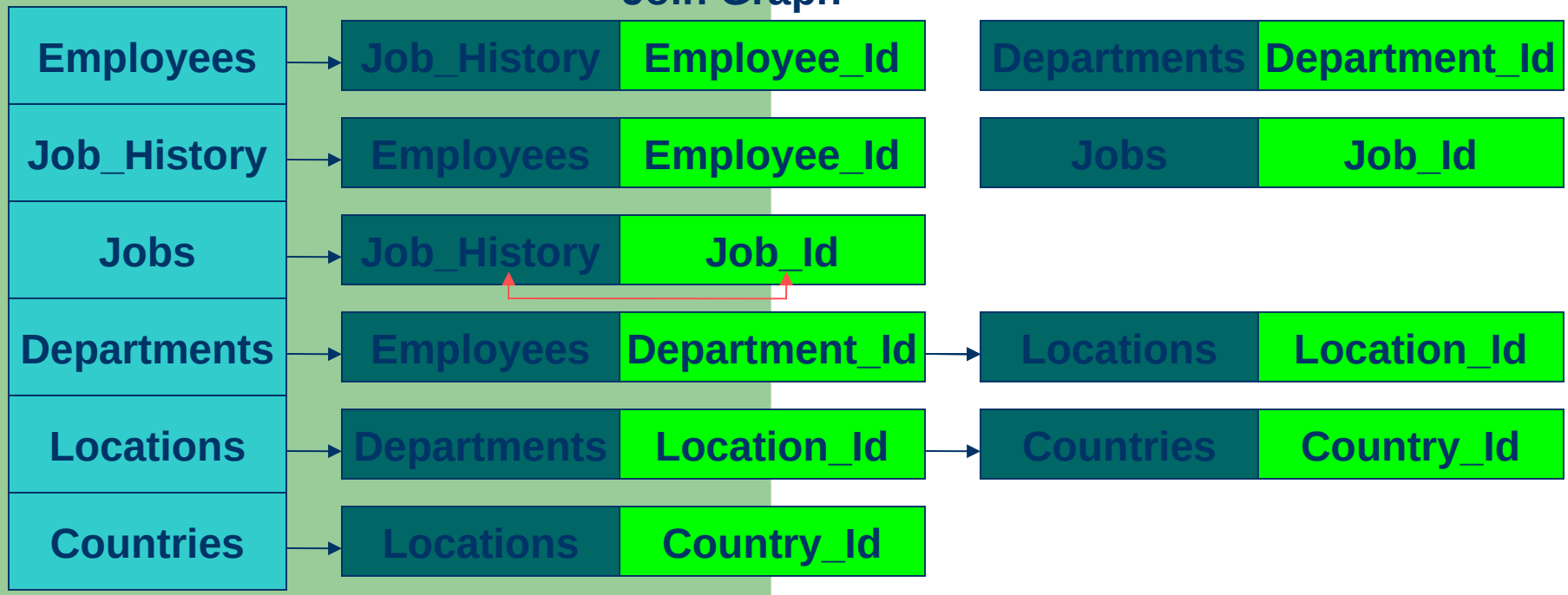


generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty

Join Base Tables

Join Graph



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables

create 2 dynamic arrays queue and path

insert T_0 into path

insert T_0 into queue

repeat

$T_{Element}$ = First Table in queue

for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do

if the Link Item is in the join sequence then

if path doesn't contain the Link Item then

insert Link Item into path

insert Link Item into queue

remove $T_{Element}$ from queue

until queue is empty

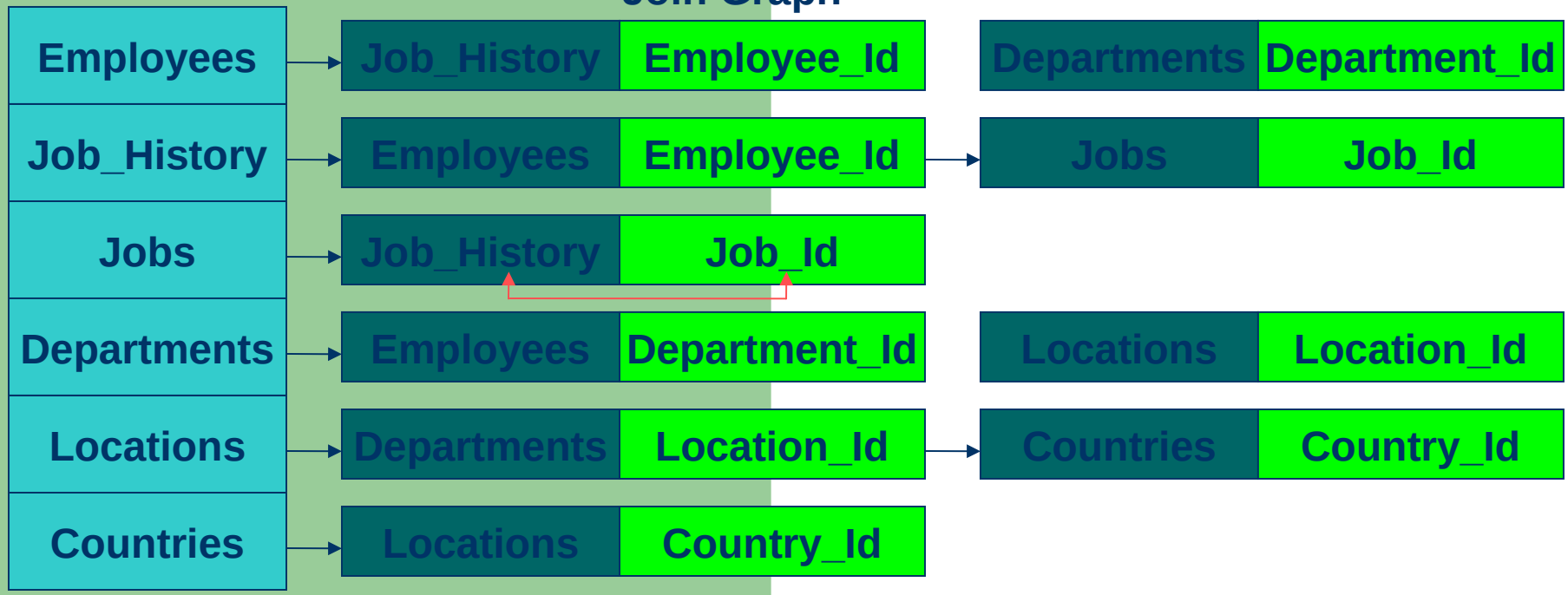
path

Employees	Jobs
Job_History	Locations
Departments	

Join Base Tables

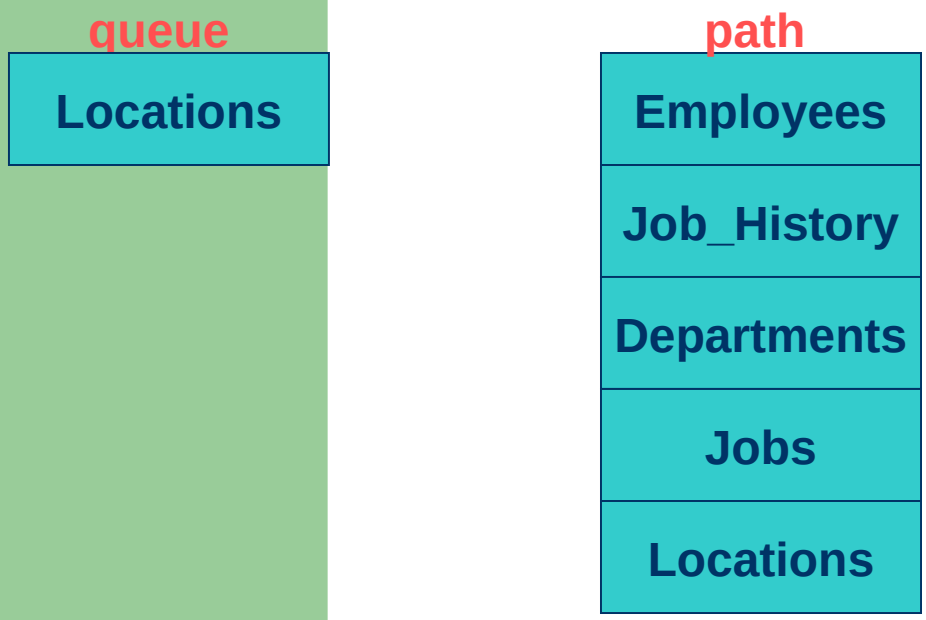
Employees	Job_History	Jobs	Departments	Locations	Countries

Join Graph



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables

create 2 dynamic arrays queue and path

insert T_0 into path

insert T_0 into queue

repeat

$T_{Element}$ = First Table in queue

for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do

if the Link Item is in the join sequence then

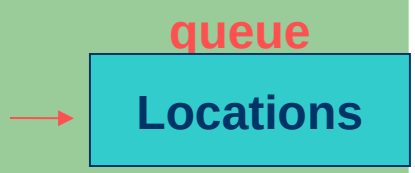
if path doesn't contain the Link Item then

insert Link Item into path

insert Link Item into queue

remove $T_{Element}$ from queue

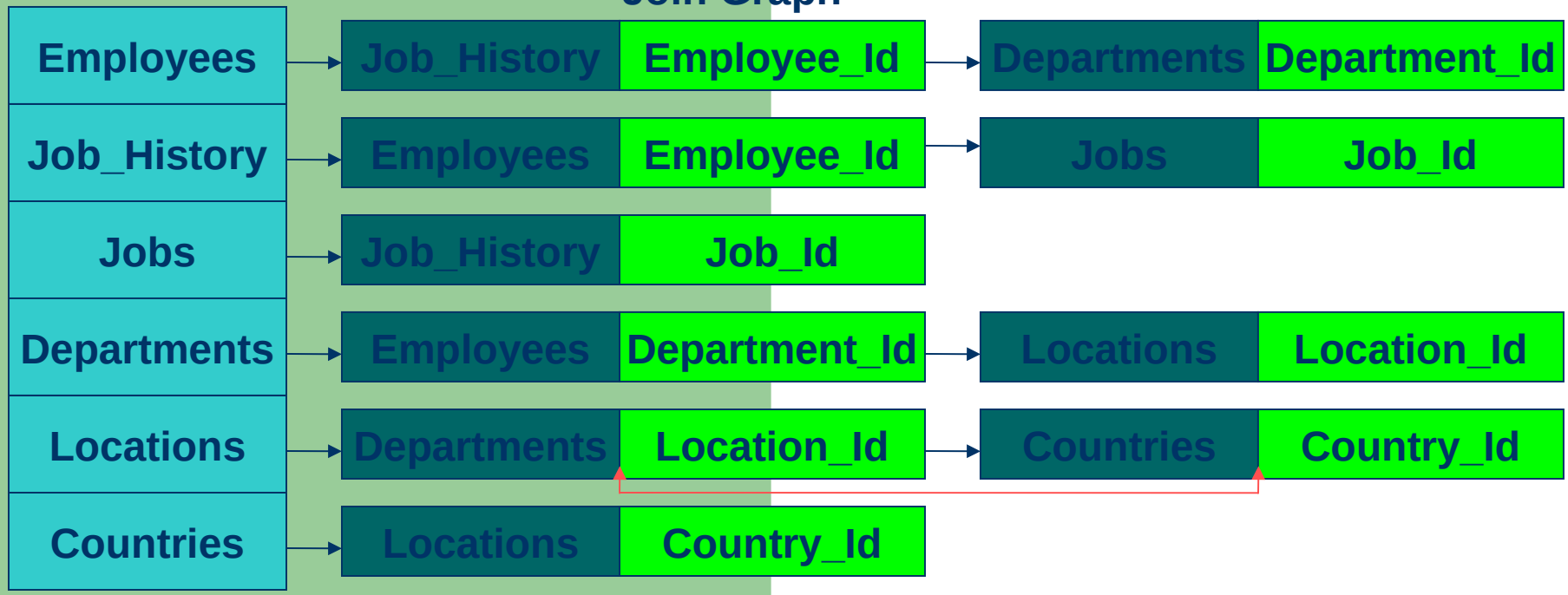
until queue is empty



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty

Join Graph

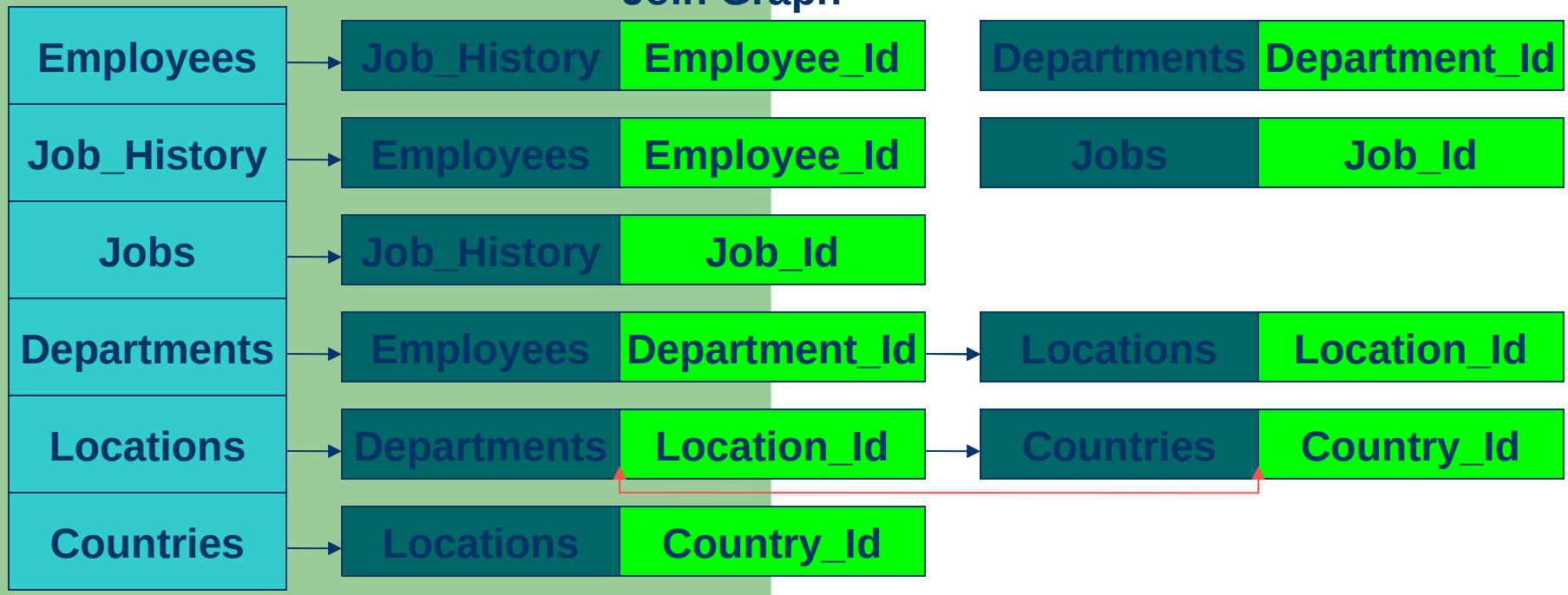


generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty

Join Base Tables

Join Graph



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty

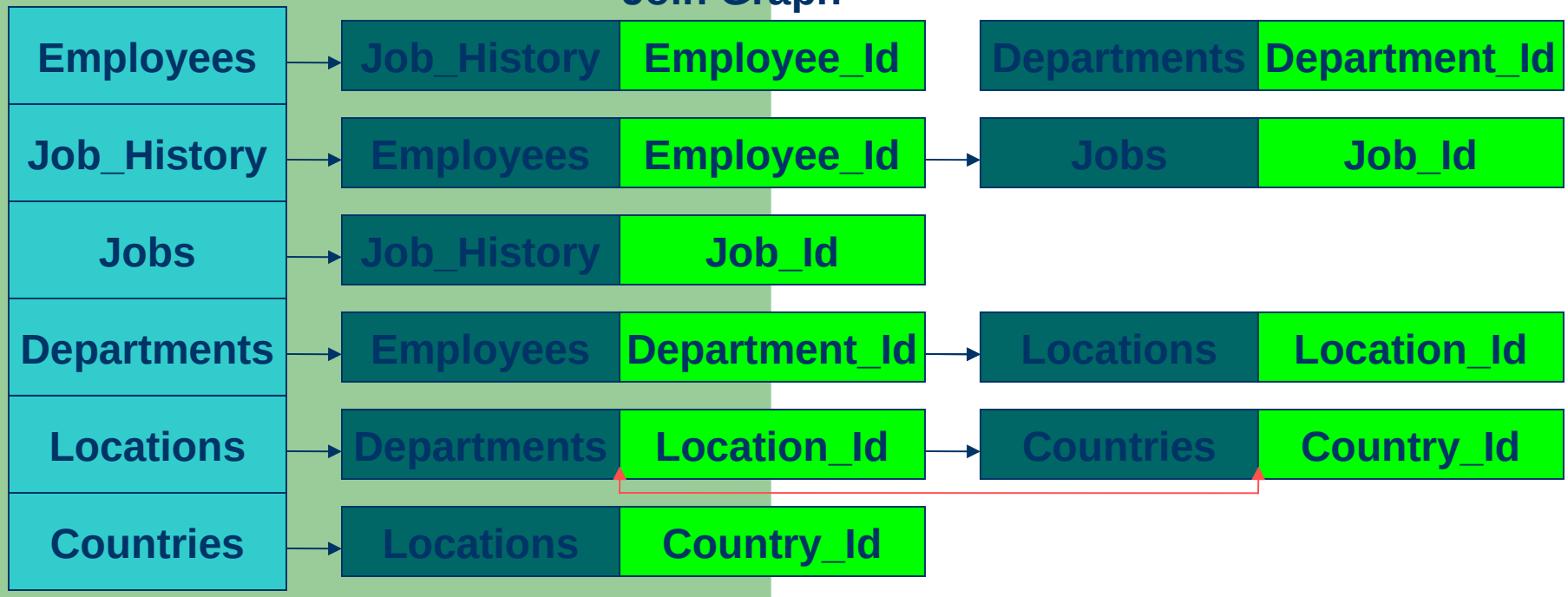
path

Employees	Jobs
Job_History	Locations
Departments	

Join Base Tables

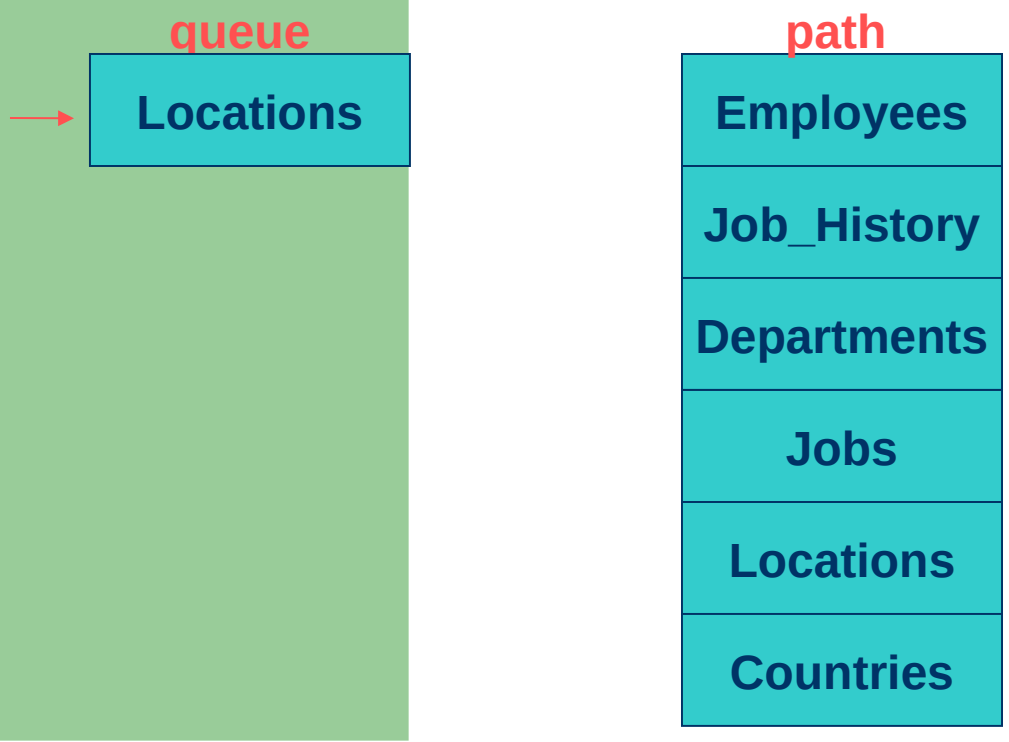
Employees	Job_History	Jobs	Departments	Locations	Countries

Join Graph



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables

create 2 dynamic arrays queue and path

insert T_0 into path

insert T_0 into queue

repeat

    $T_{Element}$ = First Table in queue

    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do

        if the Link Item is in the join sequence then

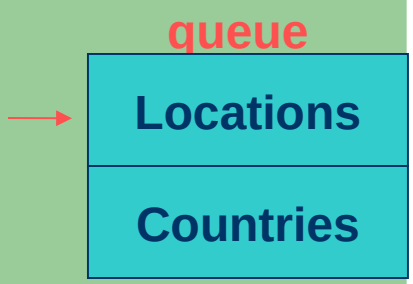
            if path doesn't contain the Link Item then

                insert Link Item into path

                insert Link Item into queue

    remove $T_{Element}$ from queue

until queue is empty



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables

create 2 dynamic arrays queue and path

insert T_0 into path

insert T_0 into queue

repeat

    $T_{Element}$ = First Table in queue

    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do

        if the Link Item is in the join sequence then

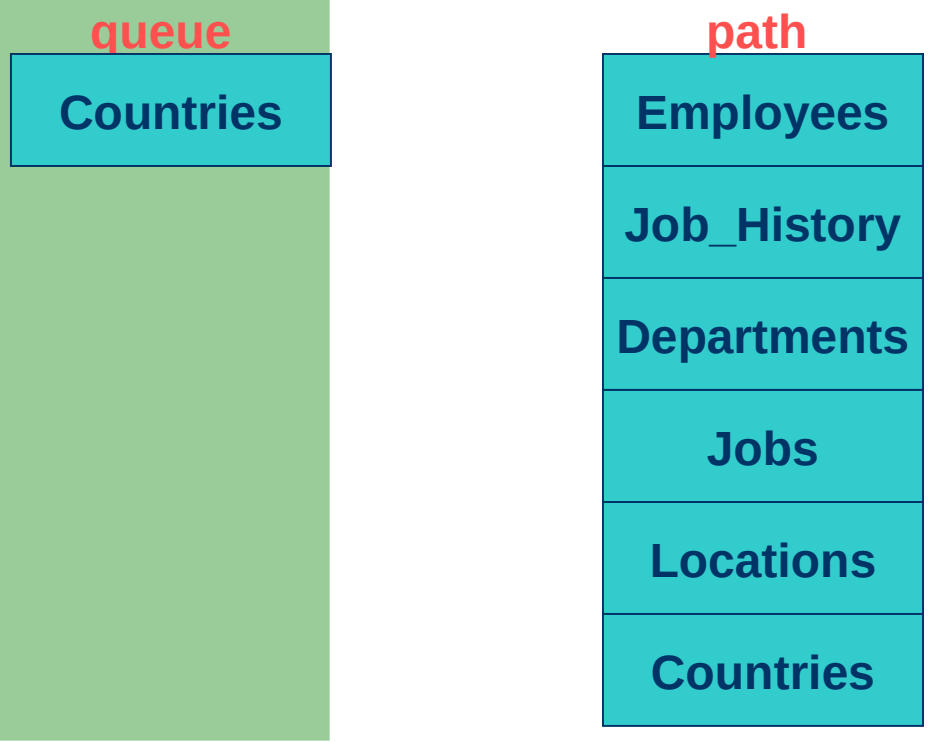
            if path doesn't contain the Link Item then

                insert Link Item into path

                insert Link Item into queue

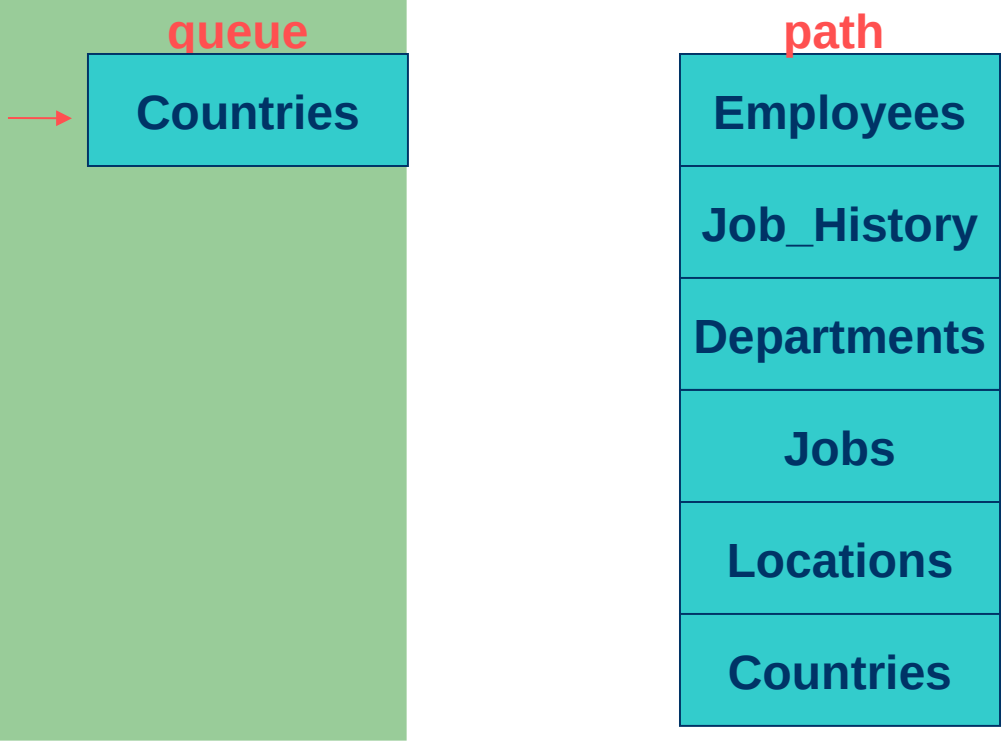
    remove $T_{Element}$ from queue

until queue is empty



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

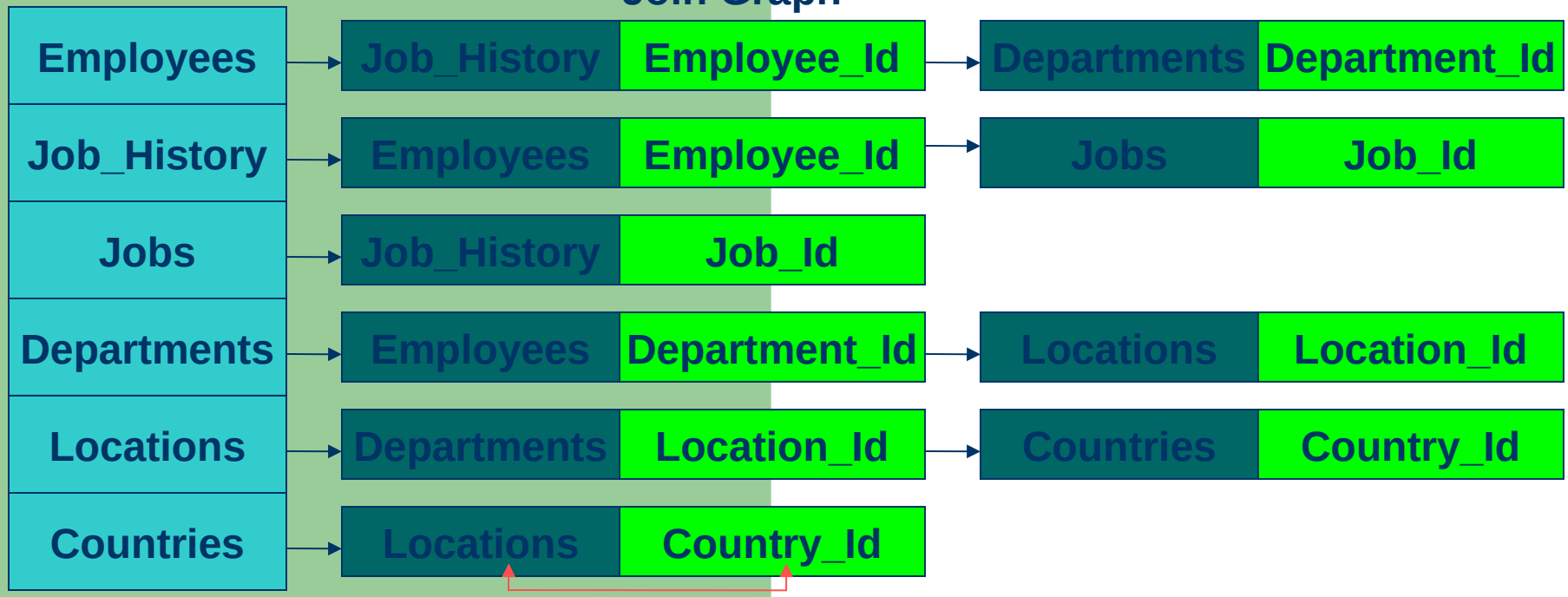
let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty

Join Graph

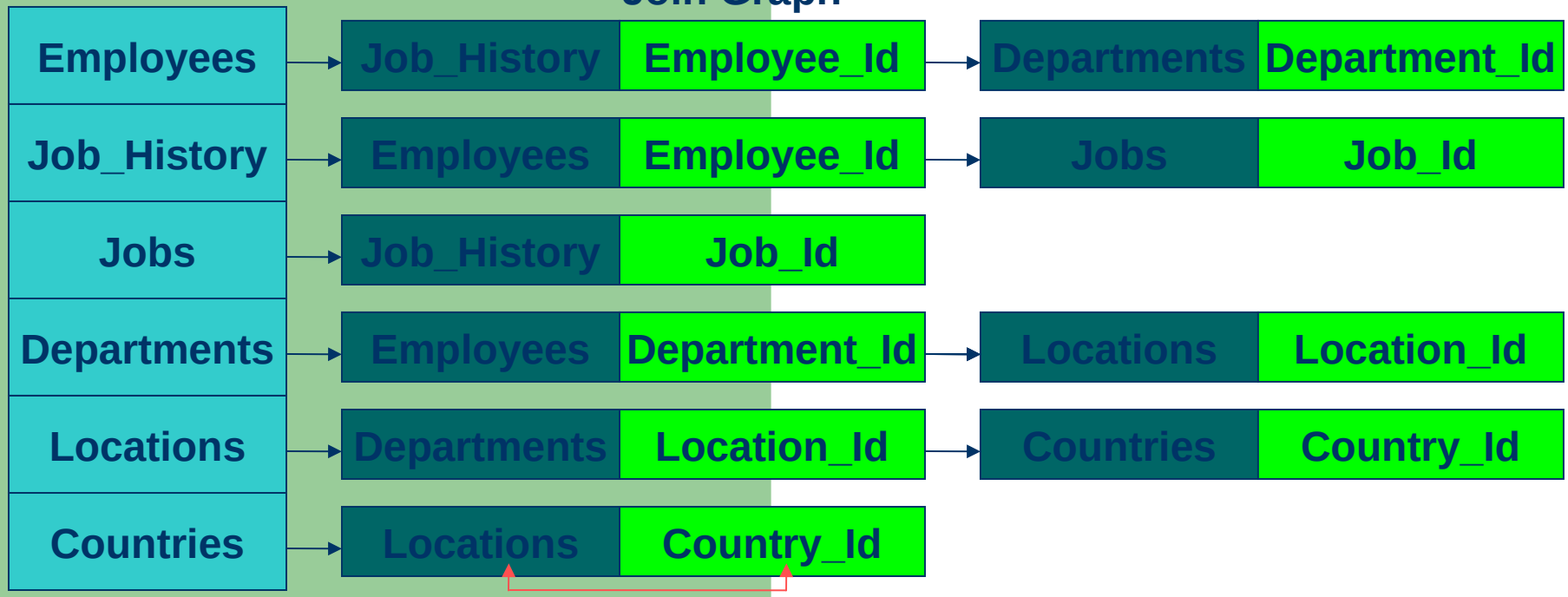


generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert T_0 into path
insert T_0 into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue
    remove $T_{Element}$ from queue
until queue is empty

Join Base Tables

Join Graph



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables

create 2 dynamic arrays queue and path

insert T_0 into path

insert T_0 into queue

repeat

$T_{Element}$ = First Table in queue

for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do

if the Link Item is in the join sequence then

if path doesn't contain the Link Item then

insert Link Item into path

insert Link Item into queue

remove $T_{Element}$ from queue

until queue is empty

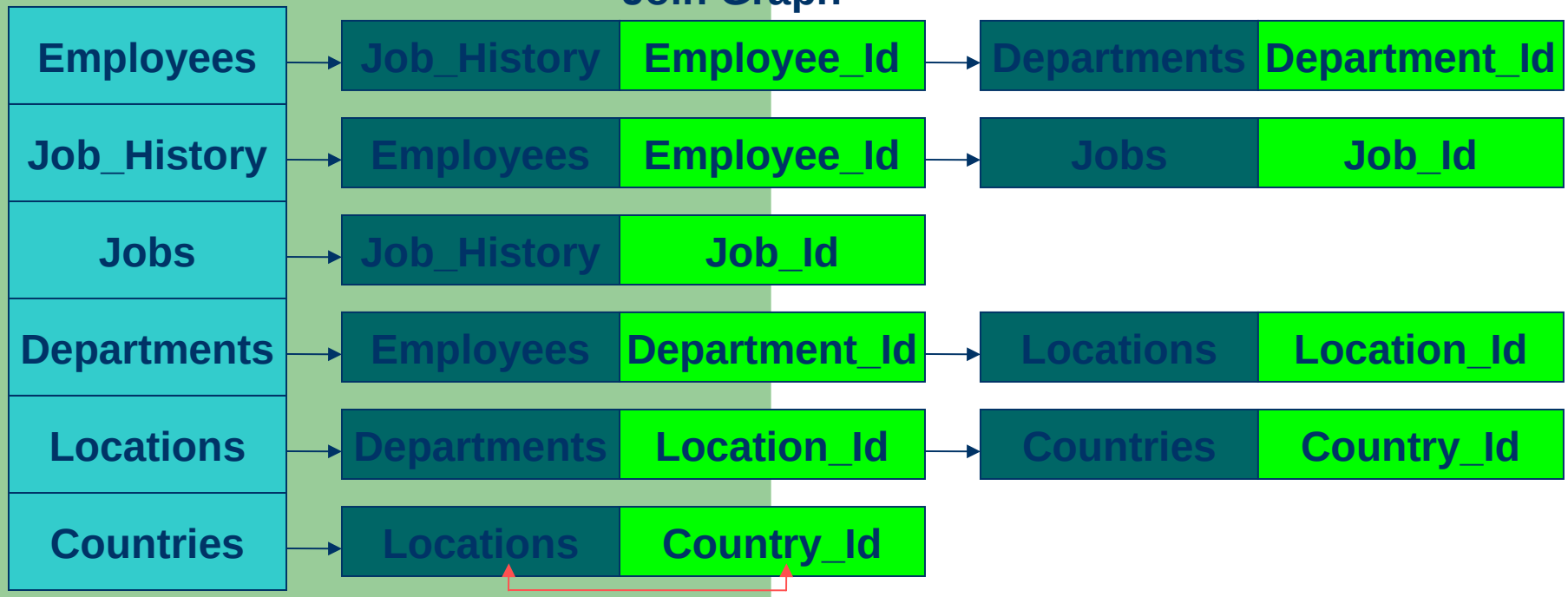
path

Employees	Jobs
Job_History	Locations
Departments	Countries

Join Base Tables

Employees	Job_History	Jobs	Departments	Locations	Countries

Join Graph



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables

create 2 dynamic arrays queue and path

insert T_0 into path

insert T_0 into queue

repeat

$T_{Element}$ = First Table in queue

for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do

if the Link Item is in the join sequence then

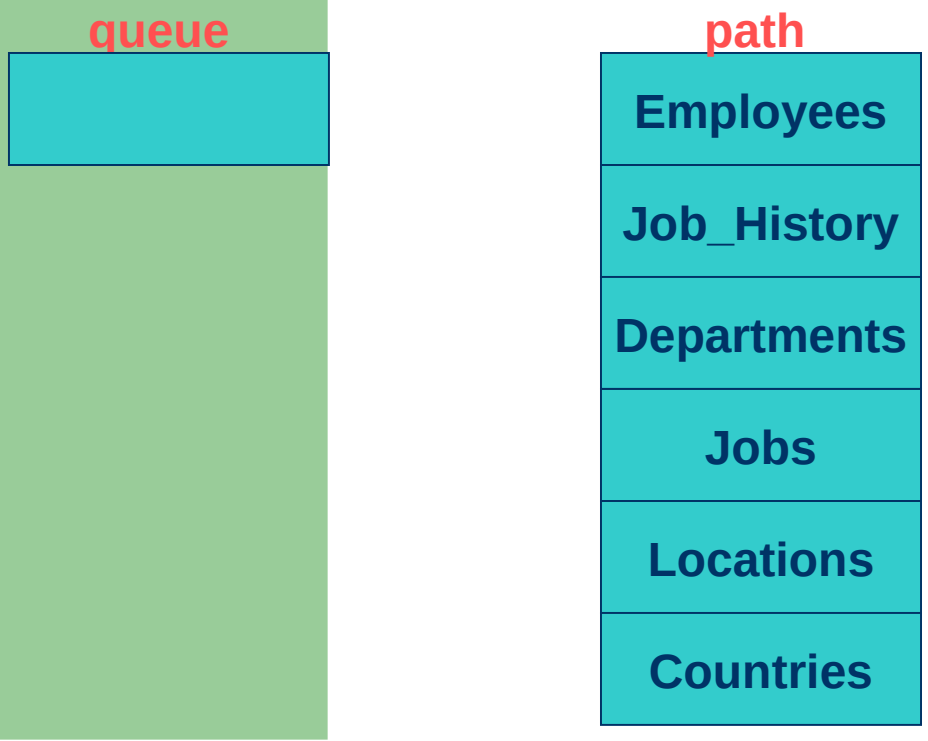
if path doesn't contain the Link Item then

insert Link Item into path

insert Link Item into queue

remove $T_{Element}$ from queue

until queue is empty



generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

let $T_0 \ldots T_m$ be the base tables

create 2 dynamic arrays queue and path

insert T_0 into path

insert T_0 into queue

repeat

    $T_{Element}$ = First Table in queue

    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do

        if the Link Item is in the join sequence then

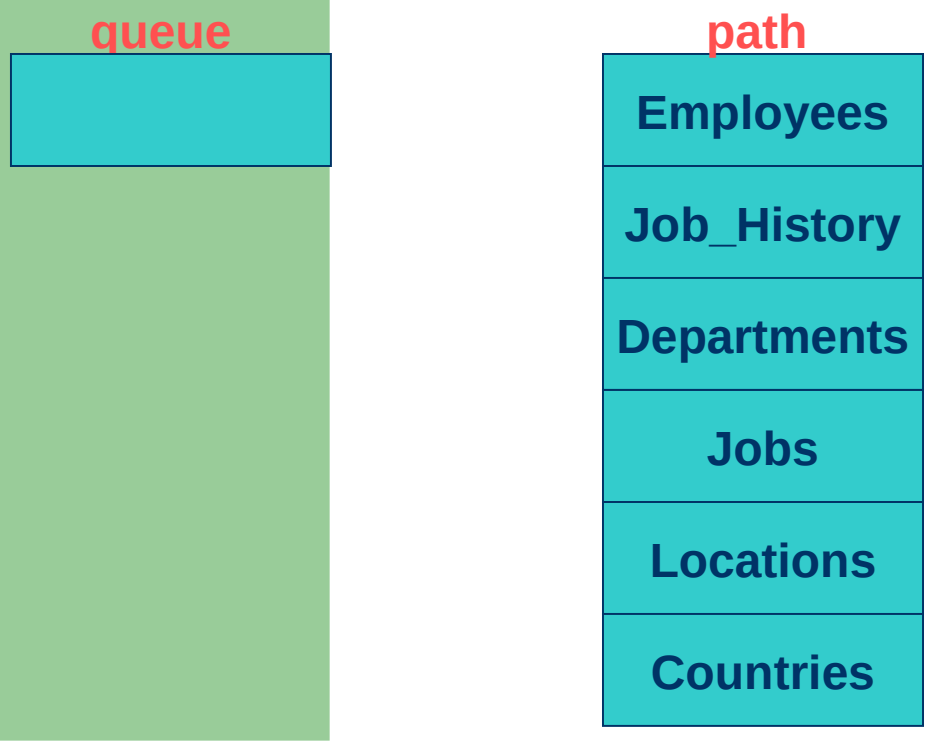
            if path doesn't contain the Link Item then

                insert Link Item into path

                insert Link Item into queue

    remove $T_{Element}$ from queue

until queue is empty



→ insert all the names of base tables from path as vertexes in JoinPathList

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one T_i at a time

JoinPathAdjacentList(T_i) = $T_{[buf]}$

Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)

JoinPathAdjacentList($T_{[buf]}$) = T_i

Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)

$T_{[buf]}$ + = T_i

Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$

vertexes

Employees
Job_History
Deparments
Jobs
Locations
Countries

path

Employees
Job_History
Departments
Jobs
Locations
Countries

insert all the names of base tables from path as vertexes in JoinPathList

→ create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one T_i at a time

JoinPathAdjacentList(T_i) = $T_{[buf]}$

Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)

JoinPathAdjacentList($T_{[buf]}$) = T_i

Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)

$T_{[buf]}$ + = T_i

Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$

vertexes

Employees
Job_History
Deparments
Jobs
Locations
Countries

buf

path

Employees
Job_History
Departments
Jobs
Locations
Countries

insert all the names of base tables from path as vertexes in JoinPathList

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one T_i at a time

JoinPathAdjacentList(T_i) = $T_{[buf]}$

Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)

JoinPathAdjacentList($T_{[buf]}$) = T_i

Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)

$T_{[buf]}$ + = T_i

Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$

vertexes

- Employees
- Job_History
- Deparments
- Jobs
- Locations
- Countries

buf

- Employees

path

- Employees
- Job_History
- Departments
- Jobs
- Locations
- Countries

insert all the names of base tables from path as vertexes in JoinPathList

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one T_i at a time

JoinPathAdjacentList(T_i) = $T_{[buf]}$

Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)

JoinPathAdjacentList($T_{[buf]}$) = T_i

Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)

$T_{[buf]}$ + = T_i

Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$

vertexes

- Employees
- Job_History
- Deparments
- Jobs
- Locations
- Countries

buf

- Employees

path

- Employees
- Job_History
- Departments
- Jobs
- Locations
- Countries

insert all the names of base tables from path as vertexes in JoinPathList

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

→ take one T_i at a time

JoinPathAdjacentList(T_i) = $T_{[buf]}$

Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)

JoinPathAdjacentList($T_{[buf]}$) = T_i

Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)

$T_{[buf]}$ + = T_i

Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$

vertexes

- Employees
- Job_History
- Deparments
- Jobs
- Locations
- Countries

buf

- Employees

path

- Employees
- Job_History
- Departments
- Jobs
- Locations
- Countries

insert all the names of base tables from path as vertexes in JoinPathList

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

→ take one T_i at a time

$JoinPathAdjacentList(T_i) = T_{[buf]}$

$Key(T_i) = getFirstAdjacentListKey(T_i, T_{[buf]})$

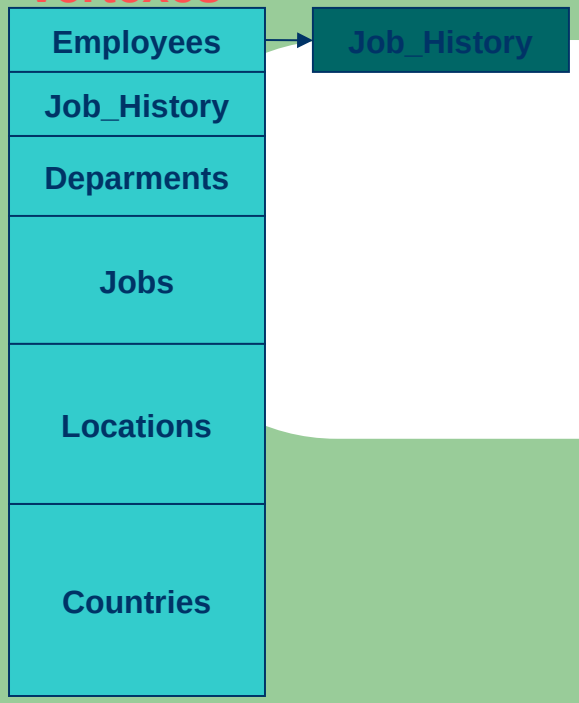
$JoinPathAdjacentList(T_{[buf]}) = T_i$

$Key(T_{[buf]}) = getFirstAdjacentListKey\ (T_{[buf]}, T_i)$

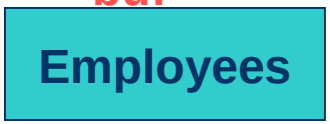
$T_{[buf]} + = T_i$

$Insert\ NodesList[T_{[buf]}] = T_{[buf]}$

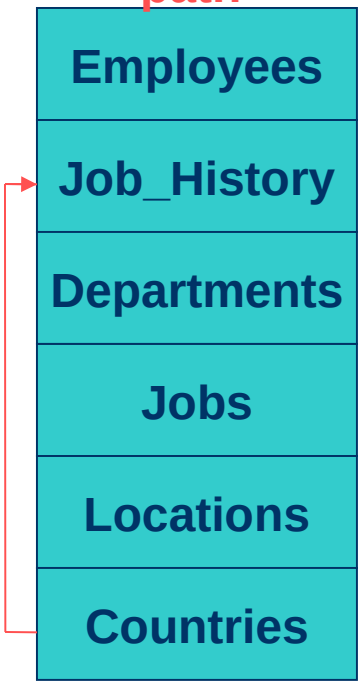
vertexes



buf



path



insert all the names of base tables from path as vertexes in JoinPathList

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one T_i at a time

$JoinPathAdjacentList(T_i) = T_{[buf]}$

$Key(T_i) = getFirstAdjacentListKey(T_i, T_{[buf]})$

$JoinPathAdjacentList(T_{[buf]}) = T_i$

$Key(T_{[buf]}) = getFirstAdjacentListKey\ (T_{[buf]}, T_i)$

$T_{[buf]} + = T_i$

$Insert\ NodesList[T_{[buf]}] = T_{[buf]}$

vertexes

- Employees
- Job_History
- Deparments
- Jobs
- Locations
- Countries

Job_History	Employees	Employee_Id

buf

Employees

path

- Employees
- Job_History
- Departments
- Jobs
- Locations
- Countries

insert all the names of base tables from path as vertexes in JoinPathList

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one T_i at a time

$JoinPathAdjacentList(T_i) = T_{[buf]}$

$Key(T_i) = getFirstAdjacentListKey(T_i, T_{[buf]})$

$JoinPathAdjacentList(T_{[buf]}) = T_i$

$Key(T_{[buf]}) = getFirstAdjacentListKey(T_{[buf]}, T_i)$

$T_{[buf]} += T_i$

$Insert\ NodesList[T_{[buf]}] = T_{[buf]}$

vertexes

- Employees
- Job_History
- Deparments
- Jobs
- Locations
- Countries

Job_History	Employees	Employee_Id
Employees		

buf

Employees

path

- Employees
- Job_History
- Departments
- Jobs
- Locations
- Countries

insert all the names of base tables from path as vertexes in JoinPathList

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one T_i at a time

$JoinPathAdjacentList(T_i) = T_{[buf]}$

$Key(T_i) = getFirstAdjacentListKey(T_i, T_{[buf]})$

$JoinPathAdjacentList(T_{[buf]}) = T_i$

$Key(T_{[buf]}) = getFirstAdjacentListKey(T_{[buf]}, T_i)$

$T_{[buf]} += T_i$

$Insert\ NodesList[T_{[buf]}] = T_{[buf]}$

vertexes

Employees	Job_History	Employees	Employee_Id
Job_History	Employees	Job_History	Employee_Id
Deparments			
Jobs			
Locations			
Countries			

buf

Employees

path

Employees
Job_History
Departments
Jobs
Locations
Countries

insert all the names of base tables from path as vertexes in JoinPathList

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one T_i at a time

$JoinPathAdjacentList(T_i) = T_{[buf]}$

$Key(T_i) = getFirstAdjacentListKey(T_i, T_{[buf]})$

$JoinPathAdjacentList(T_{[buf]}) = T_i$

→ $Key(T_{[buf]}) = getFirstAdjacentListKey(T_{[buf]}, T_i)$

$T_{[buf]} += T_i$

$Insert\ NodesList[T_{[buf]}] = T_{[buf]}$

vertexes

Employees
Job_History
Deparments
Jobs
Locations
Countries

Job_History	Employees	Employee_Id
Employees	Job_History	Employee_Id

buf

Employees
Job_History

path

Employees
Job_History
Departments
Jobs
Locations
Countries

insert all the names of base tables from path as vertexes in JoinPathList

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one T_i at a time

$JoinPathAdjacentList(T_i) = T_{[buf]}$

$Key(T_i) = getFirstAdjacentListKey(T_i, T_{[buf]})$

$JoinPathAdjacentList(T_{[buf]}) = T_i$

$Key(T_{[buf]}) = getFirstAdjacentListKey\ (T_{[buf]}, T_i)$

→ $T_{[buf]} += T_i$

$Insert\ NodesList[T_{[buf]}] = T_{[buf]}$

vertexes

- Employees
- Job_History
- Deparments
- Jobs
- Locations
- Countries
- Employees Job_History

Job_History	Employees	Employee_Id
Employees	Job_History	Employee_Id

buf

Employees
Job_History

path

- Employees
- Job_History
- Departments
- Jobs
- Locations
- Countries

insert all the names of base tables from path as vertexes in JoinPathList

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one T_i at a time

$JoinPathAdjacentList(T_i) = T_{[buf]}$

$Key(T_i) = getFirstAdjacentListKey(T_i, T_{[buf]})$

$JoinPathAdjacentList(T_{[buf]}) = T_i$

$Key(T_{[buf]}) = getFirstAdjacentListKey(T_{[buf]}, T_i)$

$T_{[buf]} += T_i$

$Insert\ NodesList[T_{[buf]}] = T_{[buf]}$

vertexes

Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History

Job_History	Employees	Employee_Id
Employees	Job_History	Employee_Id

buf

Employees Job_History

path

Employees
Job_History
Departments
Jobs
Locations
Countries

insert all the names of base tables from path as vertexes in JoinPathList

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one T_i at a time

$JoinPathAdjacentList(T_i) = T_{[buf]}$

$Key(T_i) = getFirstAdjacentListKey(T_i, T_{[buf]})$

$JoinPathAdjacentList(T_{[buf]}) = T_i$

$Key(T_{[buf]}) = getFirstAdjacentListKey(T_{[buf]}, T_i)$

$T_{[buf]} += T_i$

$Insert\ NodesList[T_{[buf]}] = T_{[buf]}$

vertexes

Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History

Job_History	Employees	Employee_Id
Employees	Job_History	Employee_Id
Employees Job_History		

buf

Employees Job_History

path

Employees
Job_History
Departments
Jobs
Locations
Countries

insert all the names of base tables from path as vertexes in JoinPathList

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one T_i at a time

$JoinPathAdjacentList(T_i) = T_{[buf]}$

$Key(T_i) = getFirstAdjacentListKey(T_i, T_{[buf]})$

$JoinPathAdjacentList(T_{[buf]}) = T_i$

$Key(T_{[buf]}) = getFirstAdjacentListKey(T_{[buf]}, T_i)$

$T_{[buf]} += T_i$

$Insert\ NodesList[T_{[buf]}] = T_{[buf]}$

vertexes

Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History

Job_History	Employees	Employee_Id
Employees	Job_History	Employee_Id
Employees Job_History	Departments	Department_Id

buf

Employees Job_History

path

Employees
Job_History
Departments
Jobs
Locations
Countries

insert all the names of base tables from path as vertexes in JoinPathList

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one T_i at a time

JoinPathAdjacentList(T_i) = $T_{[buf]}$

Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)

JoinPathAdjacentList($T_{[buf]}$) = T_i

Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)

$T_{[buf]}$ + = T_i

Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$

vertexes

Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History

Job_History	Employees	Employee_Id
Employees	Job_History	Employee_Id
Employees Job_History	Departments	Department_Id

Departments

buf

Employees Job_History

path

Employees
Job_History
Departments
Jobs
Locations
Countries

insert all the names of base tables from path as vertexes in JoinPathList

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one T_i at a time

JoinPathAdjacentList(T_i) = $T_{[buf]}$

Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)

JoinPathAdjacentList($T_{[buf]}$) = T_i

Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)

$T_{[buf]}$ + = T_i

Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$

vertexes

Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History

Job_History	Employees	Employee_Id
Employees	Job_History	Employee_Id
Employees Job_History	Departments	Department_Id
Departments	Employees	Department_Id

buf

Employees Job_History

path

Employees
Job_History
Departments
Jobs
Locations
Countries

insert all the names of base tables from path as vertexes in JoinPathList

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one T_i at a time

JoinPathAdjacentList(T_i) = $T_{[buf]}$

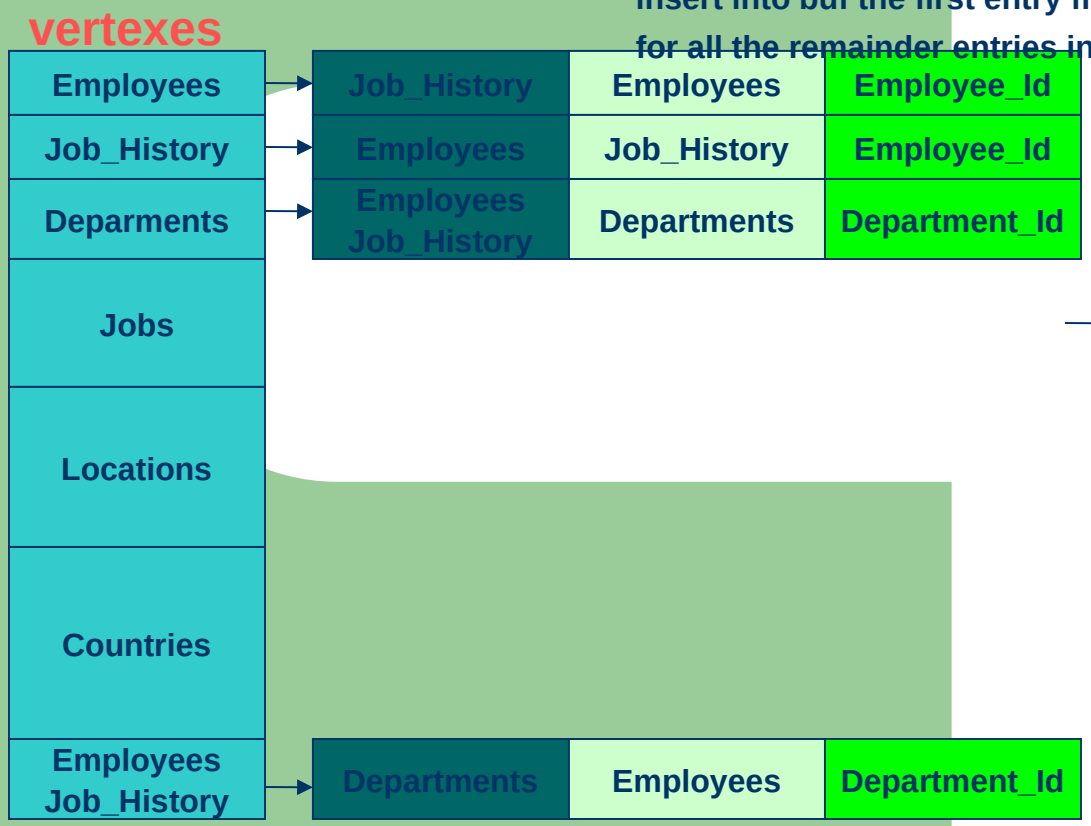
Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)

JoinPathAdjacentList($T_{[buf]}$) = T_i

Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)

$T_{[buf]}$ + = T_i

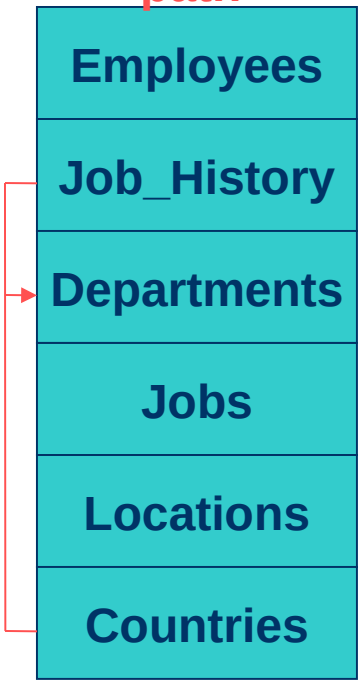
Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$



buf

Employees
Job_History
Departments

path



insert all the names of base tables from path as vertexes in JoinPathList

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one T_i at a time

JoinPathAdjacentList(T_i) = $T_{[buf]}$

Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)

JoinPathAdjacentList($T_{[buf]}$) = T_i

Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)

$T_{[buf]}$ + = T_i

Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$

vertexes

Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments

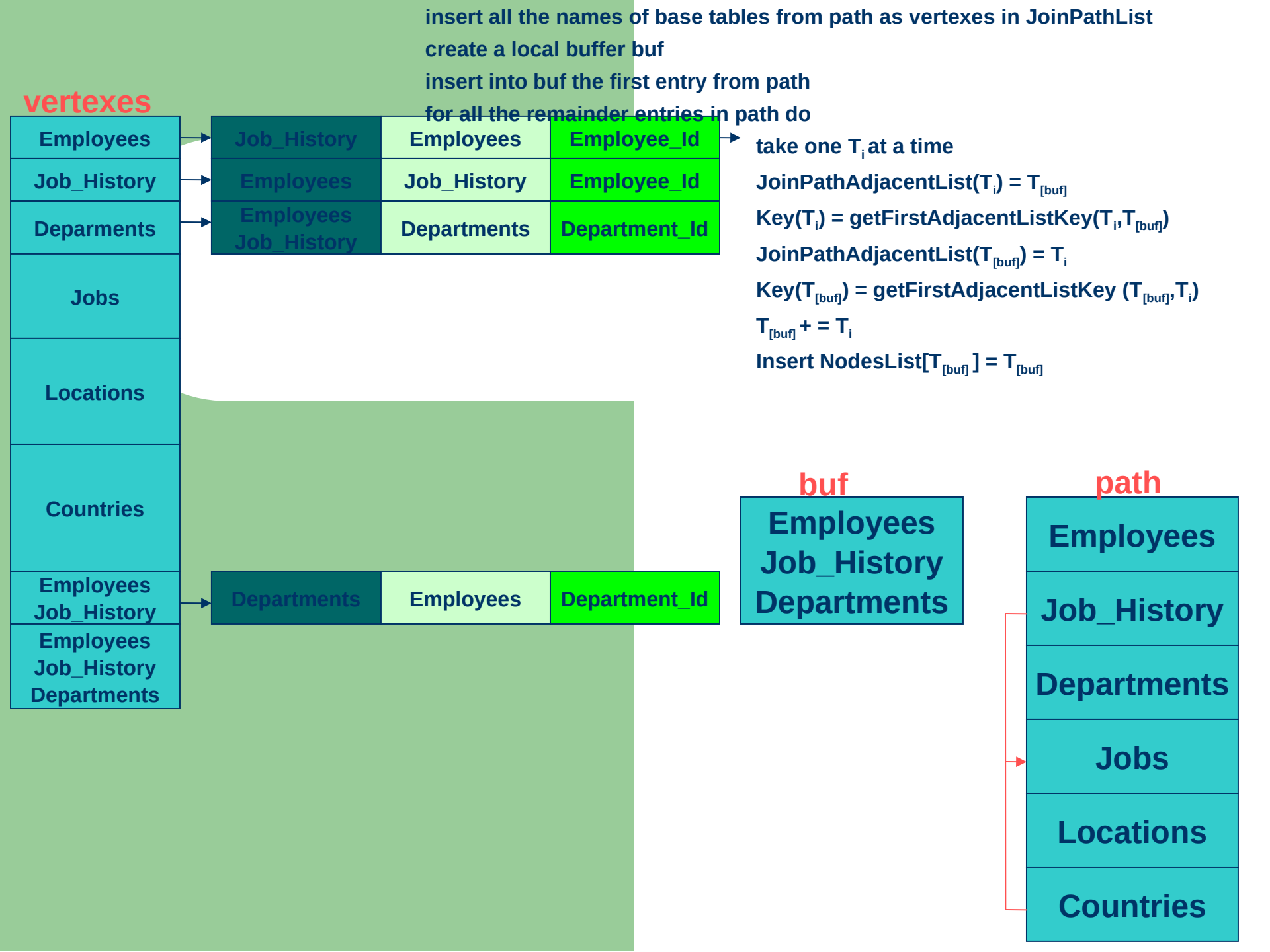
Job_History	Employees	Employee_Id
Employees	Job_History	Employee_Id
Employees Job_History	Departments	Department_Id
Departments	Employees	Department_Id

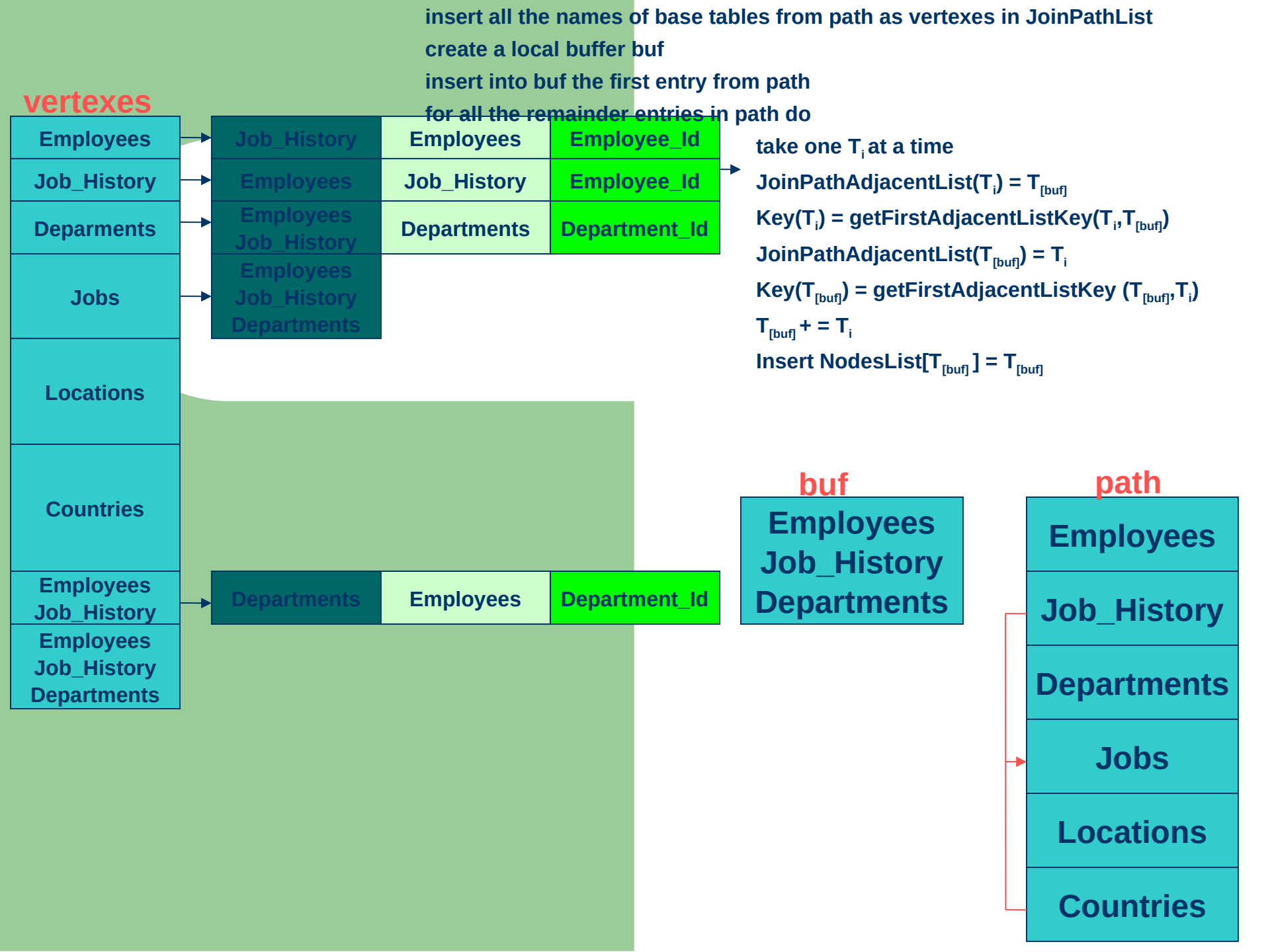
buf

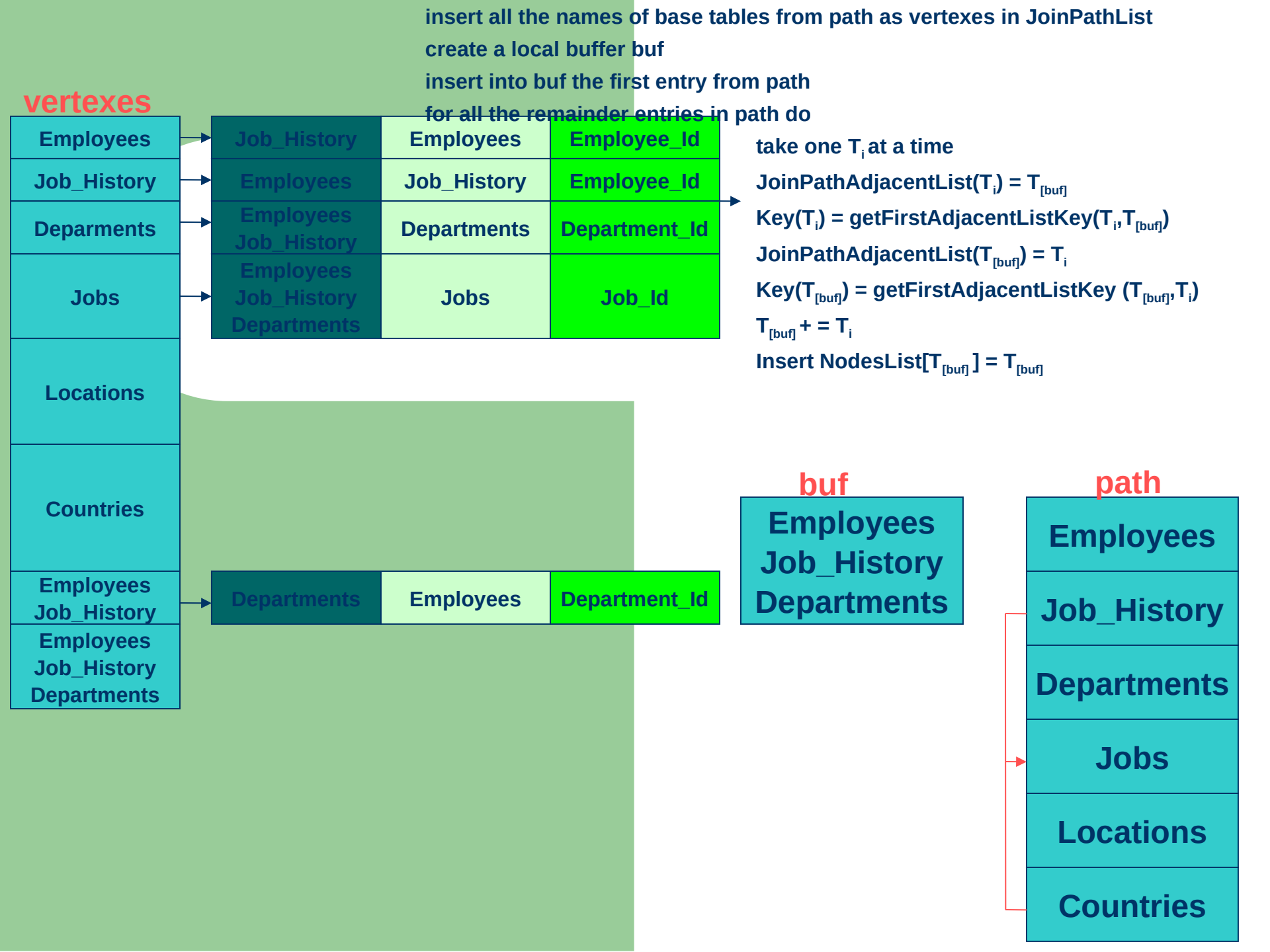
Employees
Job_History
Departments

path

Employees
Job_History
Departments
Jobs
Locations
Countries


insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
take one T_i at a time
JoinPathAdjacentList(T_i) = $T_{[buf]}$
Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)
JoinPathAdjacentList($T_{[buf]}$) = T_i
Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)
$T_{[buf]}$ + = T_i
Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$
vertexes
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Job_History Employees Employee_Id
Employees Job_History Employee_Id
Employees Job_History Departments Department_Id
Departments Employees Department_Id
buf
Employees
Job_History
Departments
path
Employees
Job_History
Departments
Jobs
Locations
Countries


insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
take one T_i at a time
JoinPathAdjacentList(T_i) = $T_{[buf]}$
Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)
JoinPathAdjacentList($T_{[buf]}$) = T_i
Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)
$T_{[buf]}$ + = T_i
Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$
vertexes
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Job_History
Employees
Employee_Id
Employees
Job_History
Employee_Id
Employees Job_History
Departments
Department_Id
Employees Job_History Departments
Departments
Employees
Department_Id
buf
Employees
Job_History
Departments
path
Employees
Job_History
Departments
Jobs
Locations
Countries


insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
take one T_i at a time
JoinPathAdjacentList(T_i) = $T_{[buf]}$
Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)
JoinPathAdjacentList($T_{[buf]}$) = T_i
Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)
$T_{[buf]}$ + = T_i
Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$
vertexes
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Job_History
Employees
Employee_Id
Employees
Job_History
Employee_Id
Employees Job_History
Departments
Department_Id
Employees Job_History Departments
Jobs
Job_Id
Departments
Employees
Department_Id
buf
Employees
Job_History
Departments
path
Employees
Job_History
Departments
Jobs
Locations
Countries

insert all the names of base tables from path as vertexes in JoinPathList

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one T_i at a time

JoinPathAdjacentList(T_i) = $T_{[buf]}$

Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)

JoinPathAdjacentList($T_{[buf]}$) = T_i

Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)

$T_{[buf]}$ + = T_i

Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$

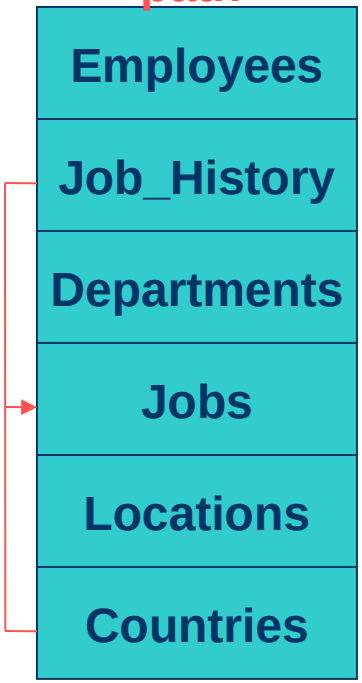
vertexes

Employees	Job_History	Employees	Employee_Id
Job_History	Employees	Job_History	Employee_Id
Deparments	Employees Job_History	Departments	Department_Id
Jobs	Employees Job_History Departments	Jobs	Job_Id
Locations			
Countries			
Employees Job_History	Departments	Employees	Department_Id
Employees Job_History Departments	Jobs		

buf

Employees
Job_History
Departments

path



insert all the names of base tables from path as vertexes in JoinPathList

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one T_i at a time

JoinPathAdjacentList(T_i) = $T_{[buf]}$

Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)

JoinPathAdjacentList($T_{[buf]}$) = T_i

Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)

$T_{[buf]}$ + = T_i

Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$

vertexes

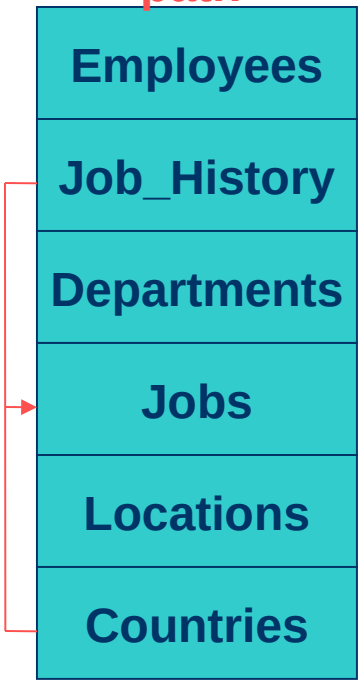
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments

Job_History	Employees	Employee_Id
Employees	Job_History	Employee_Id
Employees Job_History	Departments	Department_Id
Employees Job_History Departments	Jobs	Job_Id
Departments	Employees	Department_Id
Jobs	Job_History	Job_Id

buf

Employees
Job_History
Departments

path



insert all the names of base tables from path as vertexes in JoinPathList

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one T_i at a time

JoinPathAdjacentList(T_i) = $T_{[buf]}$

Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)

JoinPathAdjacentList($T_{[buf]}$) = T_i

Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)

$T_{[buf]}$ + = T_i

Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$

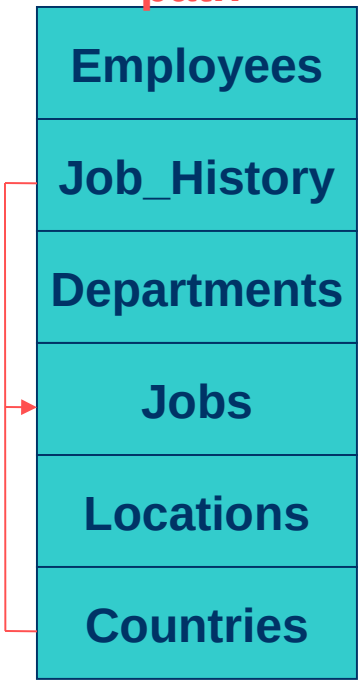
vertexes

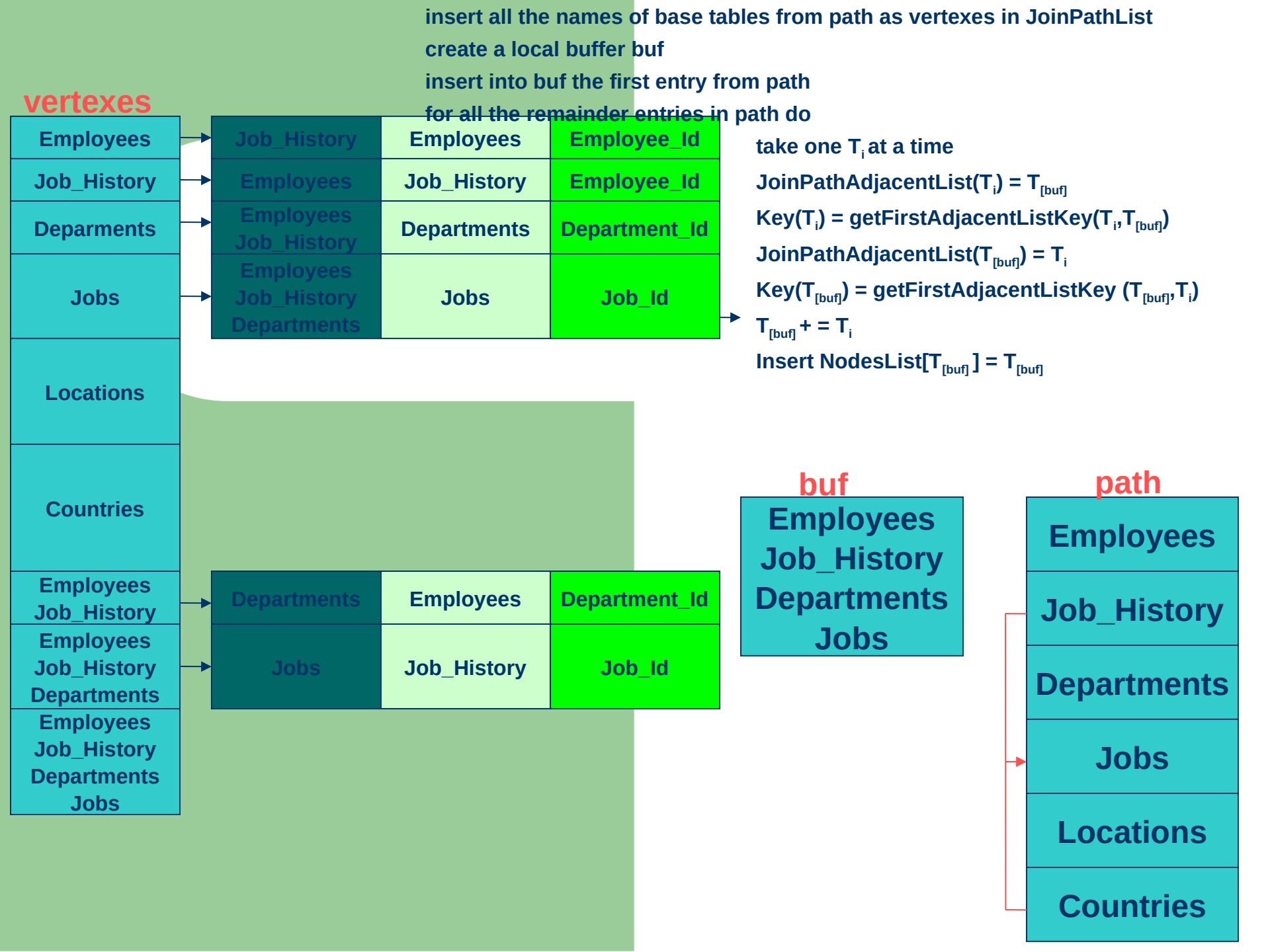
Employees	Job_History	Employees	Employee_Id
Job_History	Employees	Job_History	Employee_Id
Deparments	Employees Job_History	Departments	Department_Id
Jobs	Employees Job_History Departments	Jobs	Job_Id
Locations			
Countries			
Employees Job_History	Departments	Employees	Department_Id
Employees Job_History Departments	Jobs	Job_History	Job_Id

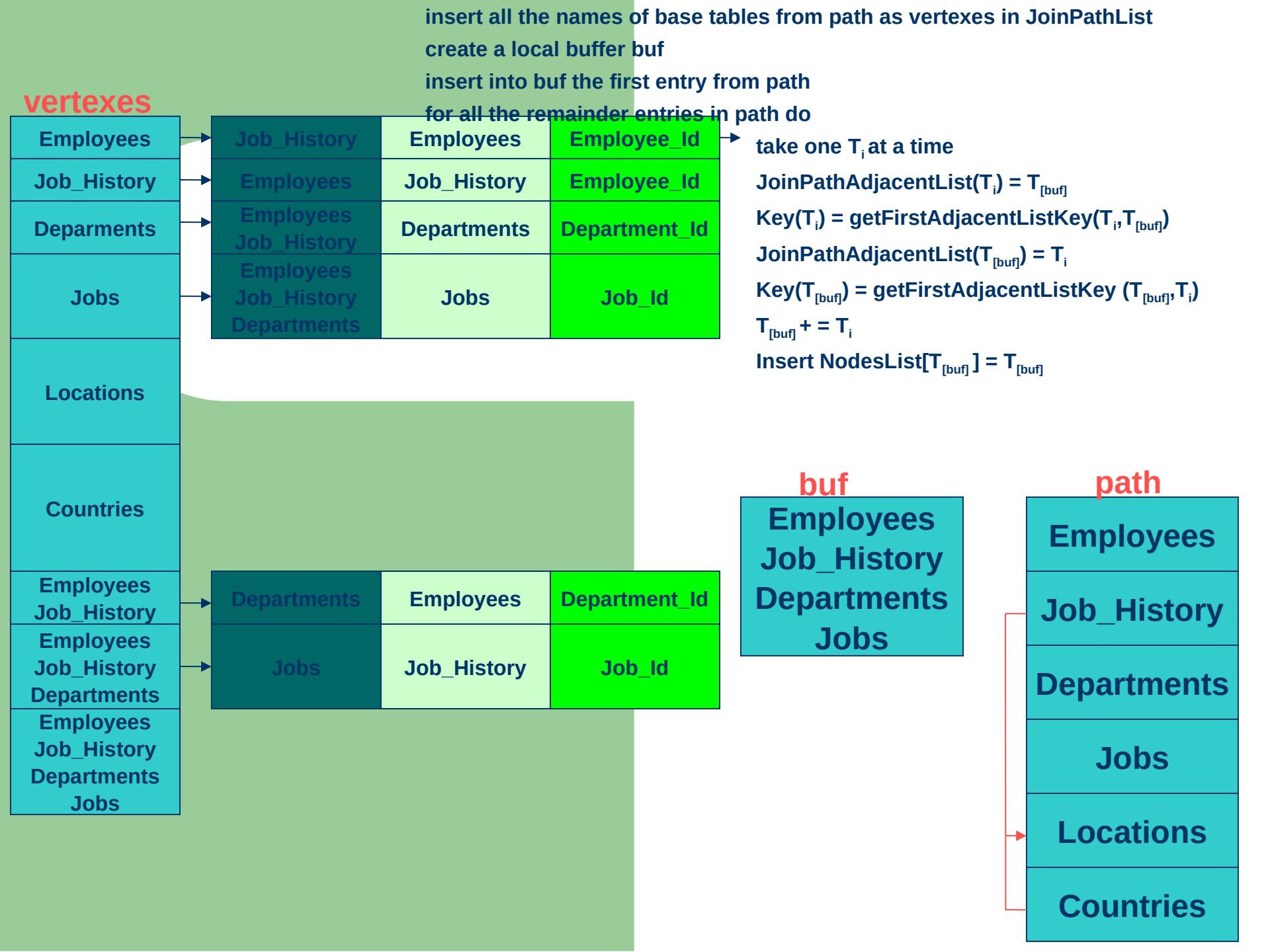
buf

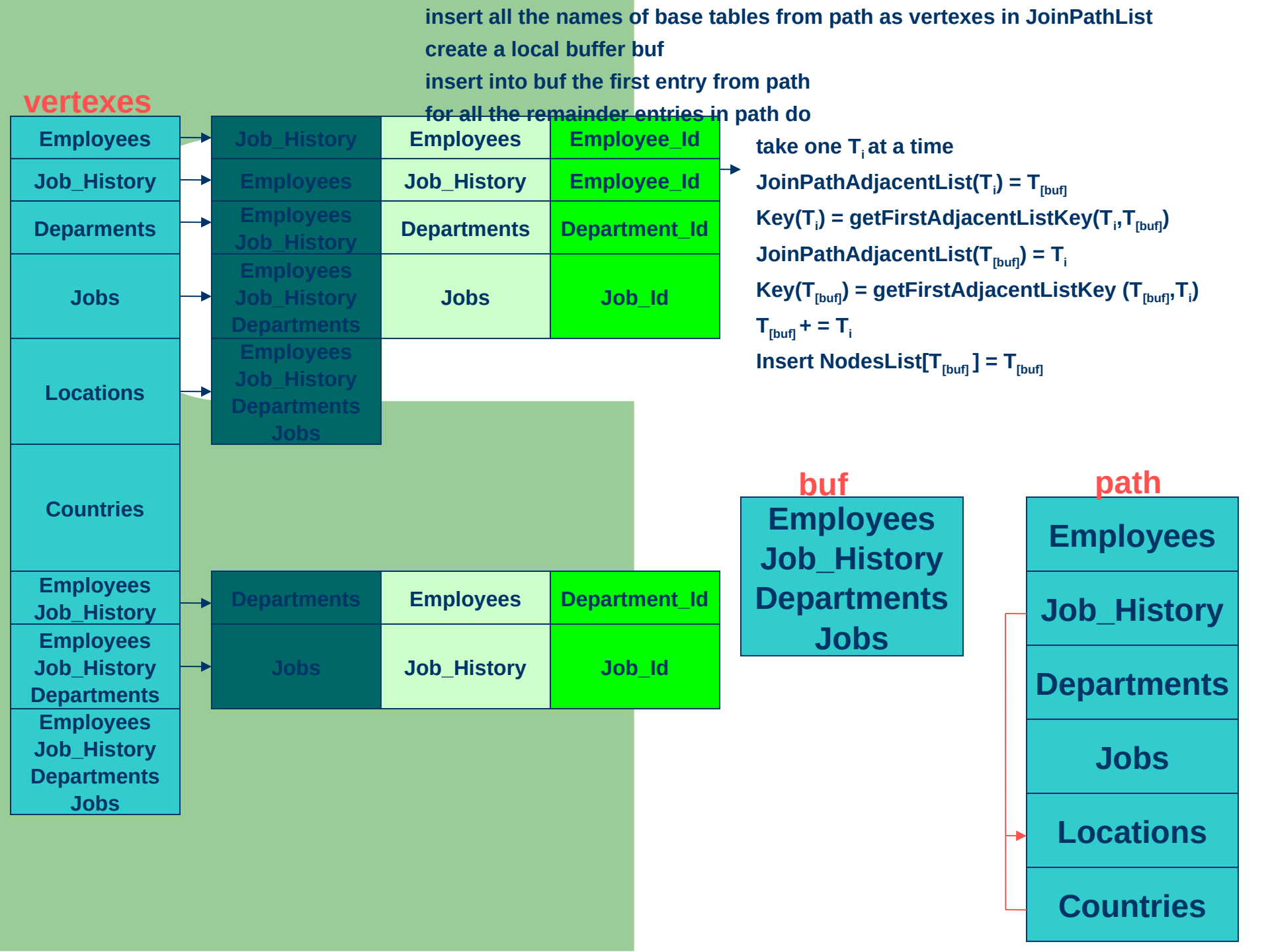
Employees
Job_History
Departments
Jobs

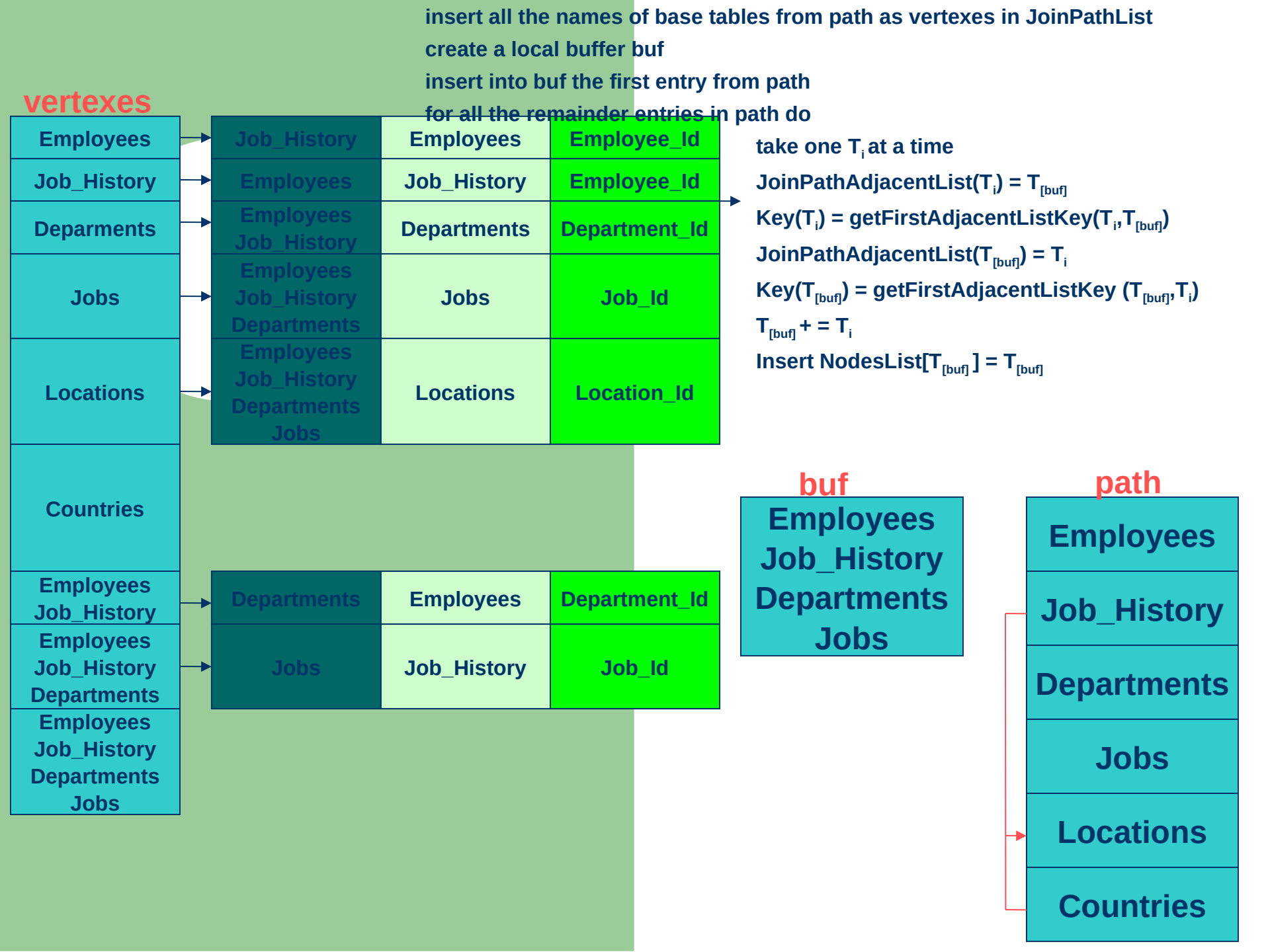
path

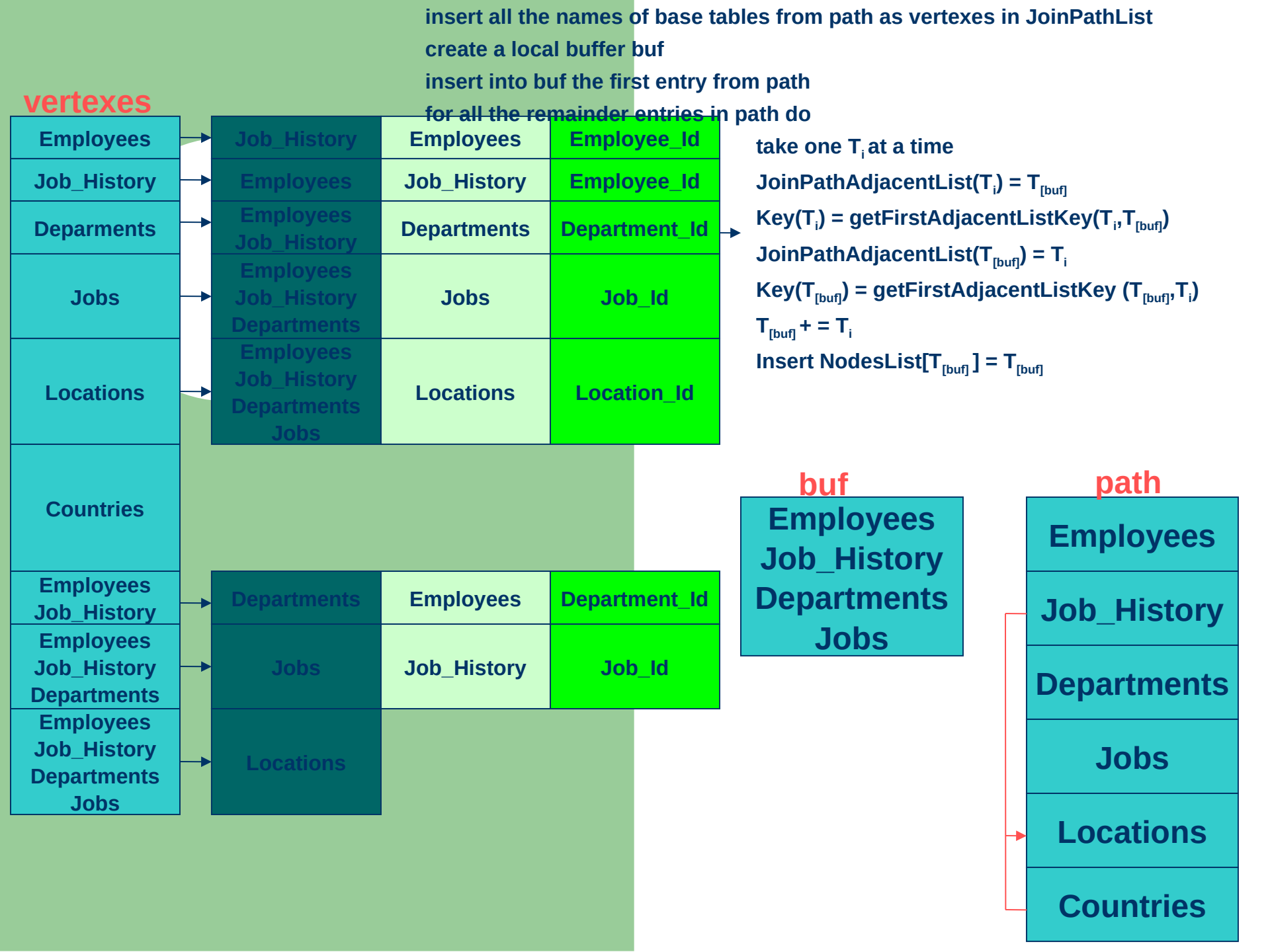



insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
take one T_i at a time
JoinPathAdjacentList(T_i) = $T_{[buf]}$
Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)
JoinPathAdjacentList($T_{[buf]}$) = T_i
Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)
$T_{[buf]}$ + = T_i
Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$
vertexes
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Job_History
Employees
Employee_Id
Employees
Job_History
Employee_Id
Employees Job_History
Departments
Department_Id
Employees Job_History Departments
Jobs
Job_Id
Departments
Employees
Department_Id
Jobs
Job_History
Job_Id
buf
Employees
Job_History
Departments
Jobs
path
Employees
Job_History
Departments
Jobs
Locations
Countries


insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
take one T_i at a time
JoinPathAdjacentList(T_i) = $T_{[buf]}$
Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)
JoinPathAdjacentList($T_{[buf]}$) = T_i
Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)
$T_{[buf]}$ + = T_i
Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$
vertexes
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Job_History
Employees
Employee_Id
Employees
Job_History
Employee_Id
Employees Job_History
Departments
Department_Id
Employees Job_History Departments
Jobs
Job_Id
Departments
Employees
Department_Id
Jobs
Job_History
Job_Id
buf
Employees
Job_History
Departments
Jobs
path
Employees
Job_History
Departments
Jobs
Locations
Countries


insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
take one T_i at a time
JoinPathAdjacentList(T_i) = $T_{[buf]}$
Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)
JoinPathAdjacentList($T_{[buf]}$) = T_i
Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)
$T_{[buf]}$ + = T_i
Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$
vertexes
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Job_History
Employees
Employee_Id
Employees
Job_History
Employee_Id
Employees Job_History
Departments
Department_Id
Employees Job_History Departments
Jobs
Jobs
Job_Id
Departments
Employees
Department_Id
Jobs
Job_History
Job_Id
buf
Employees
Job_History
Departments
Jobs
path
Employees
Job_History
Departments
Jobs
Locations
Countries


insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
take one T_i at a time
JoinPathAdjacentList(T_i) = $T_{[buf]}$
Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)
JoinPathAdjacentList($T_{[buf]}$) = T_i
Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)
$T_{[buf]}$ + = T_i
Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$
vertexes
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Job_History
Employees
Employee_Id
Employees
Job_History
Employee_Id
Employees Job_History
Departments
Department_Id
Employees Job_History Departments
Jobs
Job_Id
Employees Job_History Departments Jobs
Locations
Location_Id
Departments
Employees
Department_Id
Jobs
Job_History
Job_Id
buf
Employees
Job_History
Departments
Jobs
path
Employees
Job_History
Departments
Jobs
Locations
Countries


insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
take one T_i at a time
JoinPathAdjacentList(T_i) = $T_{[buf]}$
Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)
JoinPathAdjacentList($T_{[buf]}$) = T_i
Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)
$T_{[buf]}$ + = T_i
Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$
vertexes
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Job_History
Employees
Employee_Id
Employees
Job_History
Employee_Id
Employees Job_History
Departments
Department_Id
Employees Job_History Departments
Jobs
Job_Id
Employees Job_History Departments Jobs
Locations
Location_Id
Departments
Employees
Department_Id
Jobs
Job_History
Job_Id
Locations
buf
Employees
Job_History
Departments
Jobs
path
Employees
Job_History
Departments
Jobs
Locations
Countries

insert all the names of base tables from path as vertexes in JoinPathList

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one T_i at a time

$JoinPathAdjacentList(T_i) = T_{[buf]}$

$Key(T_i) = getFirstAdjacentListKey(T_i, T_{[buf]})$

$JoinPathAdjacentList(T_{[buf]}) = T_i$

$Key(T_{[buf]}) = getFirstAdjacentListKey(T_{[buf]}, T_i)$

$T_{[buf]} += T_i$

$Insert\ NodesList[T_{[buf]}] = T_{[buf]}$

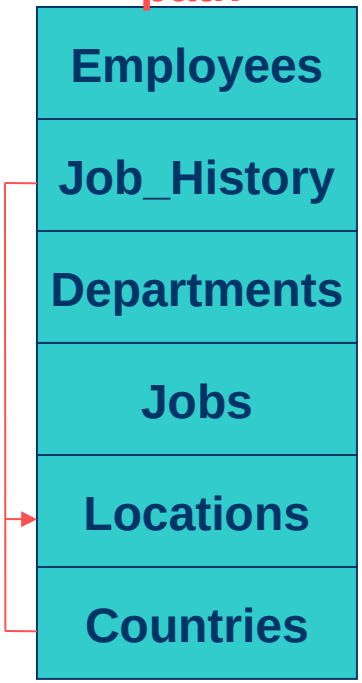
vertexes

Employees	Job_History	Employees	Employee_Id
Job_History	Employees	Job_History	Employee_Id
Deparments	Employees Job_History	Departments	Department_Id
Jobs	Employees Job_History Departments	Jobs	Job_Id
Locations	Employees Job_History Departments Jobs	Locations	Location_Id
Countries			
Employees Job_History	Departments	Employees	Department_Id
Employees Job_History Departments	Jobs	Job_History	Job_Id
Employees Job_History Departments Jobs	Locations	Departments	Location_Id

buf

Employees
Job_History
Departments
Jobs

path



insert all the names of base tables from path as vertexes in JoinPathList

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one T_i at a time

JoinPathAdjacentList(T_i) = $T_{[buf]}$

Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)

JoinPathAdjacentList($T_{[buf]}$) = T_i

Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)

$T_{[buf]}$ + = T_i

Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$

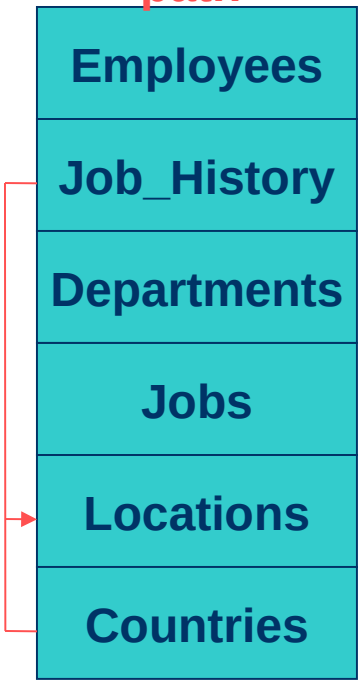
vertexes

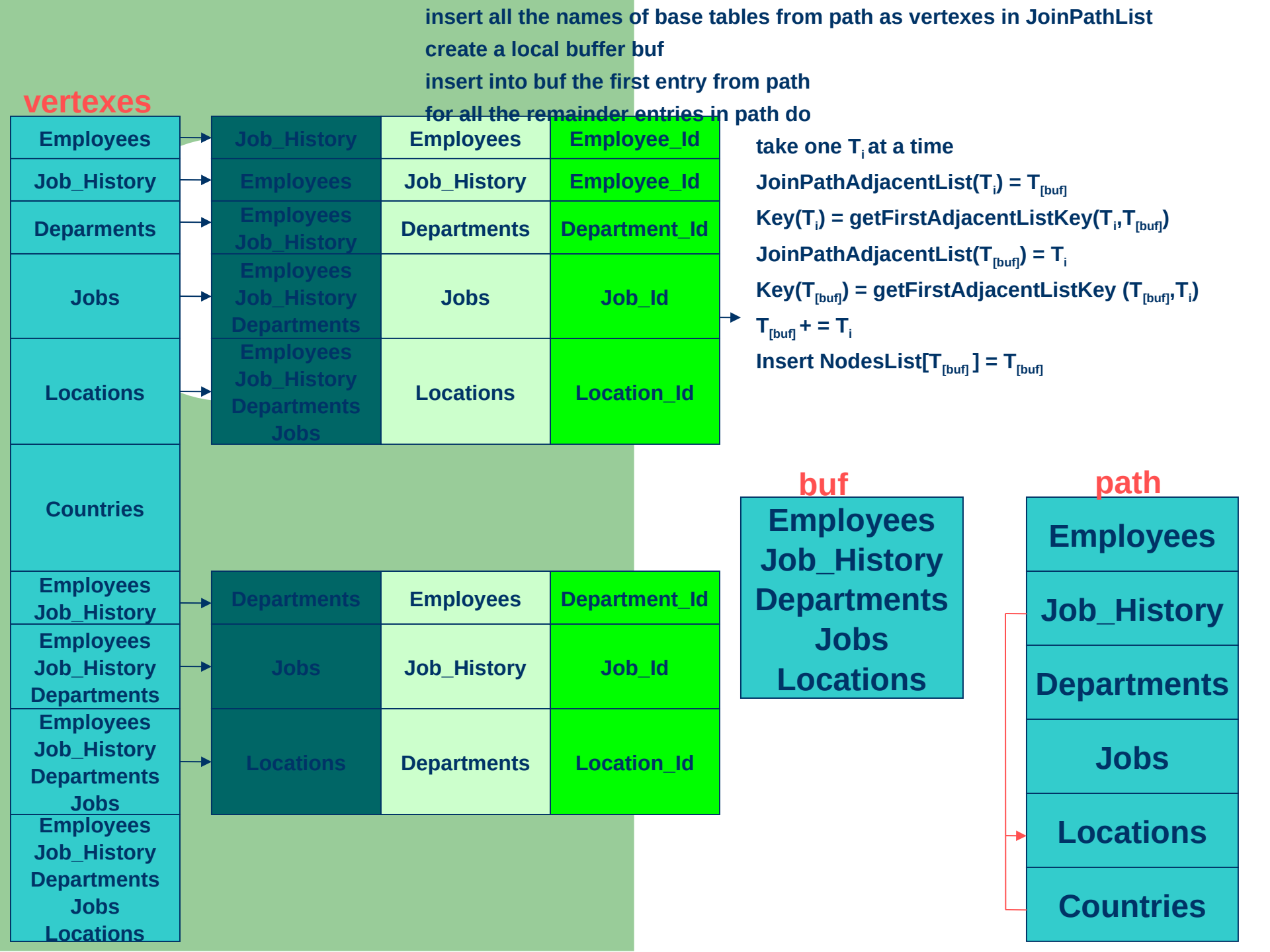
Employees	Job_History	Employees	Employee_Id
Job_History	Employees	Job_History	Employee_Id
Deparments	Employees Job_History	Departments	Department_Id
Jobs	Employees Job_History Departments	Jobs	Job_Id
Locations	Employees Job_History Departments Jobs	Locations	Location_Id
Countries			
Employees Job_History	Departments	Employees	Department_Id
Employees Job_History Departments	Jobs	Job_History	Job_Id
Employees Job_History Departments Jobs	Locations	Departments	Location_Id

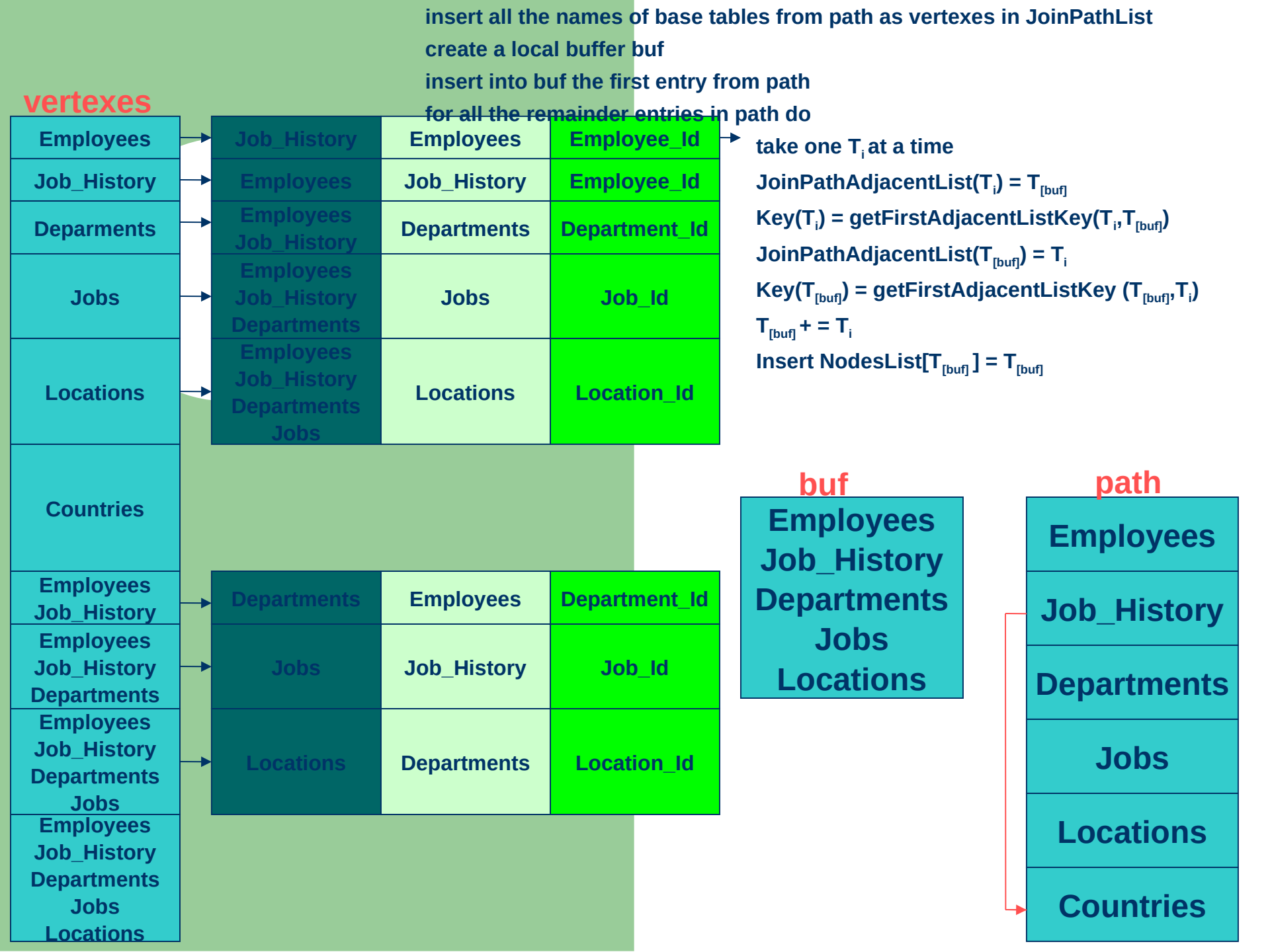
buf

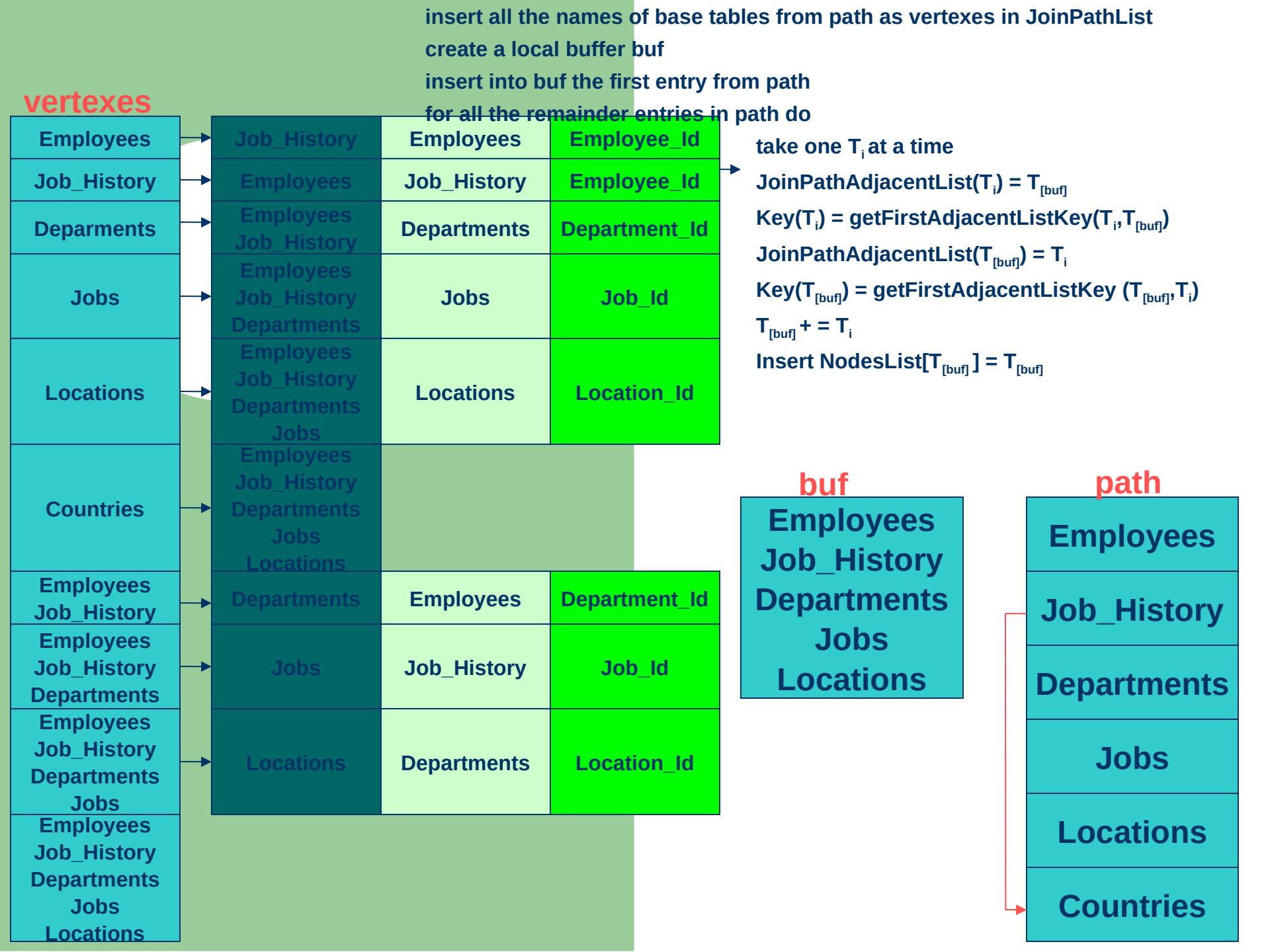
Employees
Job_History
Departments
Jobs
Locations

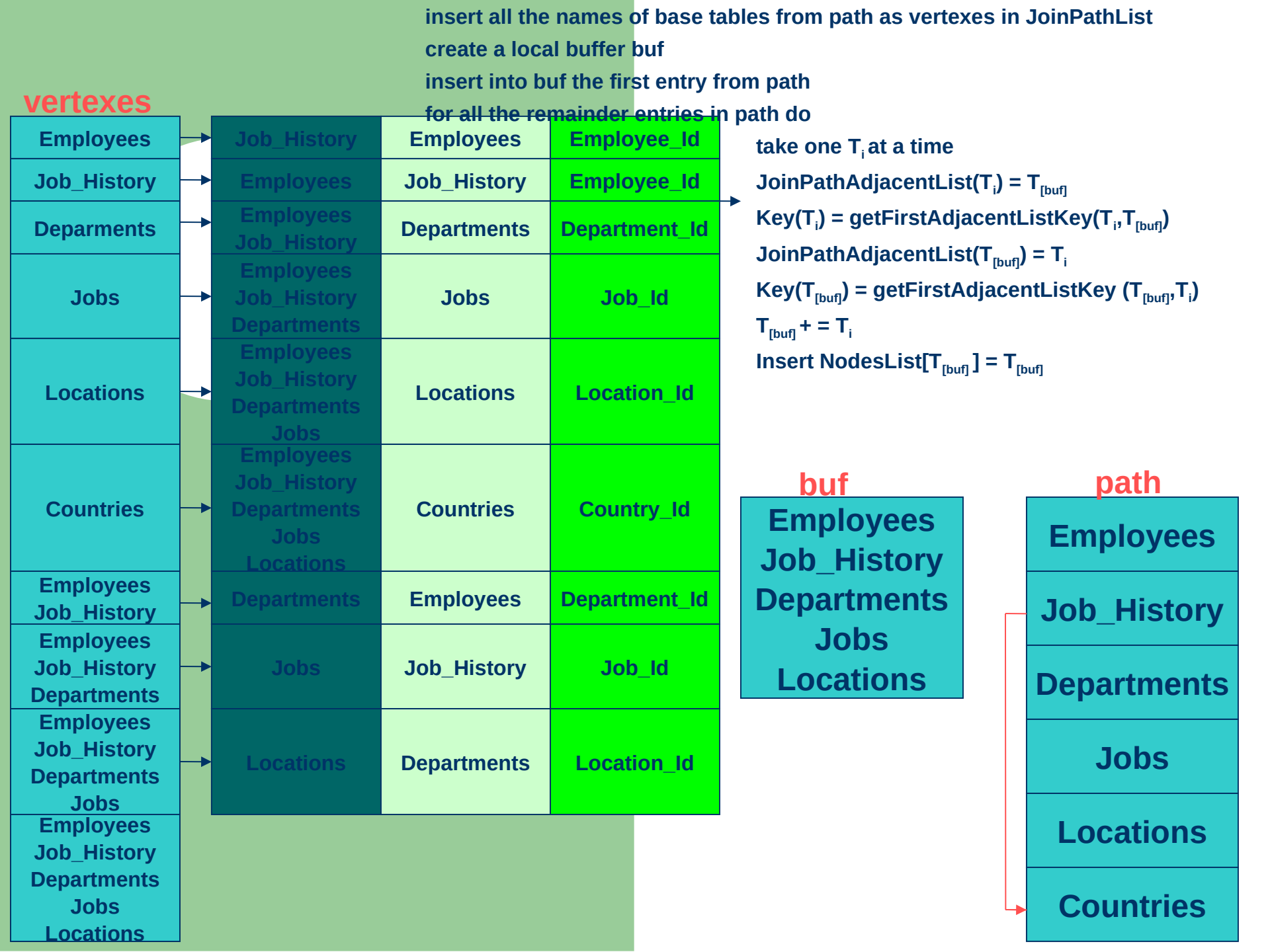
path

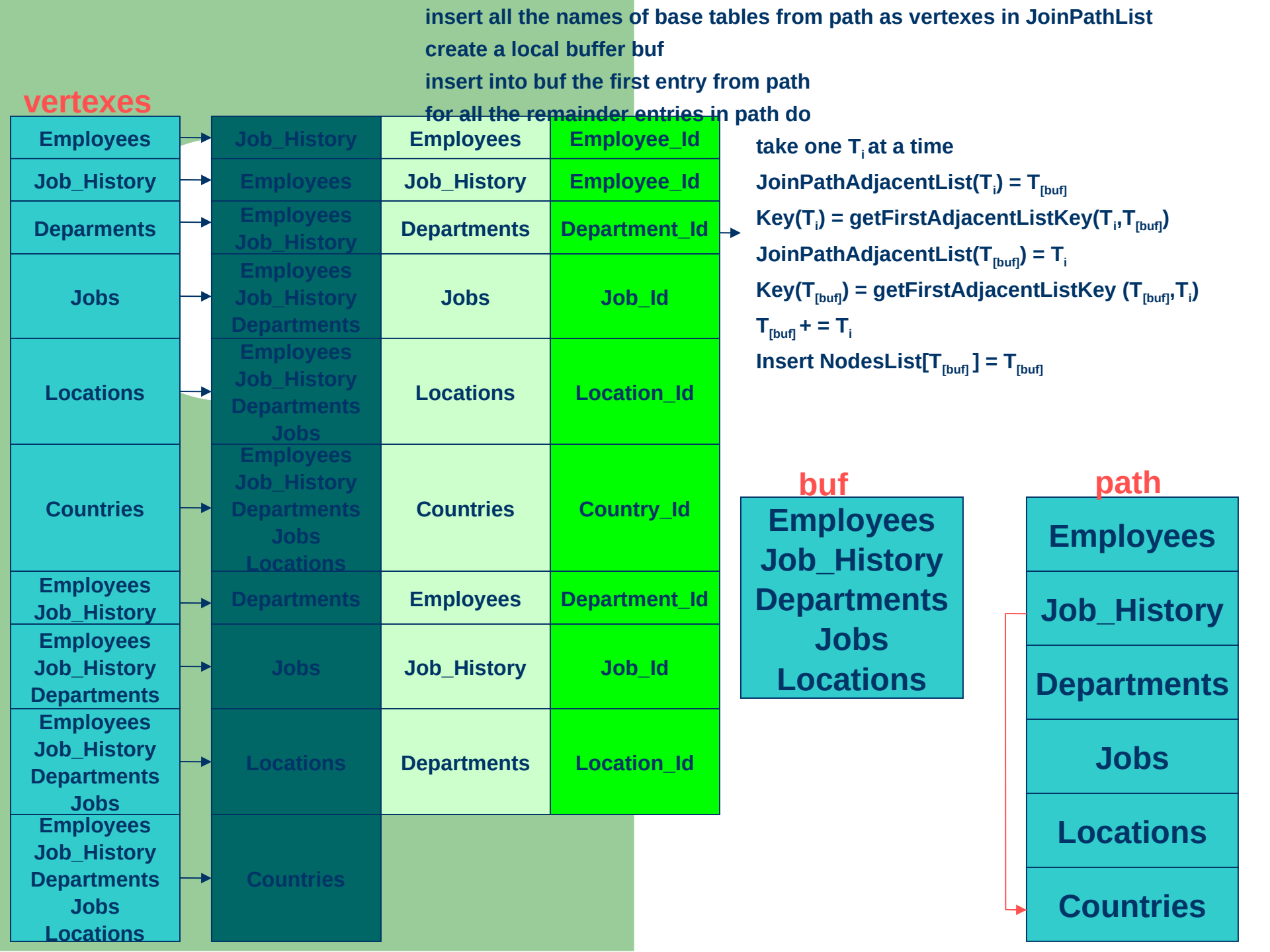


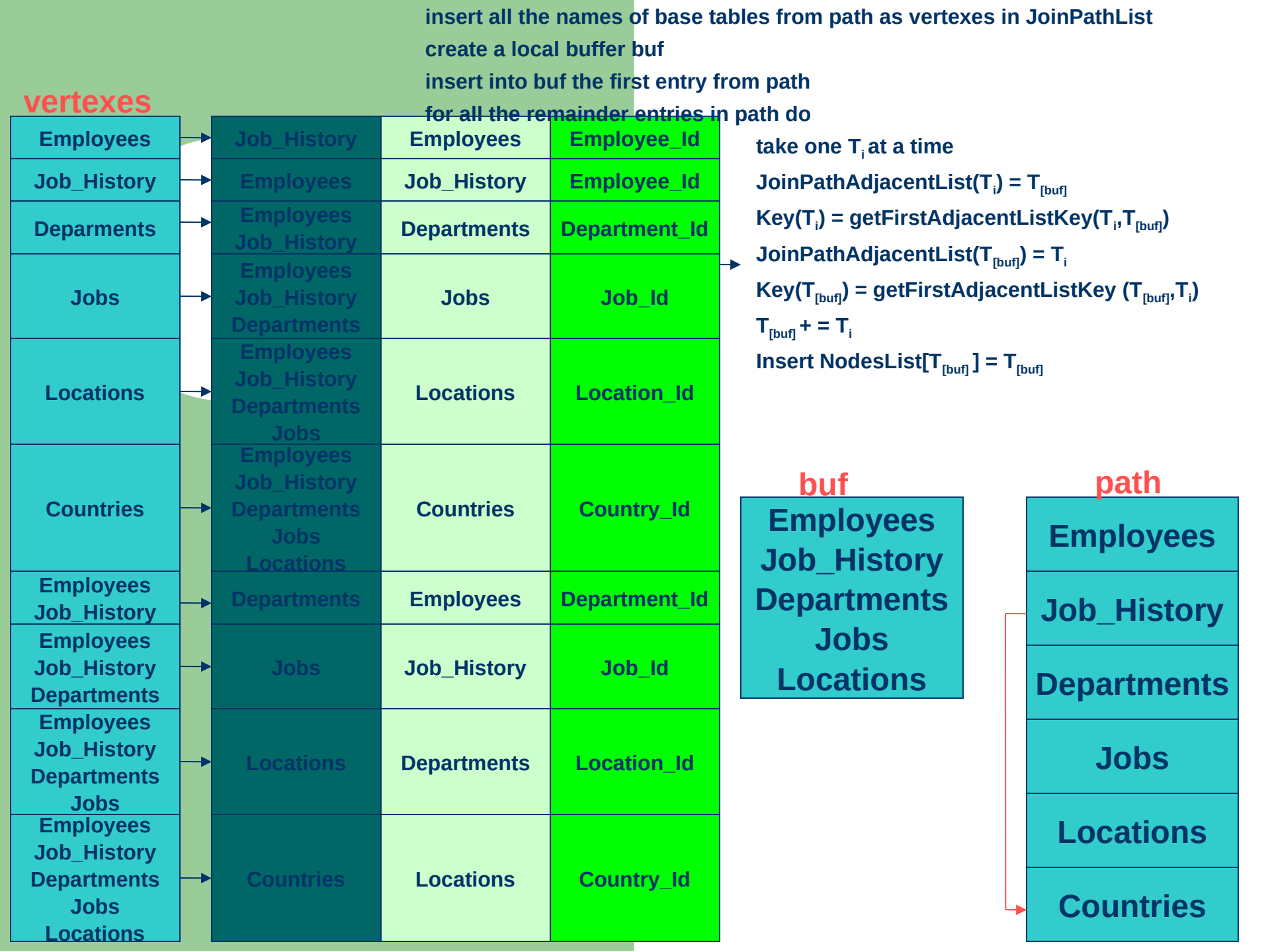

insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
take one T_i at a time
JoinPathAdjacentList(T_i) = $T_{[buf]}$
Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)
JoinPathAdjacentList($T_{[buf]}$) = T_i
Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)
$T_{[buf]}$ + = T_i
Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$
vertexes
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Job_History
Employees
Employee_Id
Employees
Job_History
Employee_Id
Employees Job_History
Departments
Department_Id
Employees Job_History Departments
Jobs
Job_Id
Employees Job_History Departments Jobs
Locations
Location_Id
Departments
Employees
Department_Id
Jobs
Job_History
Job_Id
Locations
Departments
Location_Id
buf
Employees
Job_History
Departments
Jobs
Locations
path
Employees
Job_History
Departments
Jobs
Locations
Countries

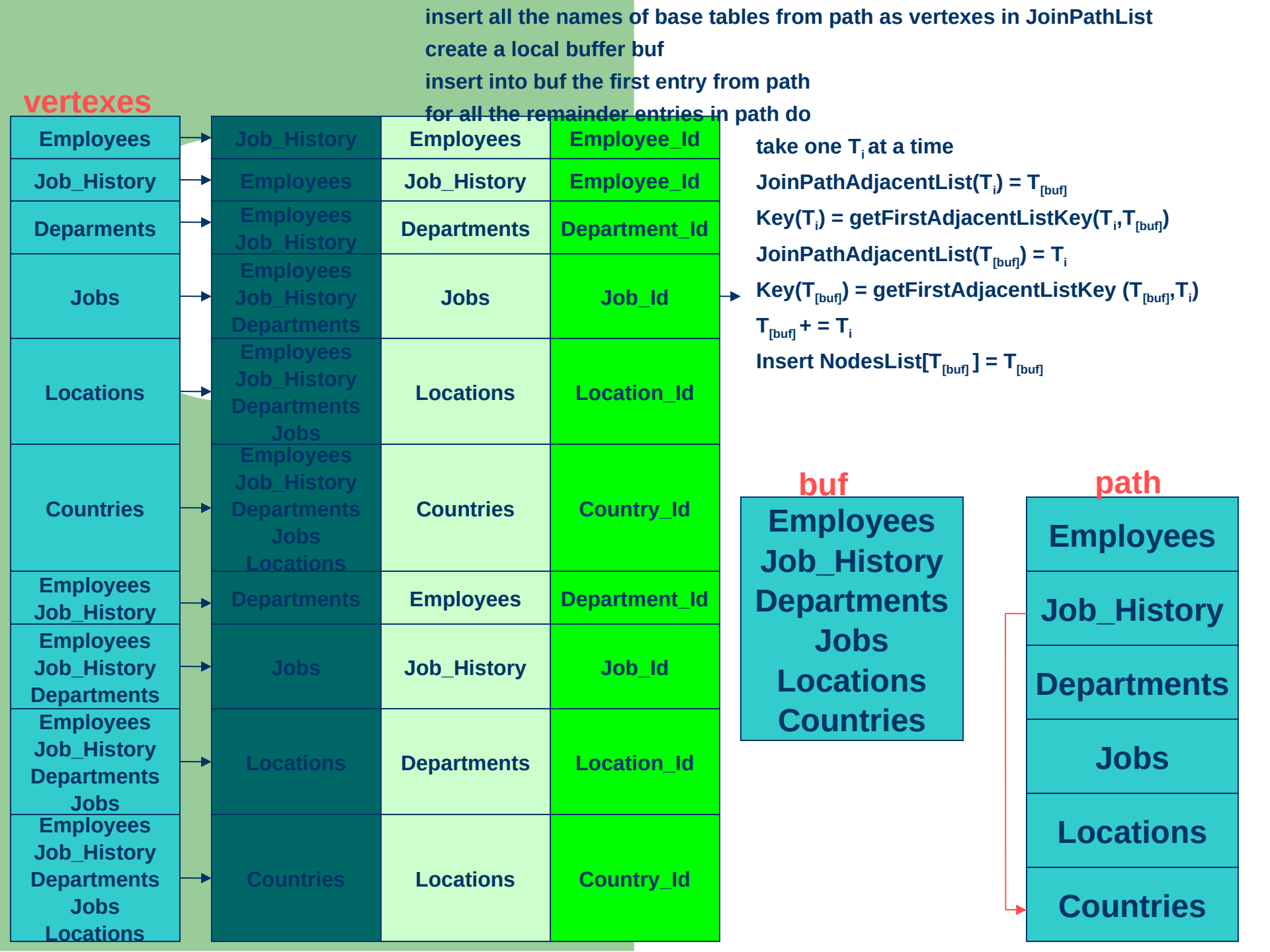

insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
take one T_i at a time
JoinPathAdjacentList(T_i) = $T_{[buf]}$
Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)
JoinPathAdjacentList($T_{[buf]}$) = T_i
Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)
$T_{[buf]}$ + = T_i
Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$
vertexes
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Job_History
Employees
Employee_Id
Employees
Job_History
Employee_Id
Employees Job_History
Departments
Department_Id
Employees Job_History Departments
Jobs
Job_Id
Employees Job_History Departments Jobs
Locations
Location_Id
Departments
Employees
Department_Id
Jobs
Job_History
Job_Id
Locations
Departments
Location_Id
buf
Employees
Job_History
Departments
Jobs
Locations
path
Employees
Job_History
Departments
Jobs
Locations
Countries

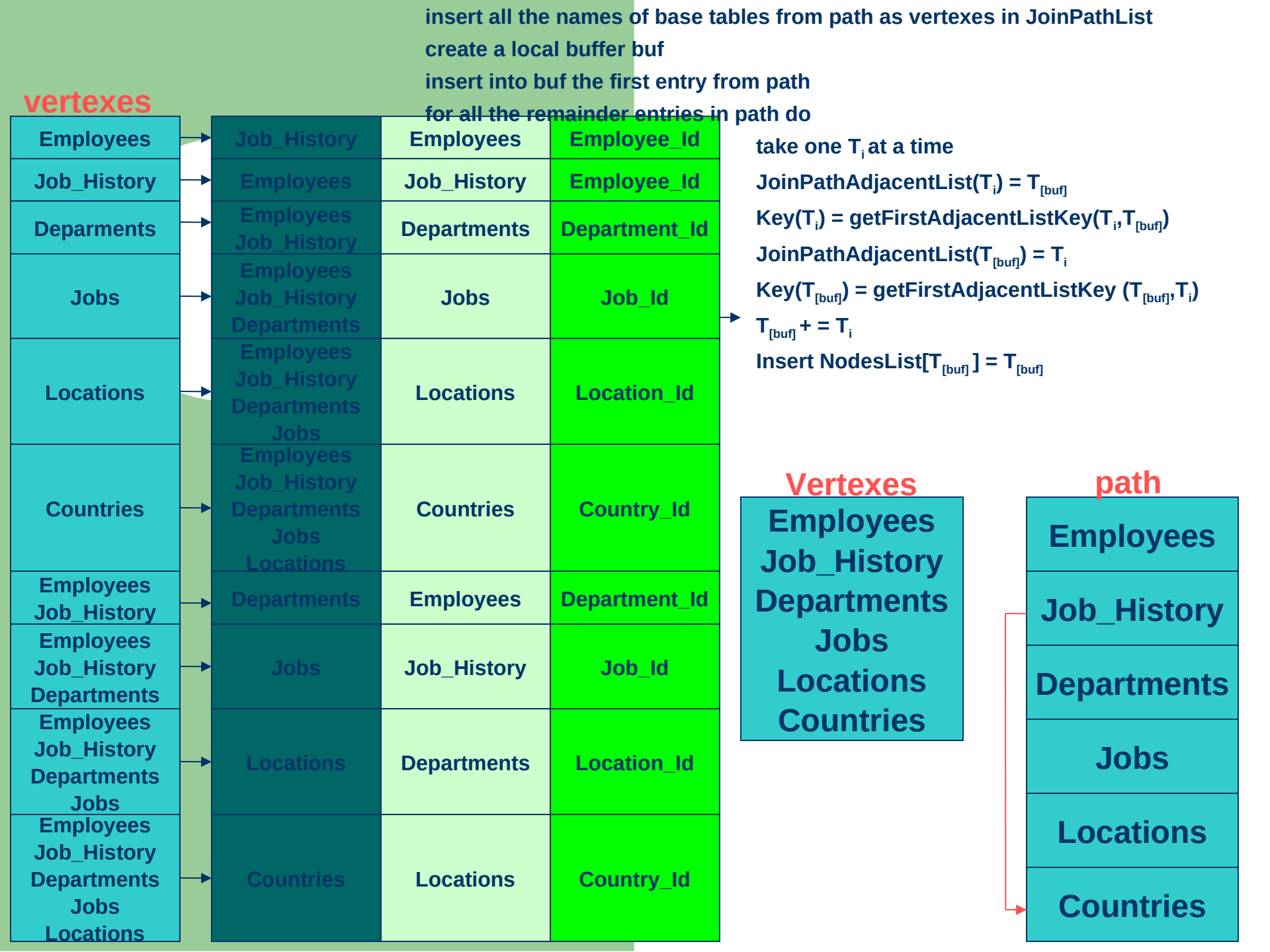

insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
take one T_i at a time
JoinPathAdjacentList(T_i) = $T_{[buf]}$
Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)
JoinPathAdjacentList($T_{[buf]}$) = T_i
Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)
$T_{[buf]}$ + = T_i
Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$
vertexes
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Job_History
Employees
Job_History
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Departments
Jobs
Locations
Employees
Job_History
Departments
Employee_Id
Employee_Id
Department_Id
Jobs
Job_Id
Locations
Location_Id
Employees
Department_Id
Jobs
Job_History
Job_Id
Locations
Departments
Location_Id
buf
Employees
Job_History
Departments
Jobs
Locations
path
Employees
Job_History
Departments
Jobs
Locations
Countries

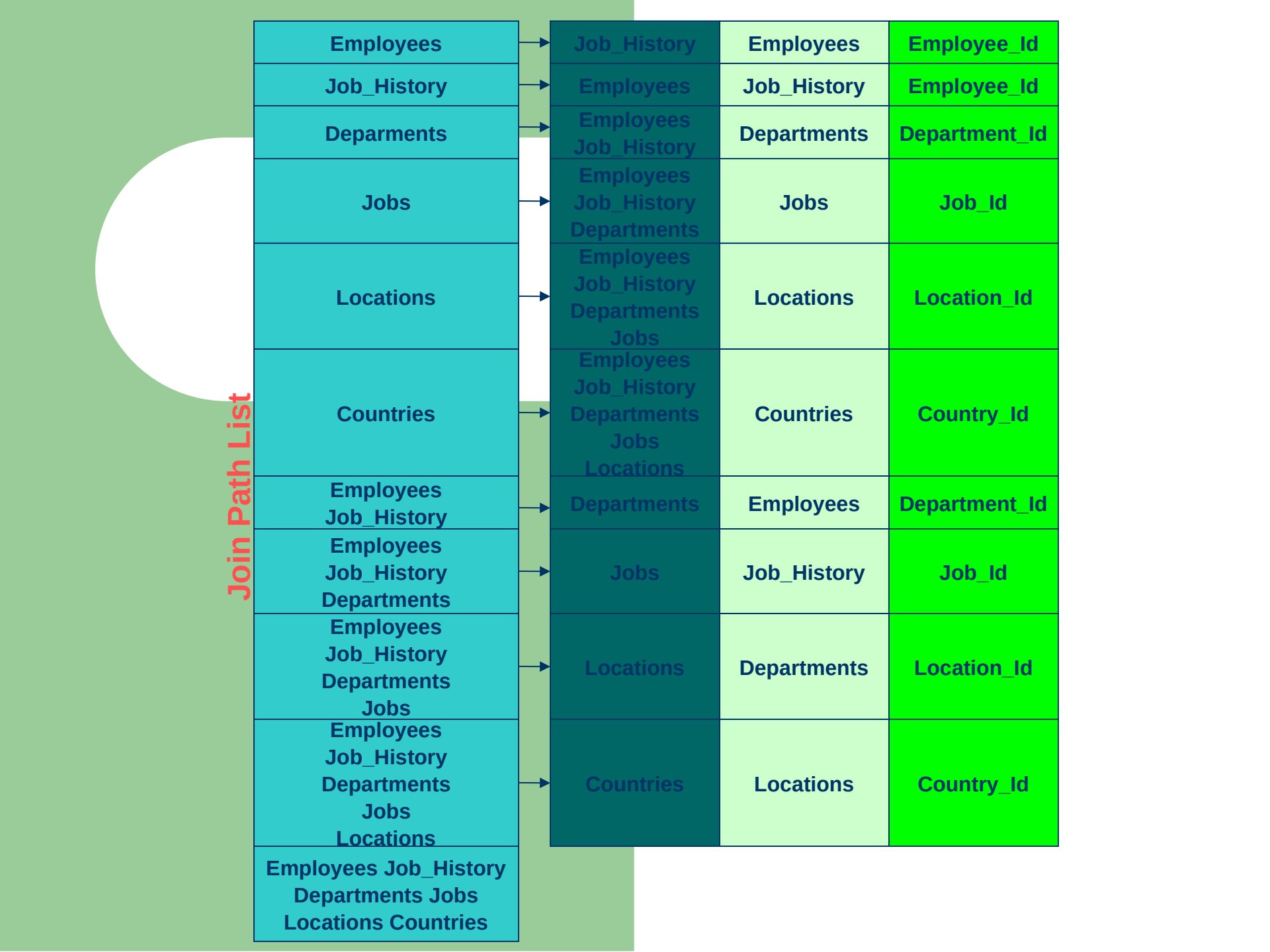

insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
take one T_i at a time
JoinPathAdjacentList(T_i) = $T_{[buf]}$
Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)
JoinPathAdjacentList($T_{[buf]}$) = T_i
Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)
$T_{[buf]}$ + = T_i
Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$
vertexes
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Job_History
Employees
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees
Job_History
Departments
Jobs
Locations
Countries
Employee_Id
Employee_Id
Department_Id
Job_Id
Location_Id
Country_Id
Department_Id
Job_Id
Location_Id
buf
Employees
Job_History
Departments
Jobs
Locations
path
Employees
Job_History
Departments
Jobs
Locations
Countries


insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
take one T_i at a time
JoinPathAdjacentList(T_i) = $T_{[buf]}$
Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)
JoinPathAdjacentList($T_{[buf]}$) = T_i
Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)
$T_{[buf]}$ + = T_i
Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$
vertexes
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees
Job_History
Employees
Job_History
Departments
Jobs
Locations
Countries
Job_History
Employees
Job_History
Employees
Job_History
Departments
Employees
Job_History
Departments
Jobs
Employees
Job_History
Departments
Jobs
Locations
Employees
Job_History
Departments
Jobs
Locations
Countries
Employees
Job_History
Departments
Jobs
Locations
Employee_Id
Employee_Id
Department_Id
Job_Id
Location_Id
Country_Id
Department_Id
Job_Id
Location_Id
buf
Employees
Job_History
Departments
Jobs
Locations
path
Employees
Job_History
Departments
Jobs
Locations
Countries

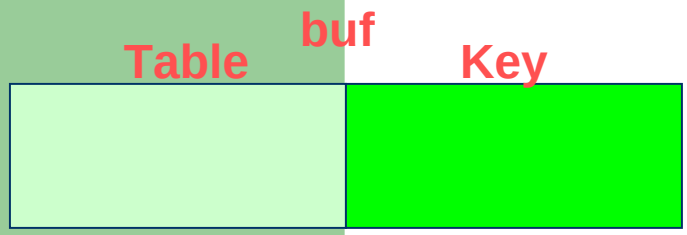

insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
take one T_i at a time
JoinPathAdjacentList(T_i) = $T_{[buf]}$
Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)
JoinPathAdjacentList($T_{[buf]}$) = T_i
Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)
$T_{[buf]}$ + = T_i
Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$
vertexes
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Job_History
Employees
Job_History
Departments
Jobs
Locations
Countries
Employee_Id
Employee_Id
Department_Id
Job_Id
Location_Id
Country_Id
Department_Id
Job_Id
Location_Id
Country_Id
buf
Employees
Job_History
Departments
Jobs
Locations
path
Employees
Job_History
Departments
Jobs
Locations
Countries


insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
take one T_i at a time
JoinPathAdjacentList(T_i) = $T_{[buf]}$
Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)
JoinPathAdjacentList($T_{[buf]}$) = T_i
Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)
$T_{[buf]}$ + = T_i
Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$
vertexes
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Job_History
Employees
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Departments
Jobs
Locations
Countries
Employee_Id
Job_Id
Location_Id
Country_Id
Department_Id
buf
Employees
Job_History
Departments
Jobs
Locations
Countries
path
Employees
Job_History
Departments
Jobs
Locations
Countries


insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
take one T_i at a time
JoinPathAdjacentList(T_i) = $T_{[buf]}$
Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)
JoinPathAdjacentList($T_{[buf]}$) = T_i
Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)
$T_{[buf]}$ + = T_i
Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$
vertexes
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Jobs
Employees Job_History Departments Jobs Locations
Job_History
Employees
Departments
Jobs
Locations
Countries
Employees
Job_History
Departments
Jobs
Locations
Employee_Id
Employee_Id
Department_Id
Job_Id
Location_Id
Country_Id
Department_Id
Job_Id
Location_Id
Country_Id
Vertexes
Employees
Job_History
Departments
Jobs
Locations
Countries
path
Employees
Job_History
Departments
Jobs
Locations
Countries


Join Path List
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees Job_History Departments Jobs Locations Countries
Job_History
Employees
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees
Job_History
Departments
Jobs
Locations
Countries
Employee_Id
Employee_Id
Department_Id
Job_Id
Location_Id
Country_Id
Department_Id
Job_Id
Location_Id
Country_Id

- create a structure buf with 2 fields: Table and Key
  for all the tables in JoinPathList going downward do
    take one $T_{[i]}$ at a time
    for all Base Tables in $T_{[i]}$ do
      take one T_k at a time
      for every buf.Table = T_k do
        if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then
        InheritedKey($T_{[i]}$) += buf.key
    if T_l is the table from which comes Key($T_{[i]}$) then
      buf.Table = T_l
      buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

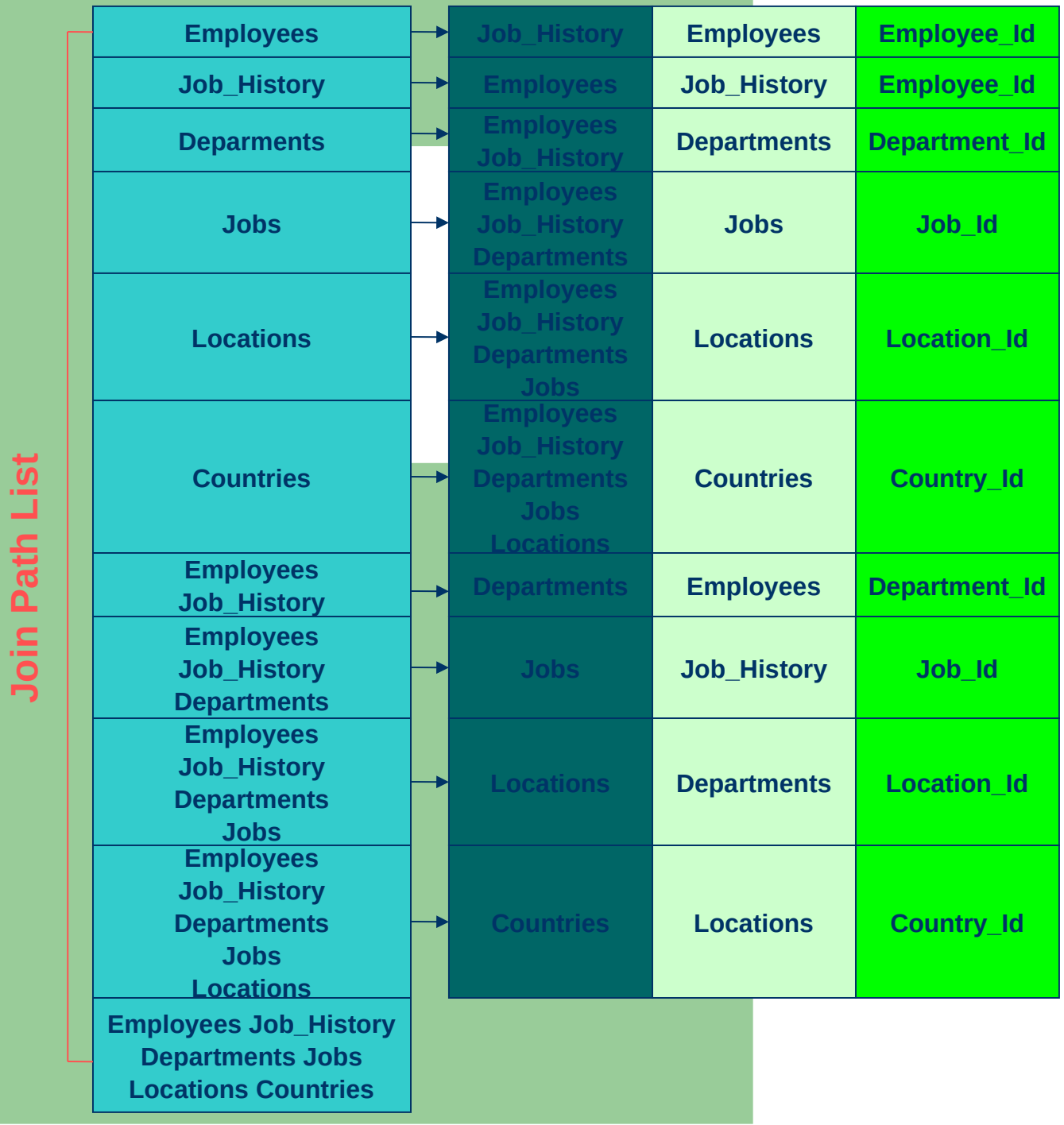
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then
InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)


Join Path List
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees Job_History Departments Jobs Locations Countries
Job_History
Employees
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees
Job_History
Departments
Jobs
Locations
Countries
Employees
Job_History
Departments
Jobs
Locations
Employees
Job_History
Departments
Jobs
Locations
Employee_Id
Employee_Id
Department_Id
Job_Id
Location_Id
Country_Id
Department_Id
Job_Id
Location_Id
Country_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

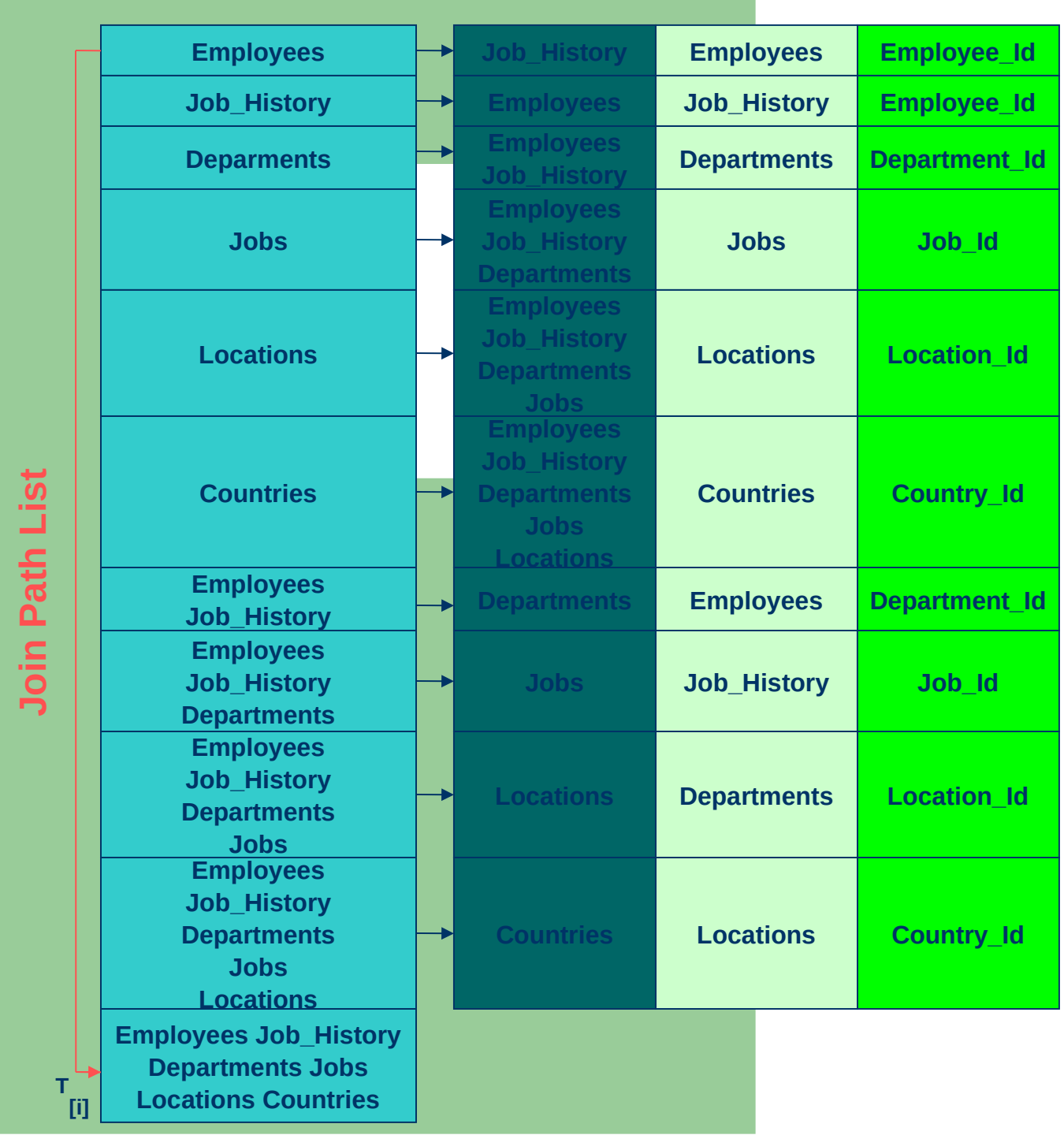
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then
InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)


Join Path List
T [i]
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees Job_History Departments Jobs Locations Countries
Job_History
Employees
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Departments
Jobs
Locations
Countries
Employees
Job_History
Departments
Jobs
Locations
Employee_Id
Employee_Id
Department_Id
Job_Id
Location_Id
Country_Id
Department_Id
Job_Id
Location_Id
Country_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

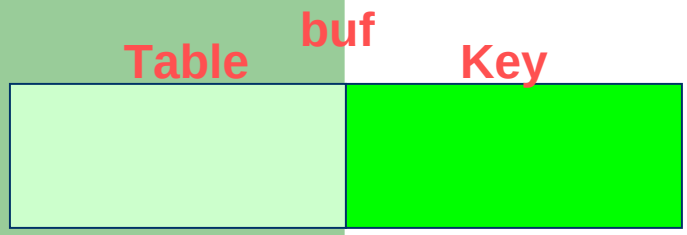
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

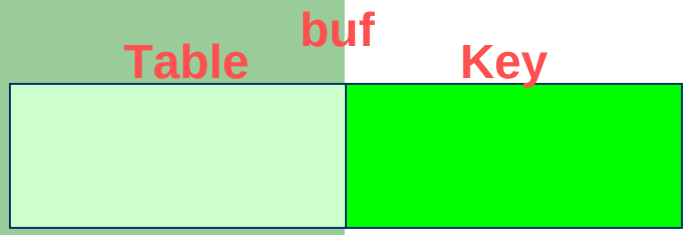
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

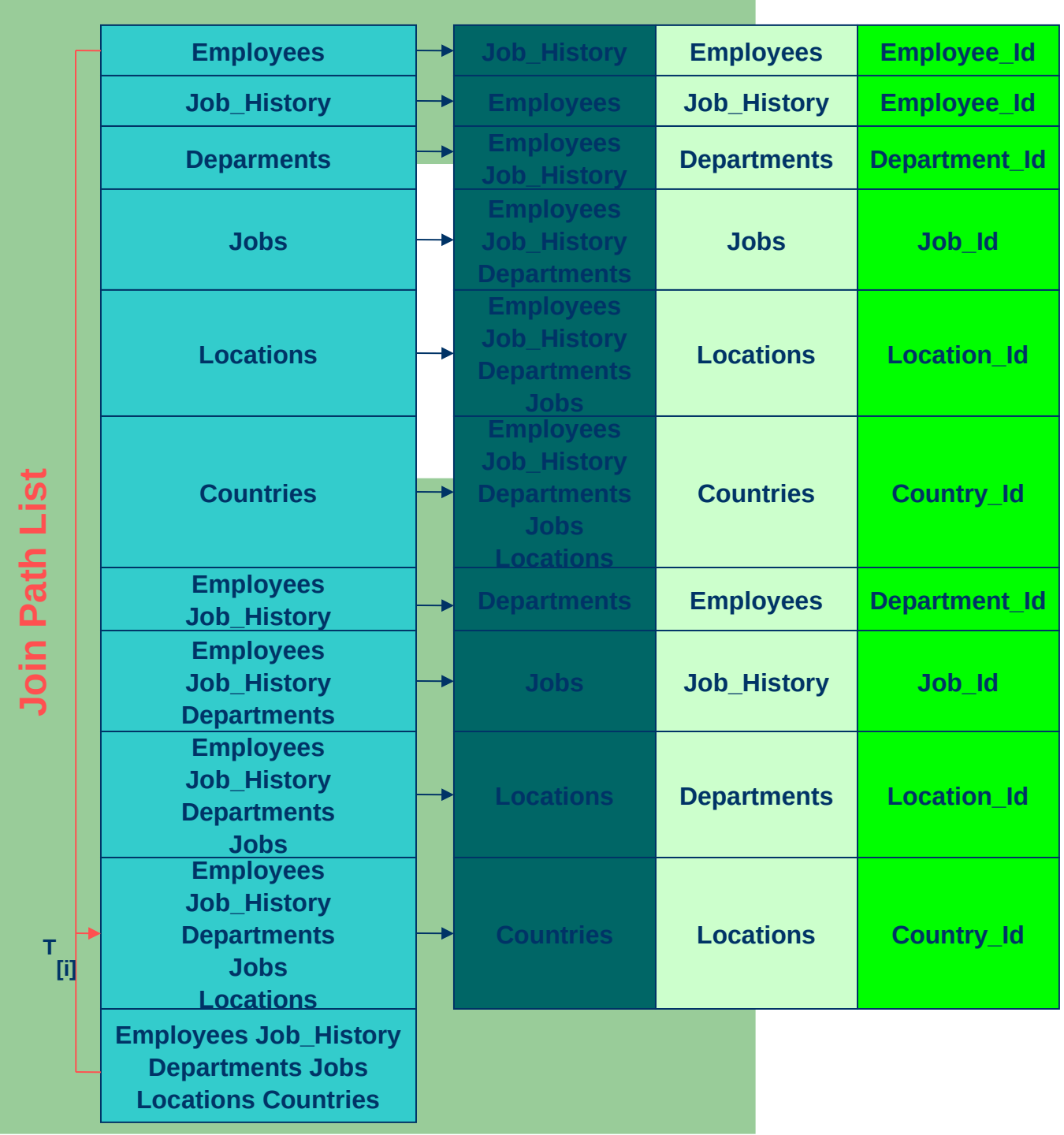
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then
InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)


Join Path List
T
[i]
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees Job_History Departments Jobs Locations Countries
Job_History
Employees
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees
Job_History
Departments
Jobs
Locations
Countries
Employee_Id
Employee_Id
Department_Id
Job_Id
Location_Id
Country_Id
Department_Id
Job_Id
Location_Id
Country_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

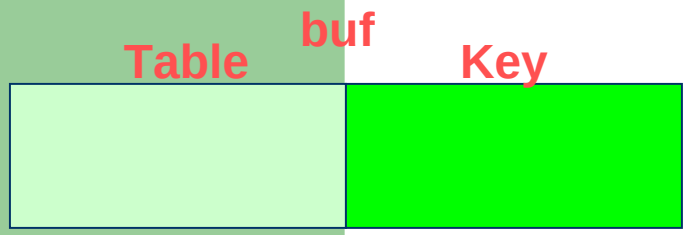
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

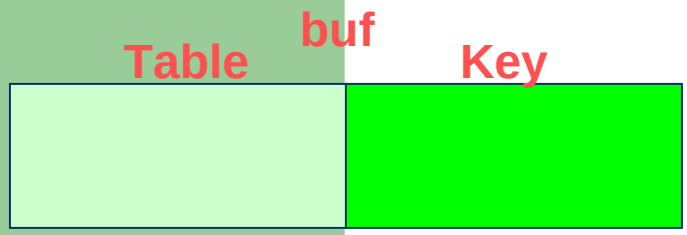
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

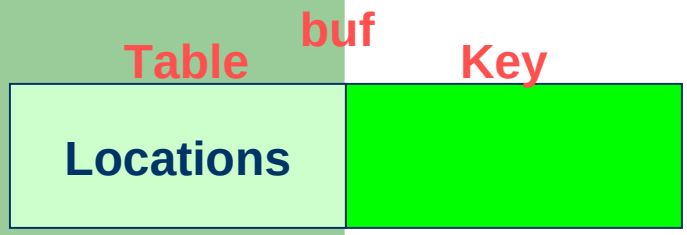
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

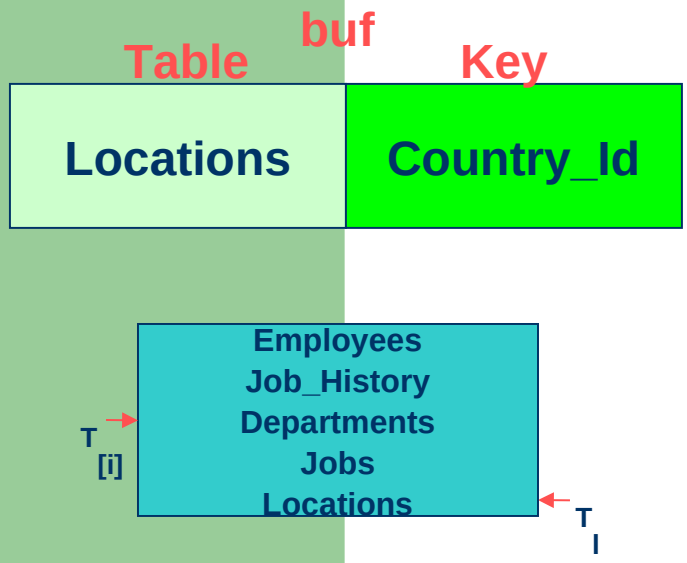
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

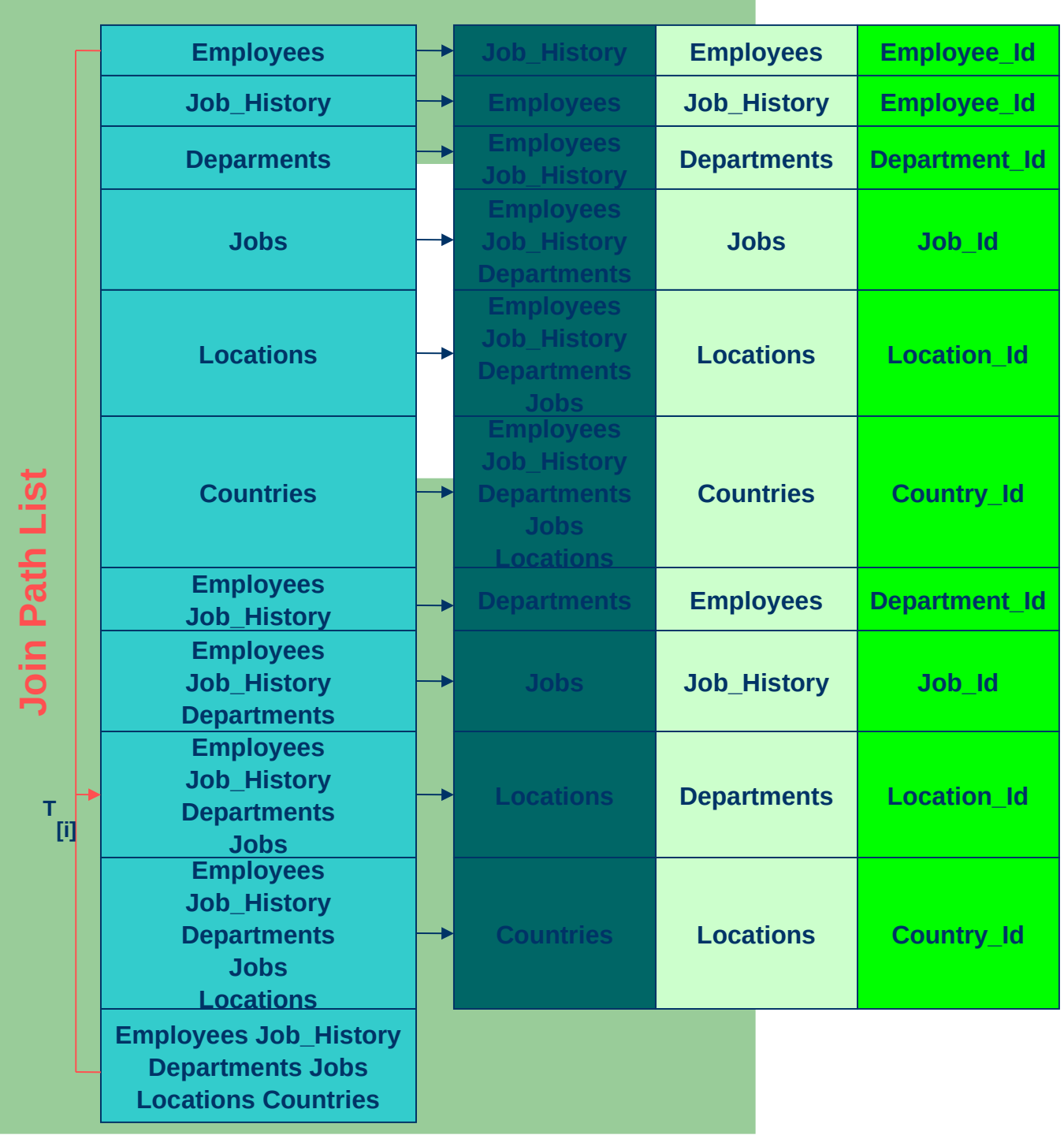
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then
InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)


Join Path List
T
[i]
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees Job_History Departments Jobs Locations Countries
Job_History
Employees
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees
Job_History
Departments
Jobs
Locations
Countries
Employees
Job_History
Departments
Jobs
Locations
Employees
Job_History
Departments
Jobs
Locations
Employee_Id
Employee_Id
Department_Id
Job_Id
Location_Id
Country_Id
Department_Id
Job_Id
Location_Id
Country_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

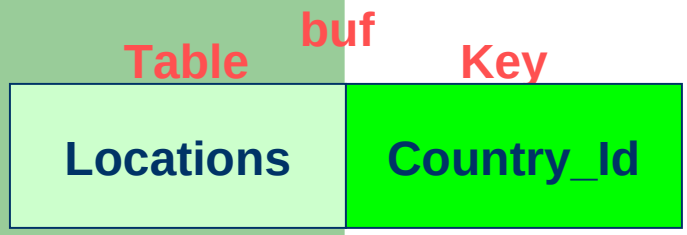
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

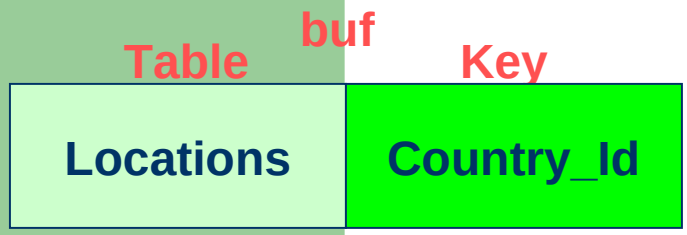
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

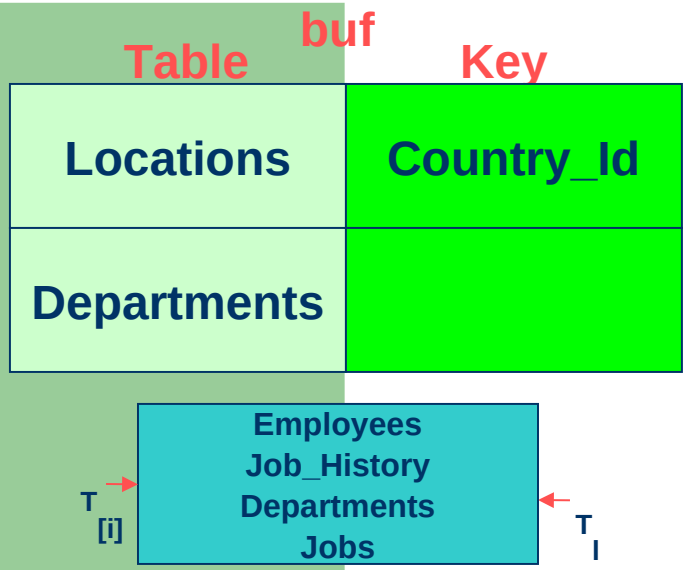
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

→ if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

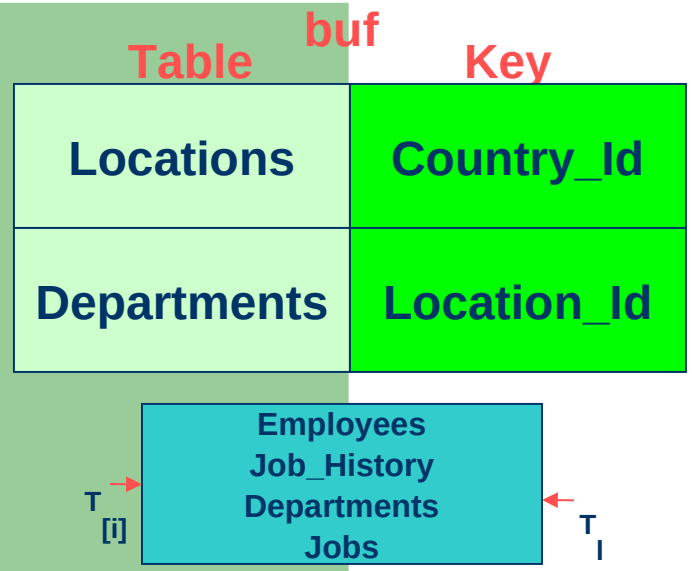
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

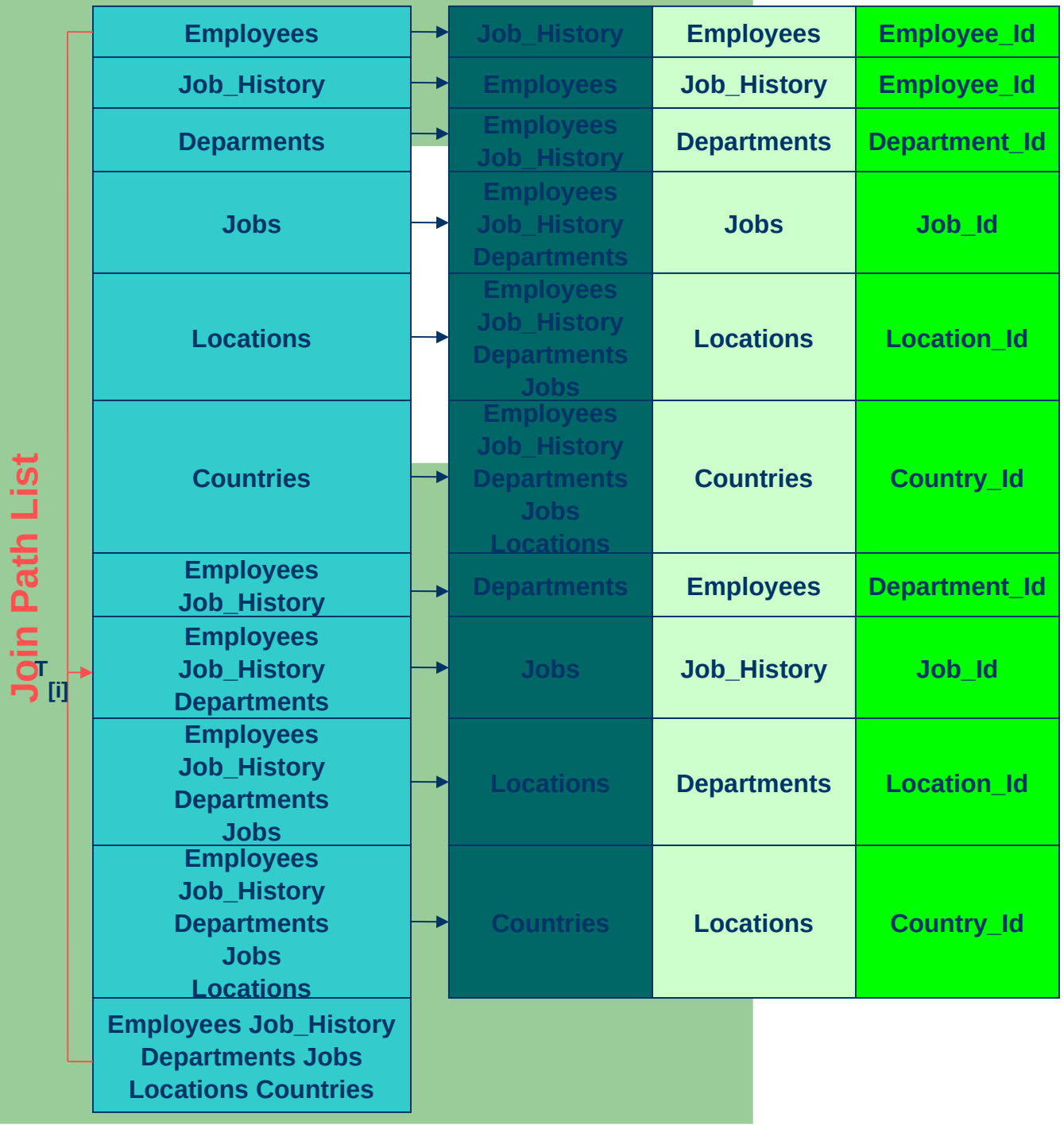
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then
InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)


Join Path List
T
[i]
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees Job_History Departments Jobs Locations Countries
Job_History
Employees
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees Job_History Departments Jobs Locations Countries
Employees
Job_History
Departments
Jobs
Locations
Countries
Employee_Id
Employee_Id
Department_Id
Job_Id
Location_Id
Country_Id
Department_Id
Job_Id
Location_Id
Country_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

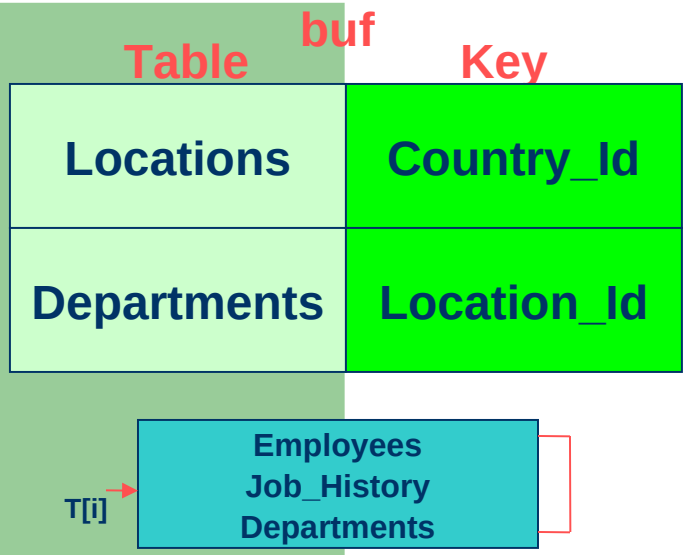
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

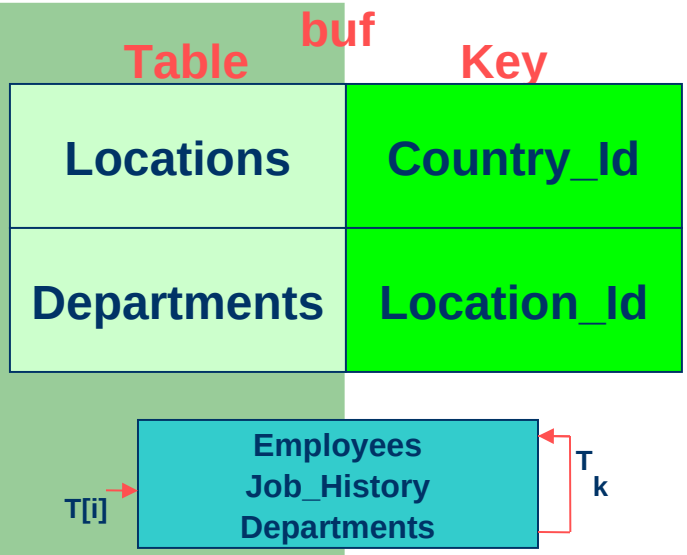
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

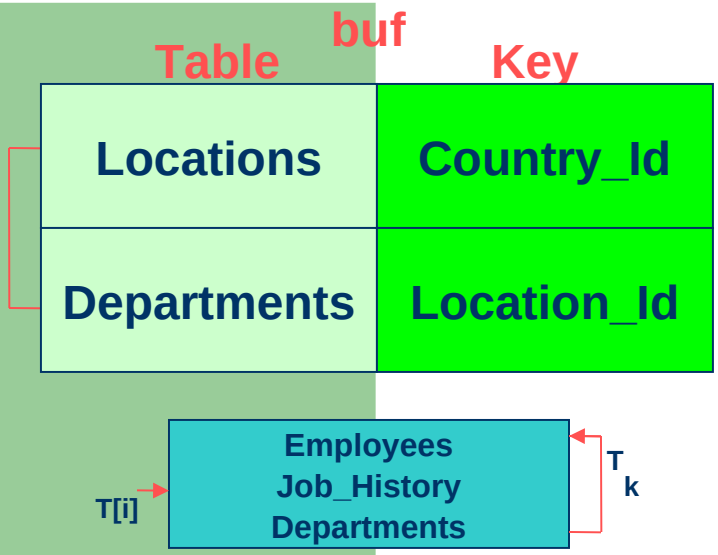
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

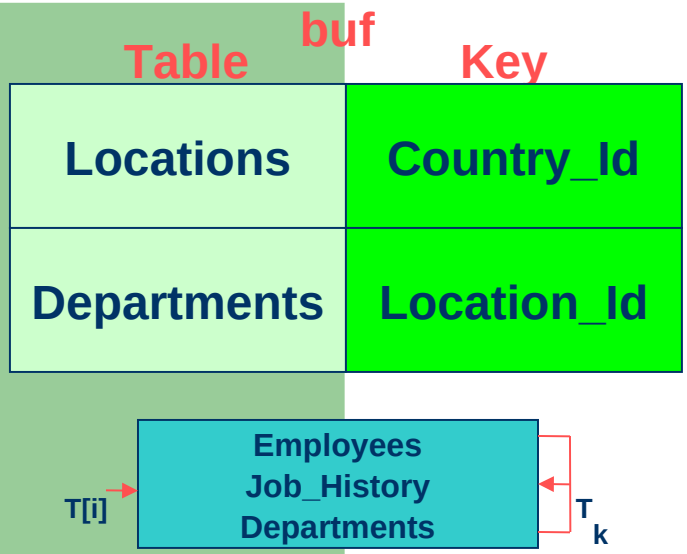
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

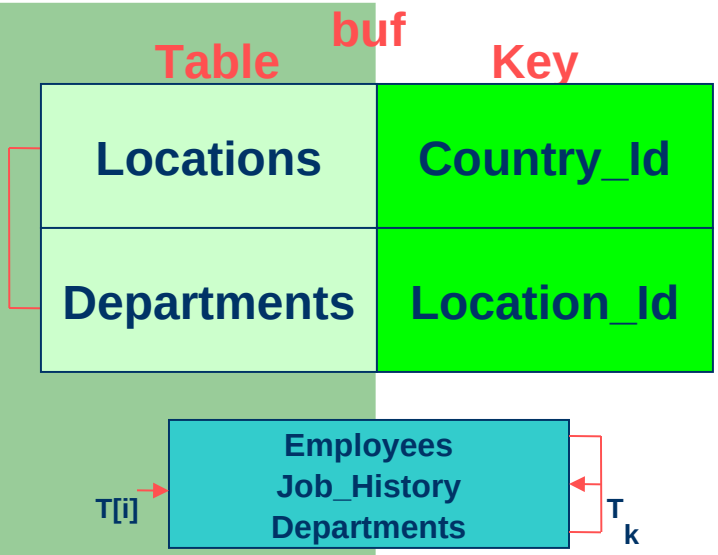
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

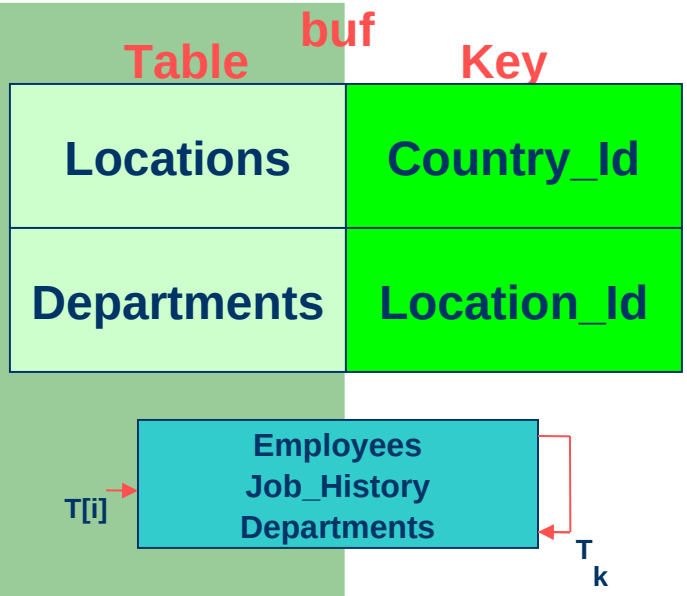
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

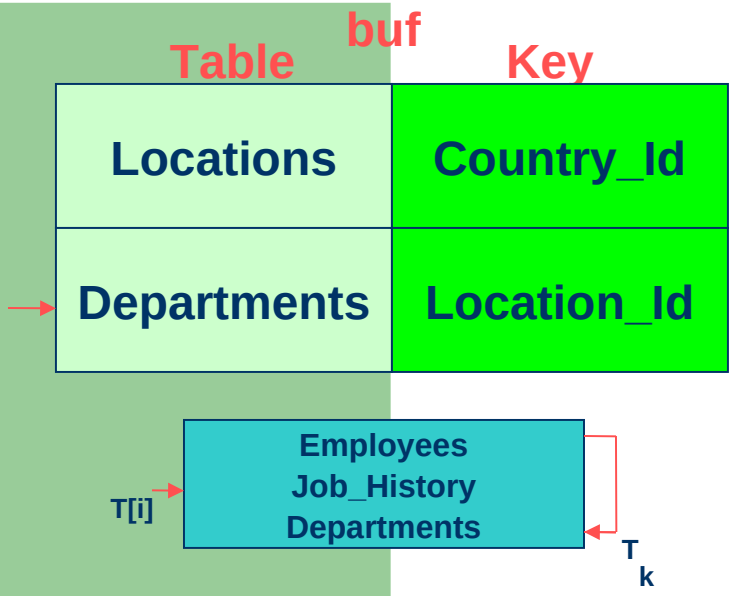
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

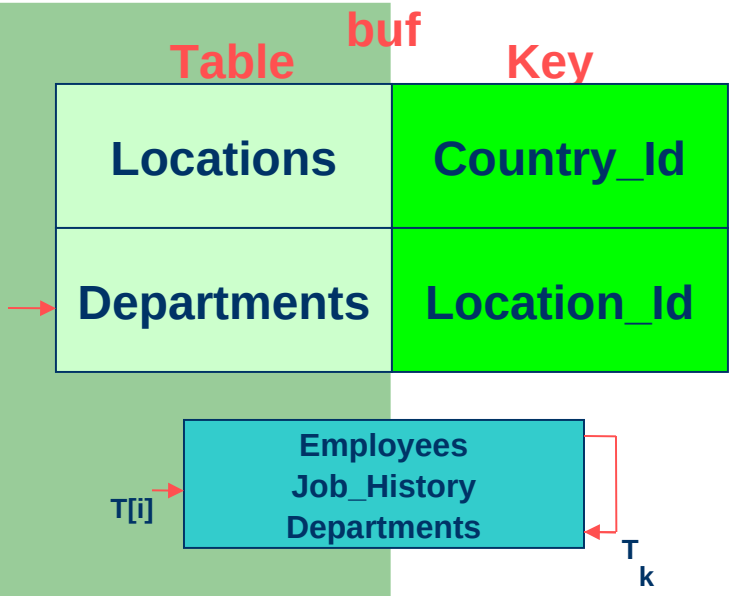
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then
InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)


Join Path List
T[i]
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees Job_History Departments Jobs Locations Countries
Job_History
Employees
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees
Job_History
Departments
Jobs
Locations
Countries
Employee_Id
Employee_Id
Department_Id
Job_Id
Location_Id
Country_Id
Department_Id
Job_Id
Location_Id
Country_Id
Departments
Location_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

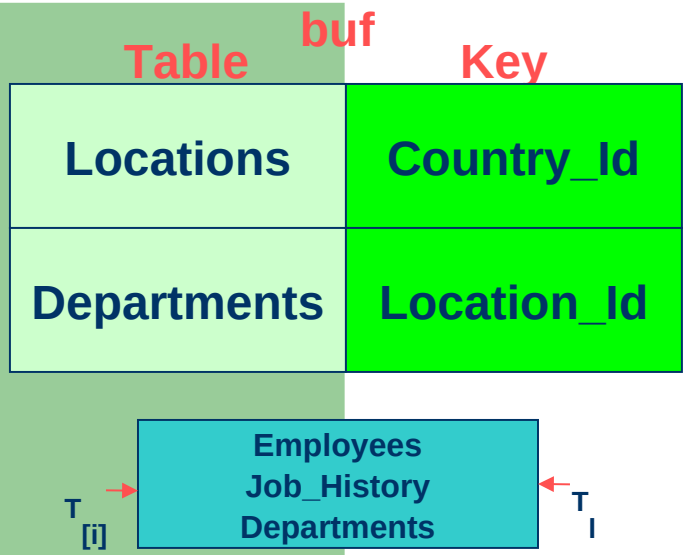
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

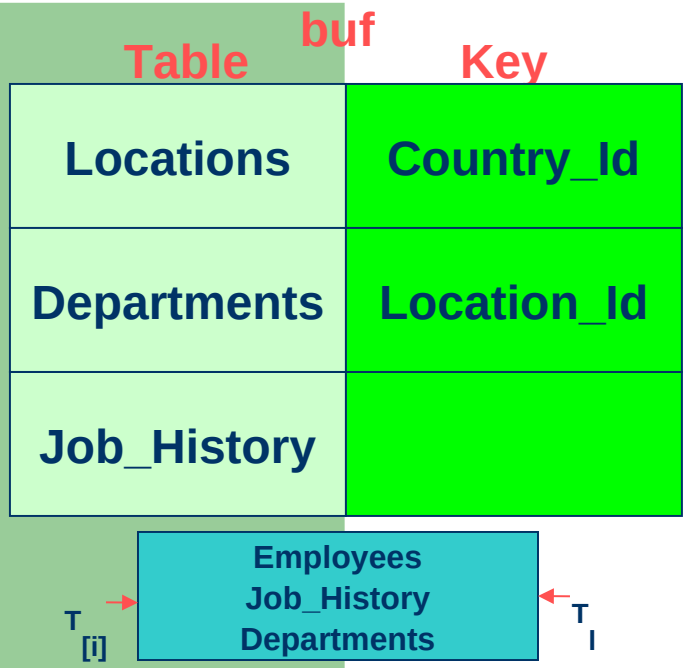
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

→ if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

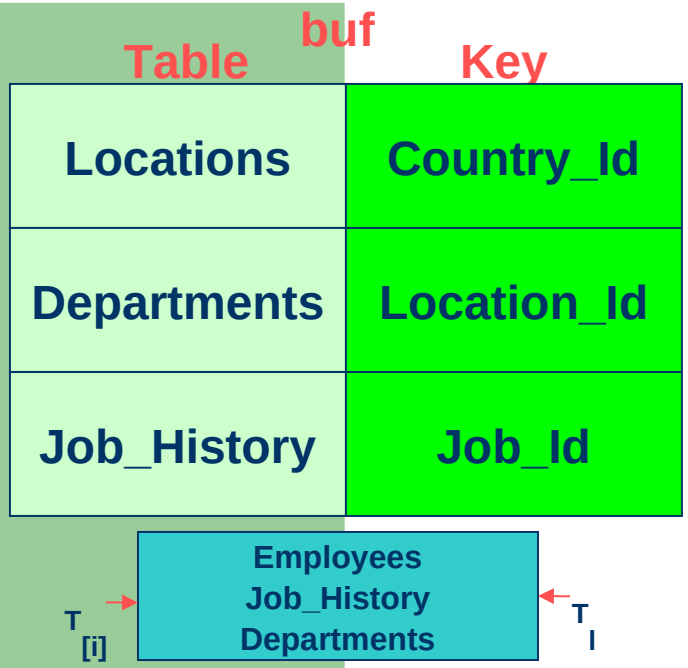
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

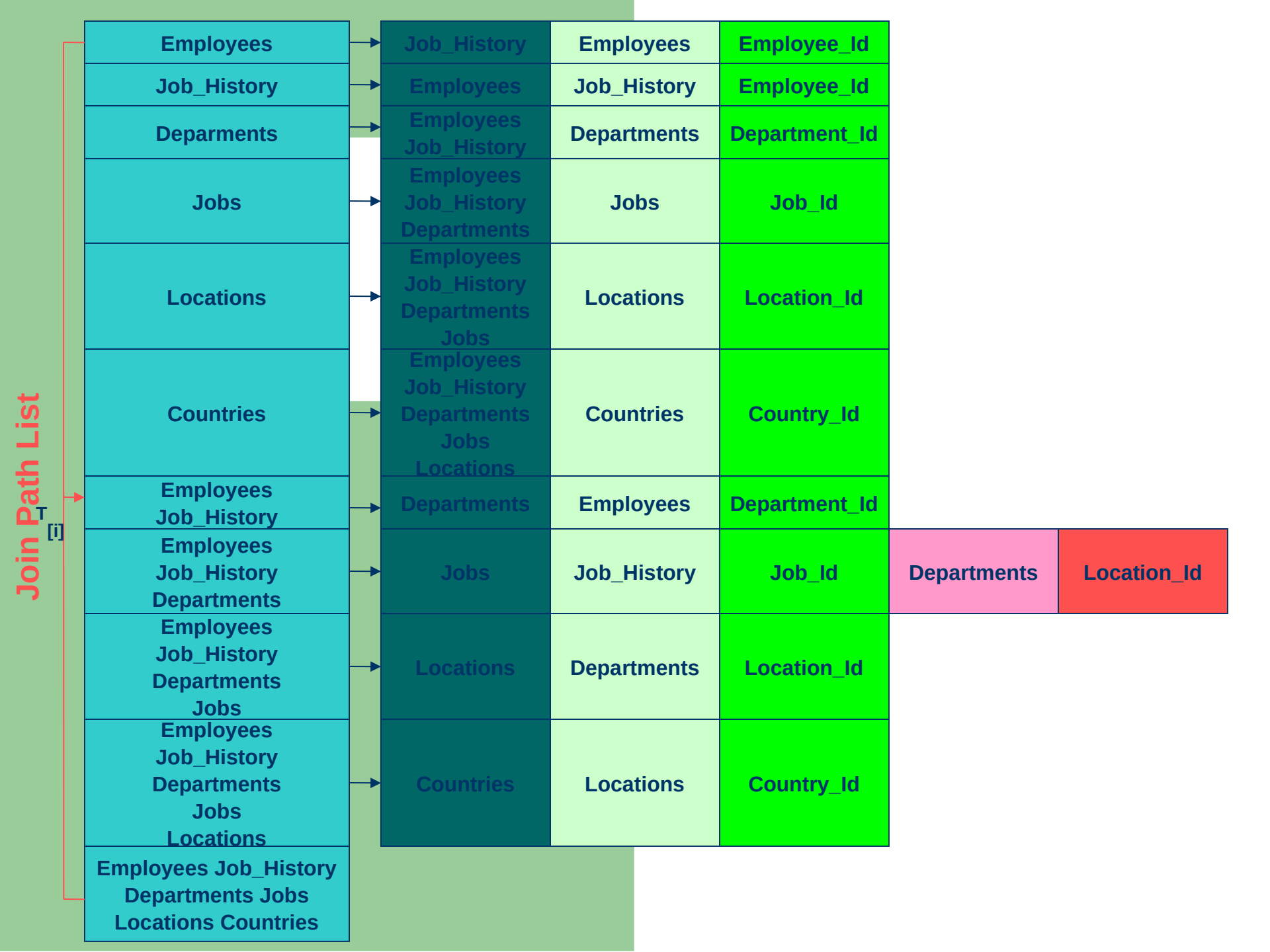
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then
InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)


Join Path List
T[i]
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees Job_History Departments Jobs Locations Countries
Job_History
Employees
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees
Job_History
Departments
Jobs
Locations
Countries
Employee_Id
Employee_Id
Department_Id
Job_Id
Location_Id
Country_Id
Department_Id
Job_Id
Location_Id
Country_Id
Departments
Location_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

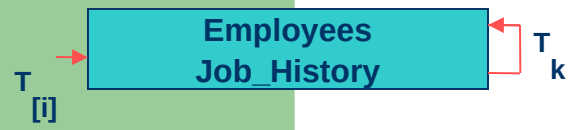
if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

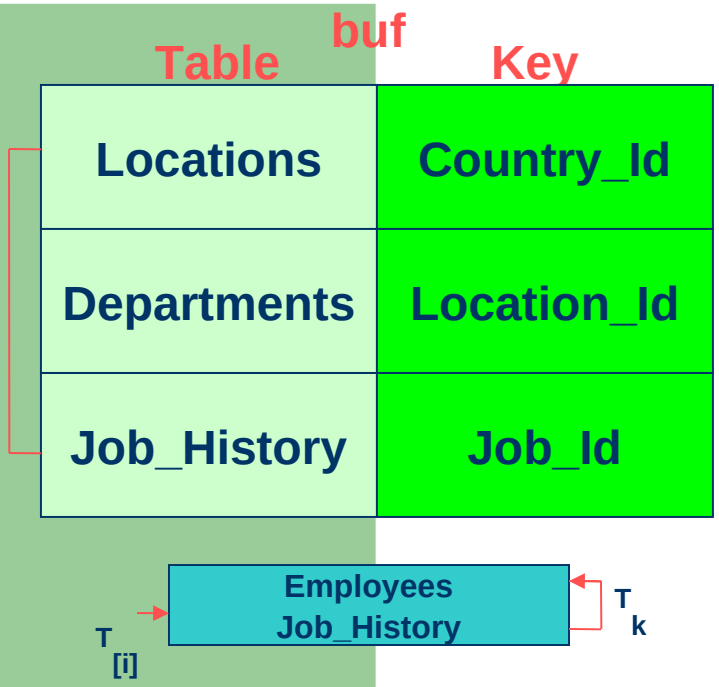
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

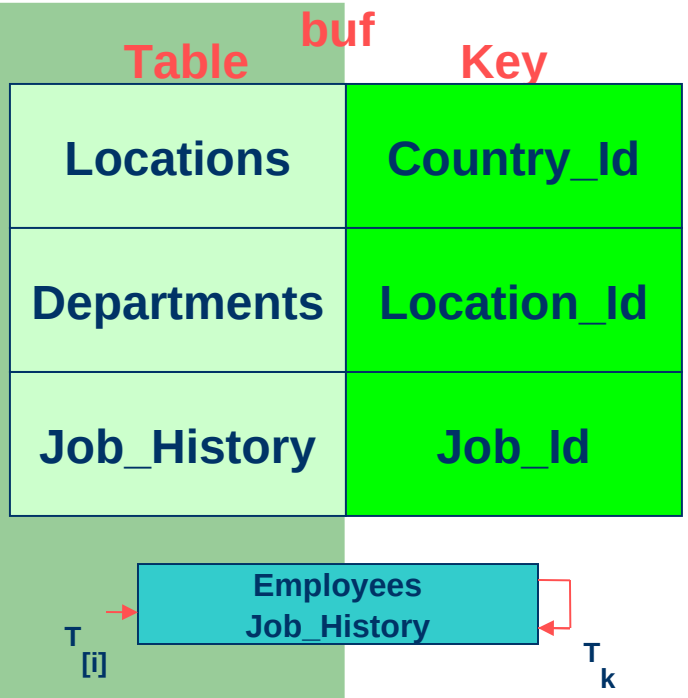
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

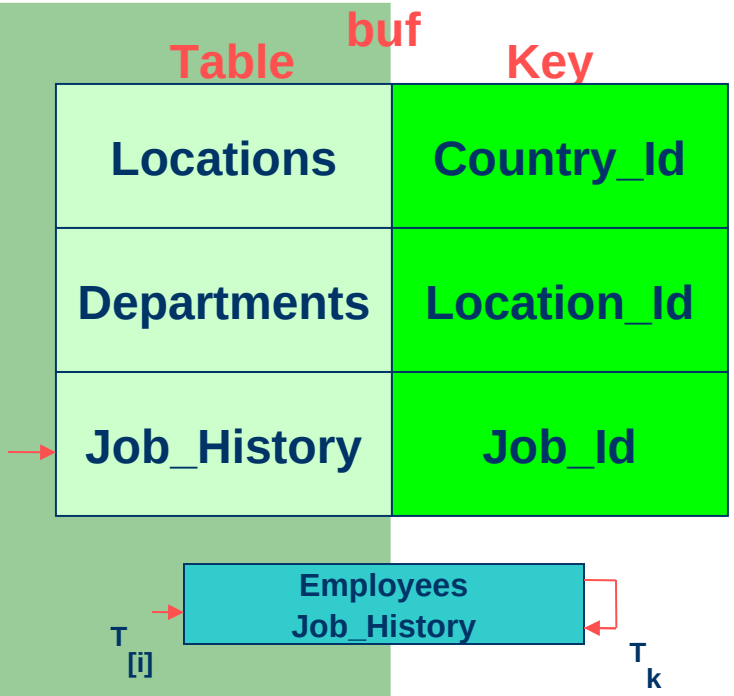
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

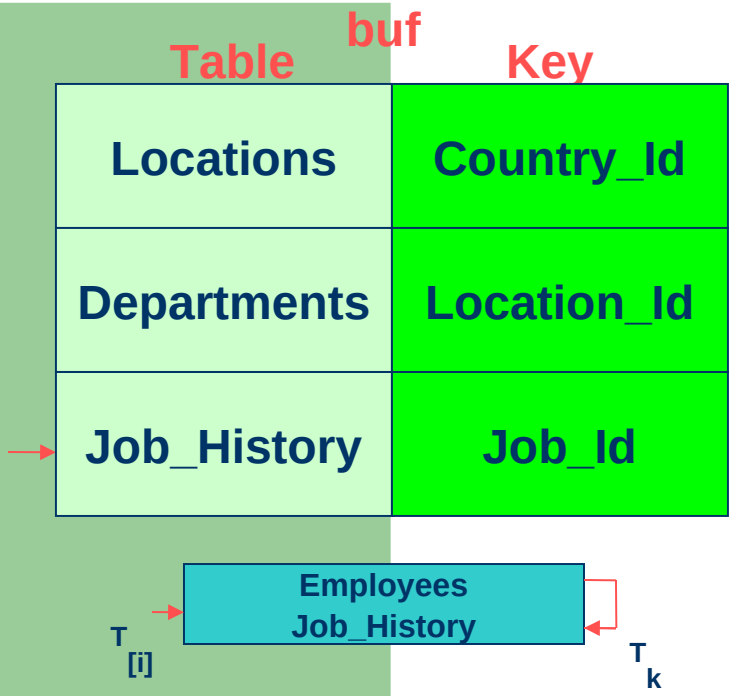
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

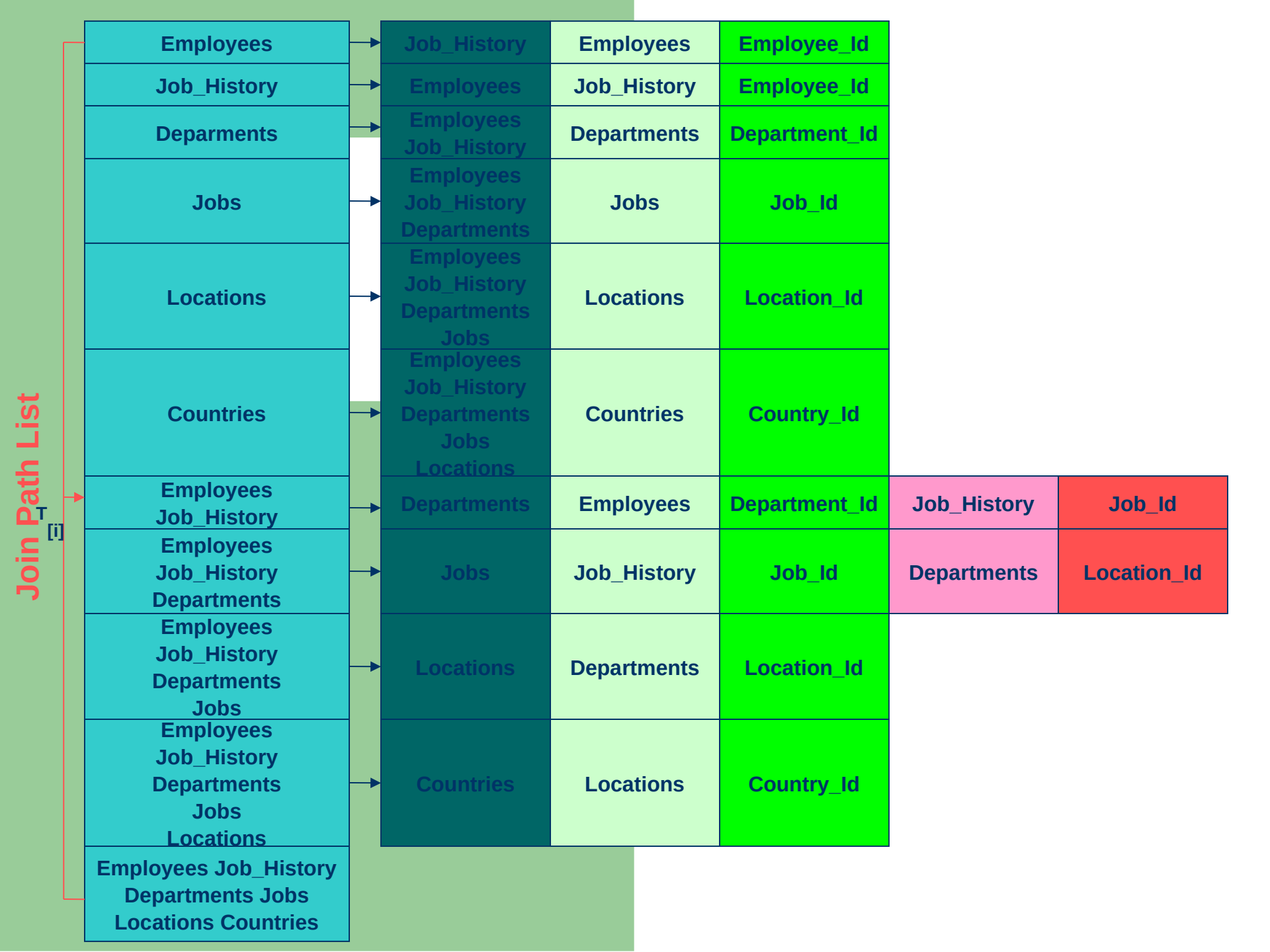
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then
InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)


Join Path List
T[i]
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees Job_History Departments Jobs Locations Countries
Job_History
Employees
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees
Job_History
Departments
Jobs
Locations
Countries
Employee_Id
Employee_Id
Department_Id
Job_Id
Location_Id
Country_Id
Job_History
Job_Id
Departments
Location_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

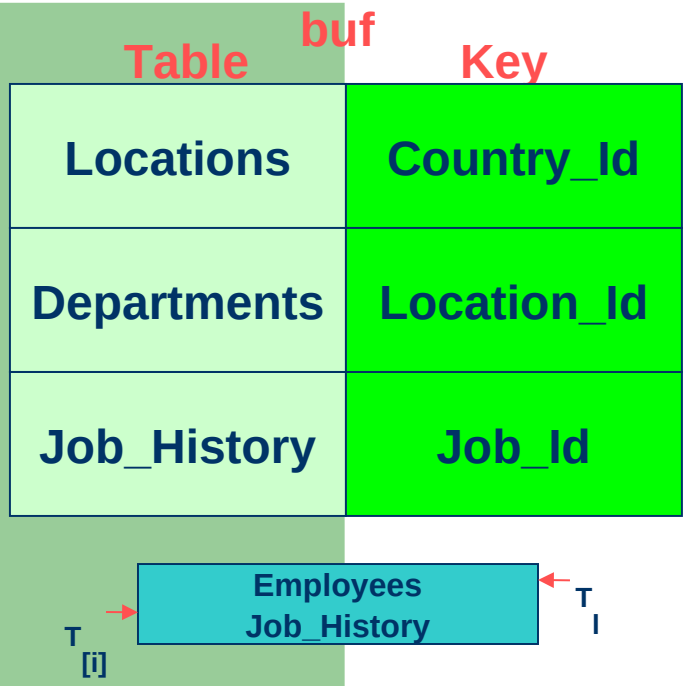
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

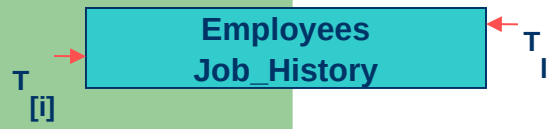
→ if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

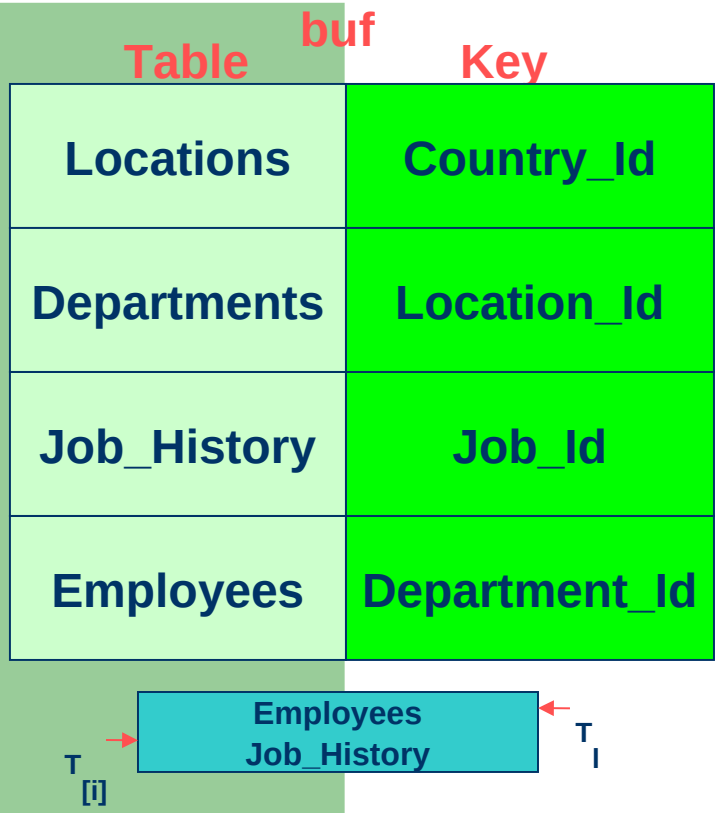
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then
InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)


Join Path List
T[i]
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Job_History
Employees
Departments
Jobs
Locations
Countries
Employees
Job_History
Departments
Jobs
Locations
Employee_Id
Employee_Id
Department_Id
Job_Id
Location_Id
Country_Id
Job_History
Job_Id
Departments
Location_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

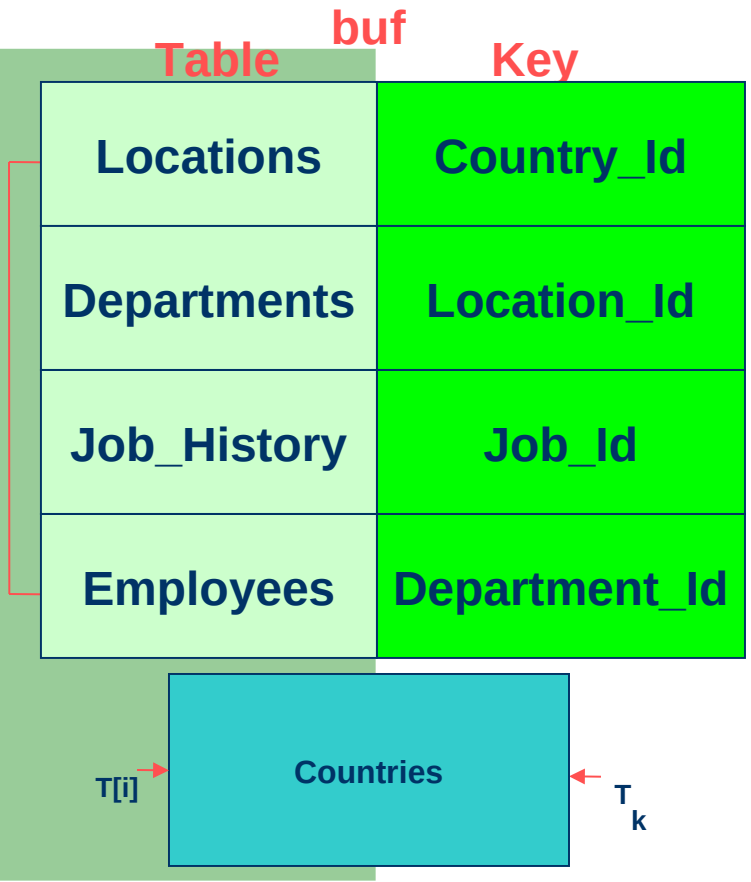
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then

InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

T[i] → Countries

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then
InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)


Join Path List
T [i]
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees Job_History Departments Jobs Locations Countries
Job_History
Employees
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees
Job_History
Departments
Jobs
Locations
Countries
Employee_Id
Employee_Id
Department_Id
Job_Id
Location_Id
Country_Id
Job_Id
Location_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

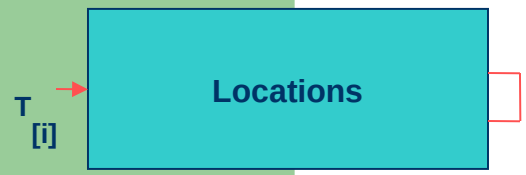
if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

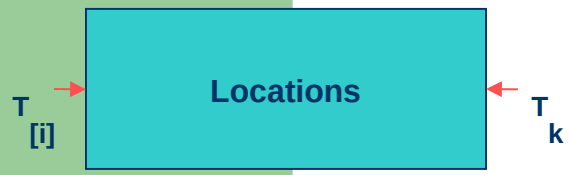
if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

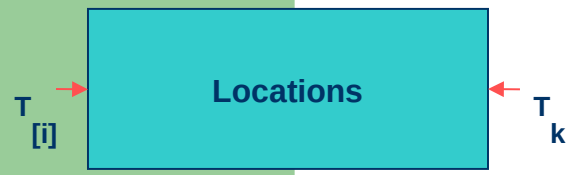
if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

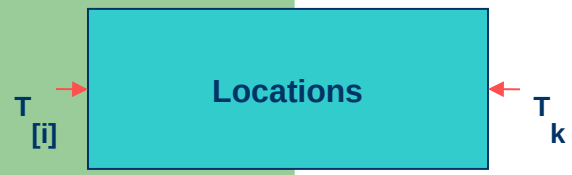
if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

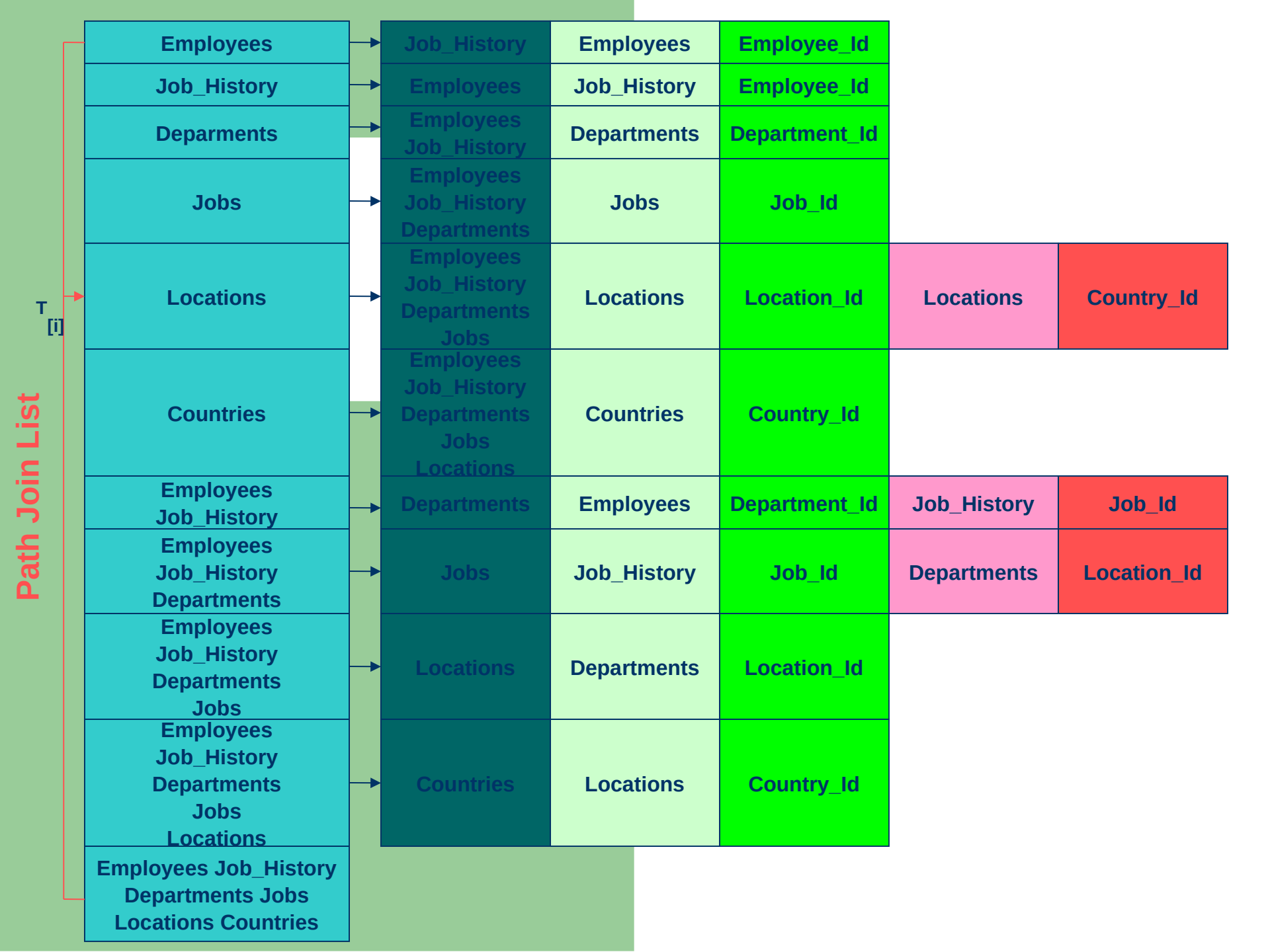
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then
InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)


Path Join List
T [i]
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Job_History
Employees
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees
Job_History
Departments
Jobs
Locations
Countries
Employee_Id
Employee_Id
Department_Id
Job_Id
Location_Id
Country_Id
Locations
Country_Id
Job_History
Job_Id
Departments
Location_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

$T_{[i]}$ → Locations

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

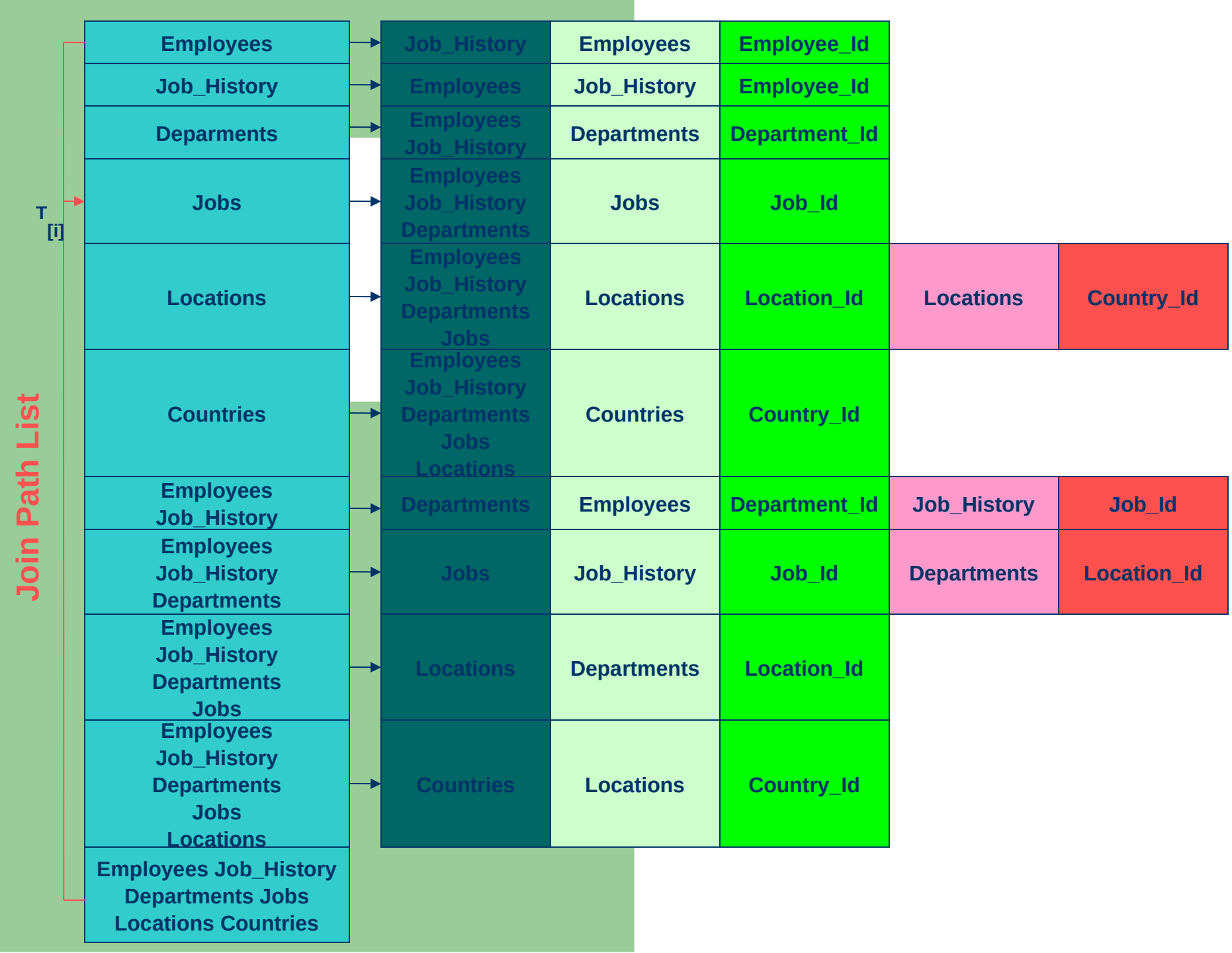
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then
InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)


Join Path List
T
[i]
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees Job_History Departments Jobs Locations Countries
Job_History
Employees
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees
Job_History
Departments
Jobs
Locations
Countries
Employee_Id
Employee_Id
Department_Id
Job_Id
Location_Id
Country_Id
Locations
Country_Id
Job_History
Job_Id
Departments
Location_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

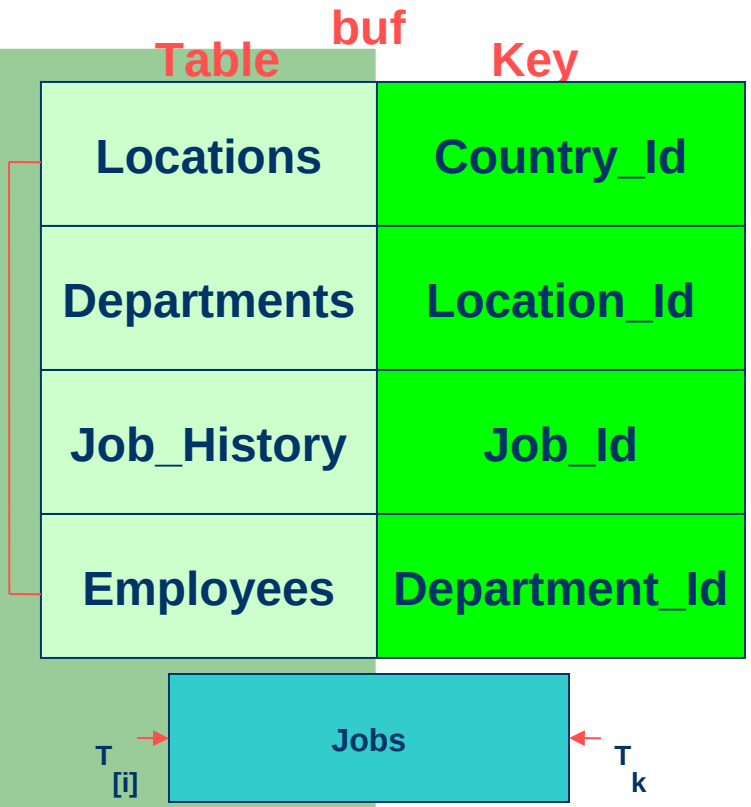
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)



create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

$T_{[i]}$ → Jobs

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

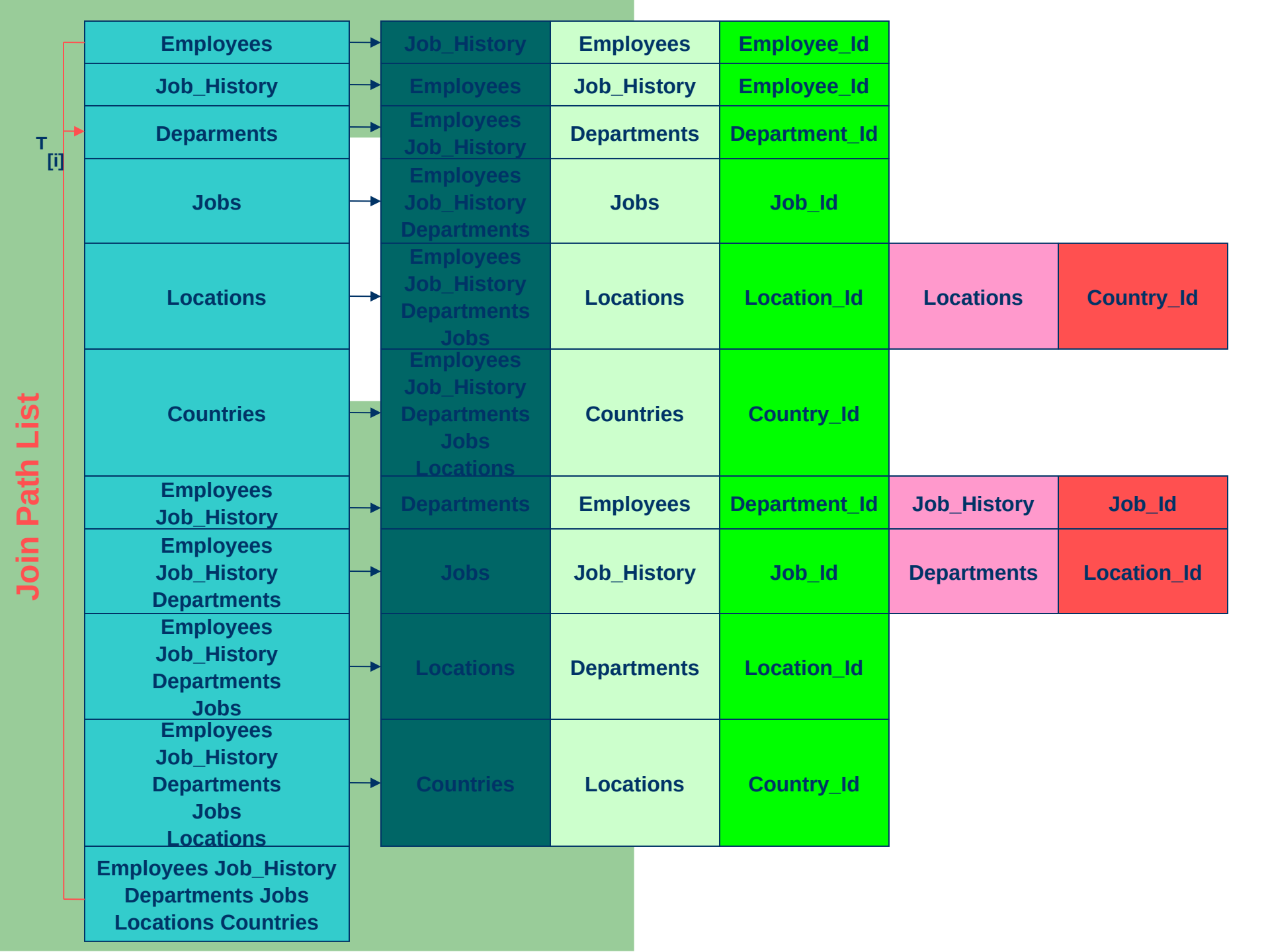
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then
InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)


Join Path List
T
[i]
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees Job_History Departments Jobs Locations Countries
Job_History
Employees
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees
Job_History
Departments
Jobs
Locations
Countries
Employee_Id
Employee_Id
Department_Id
Job_Id
Location_Id
Country_Id
Locations
Country_Id
Job_History
Job_Id
Departments
Location_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

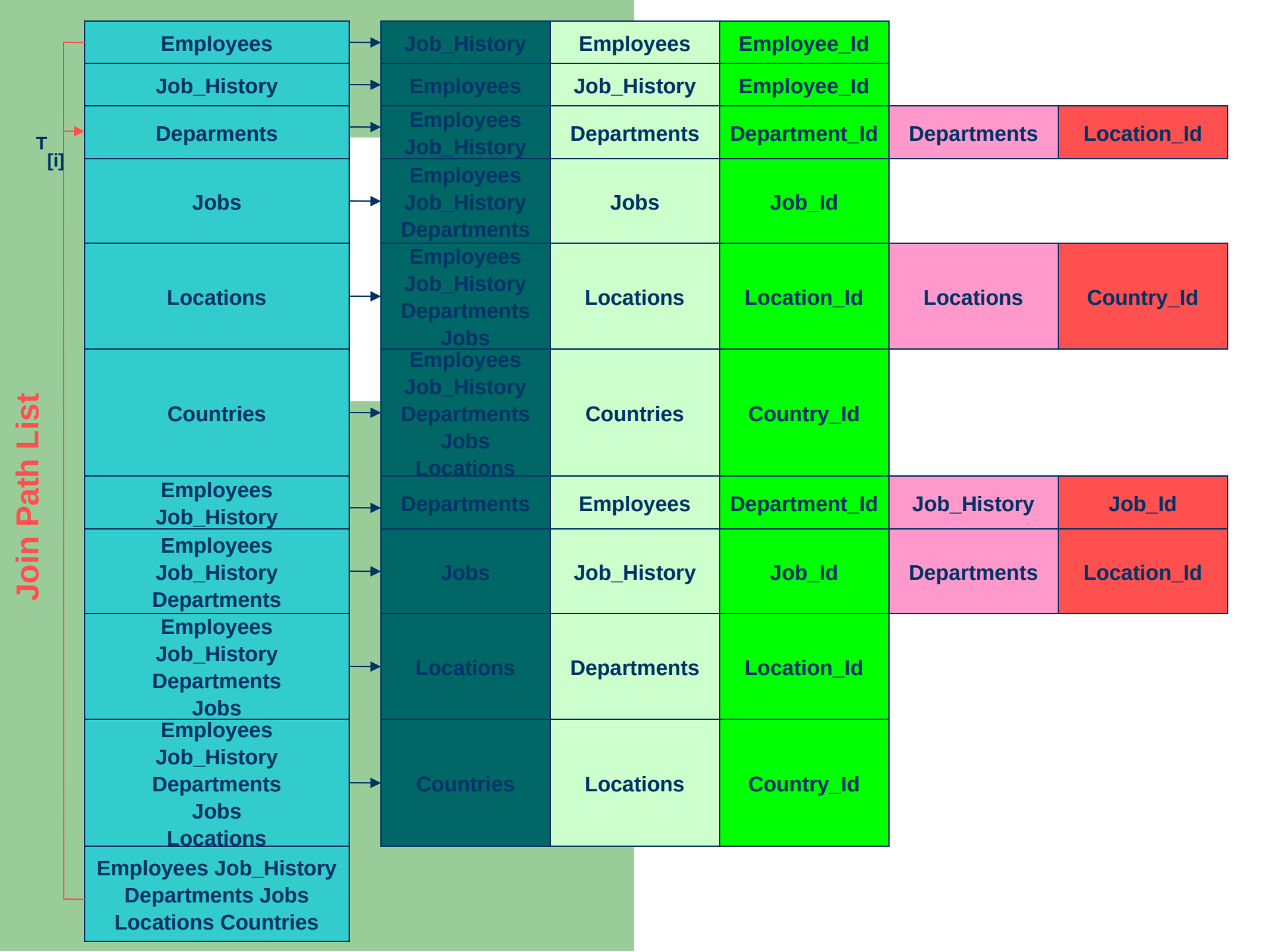
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then
InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)


Join Path List
T [i]
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees Job_History Departments Jobs Locations Countries
Job_History
Employees
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees Job_History Departments Jobs Locations Countries
Employees
Job_History
Departments
Jobs
Locations
Countries
Employee_Id
Employee_Id
Department_Id
Job_Id
Location_Id
Country_Id
Departments
Location_Id
Locations
Country_Id
Job_History
Job_Id
Departments
Location_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

$T_{[i]}$ → Deparments

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

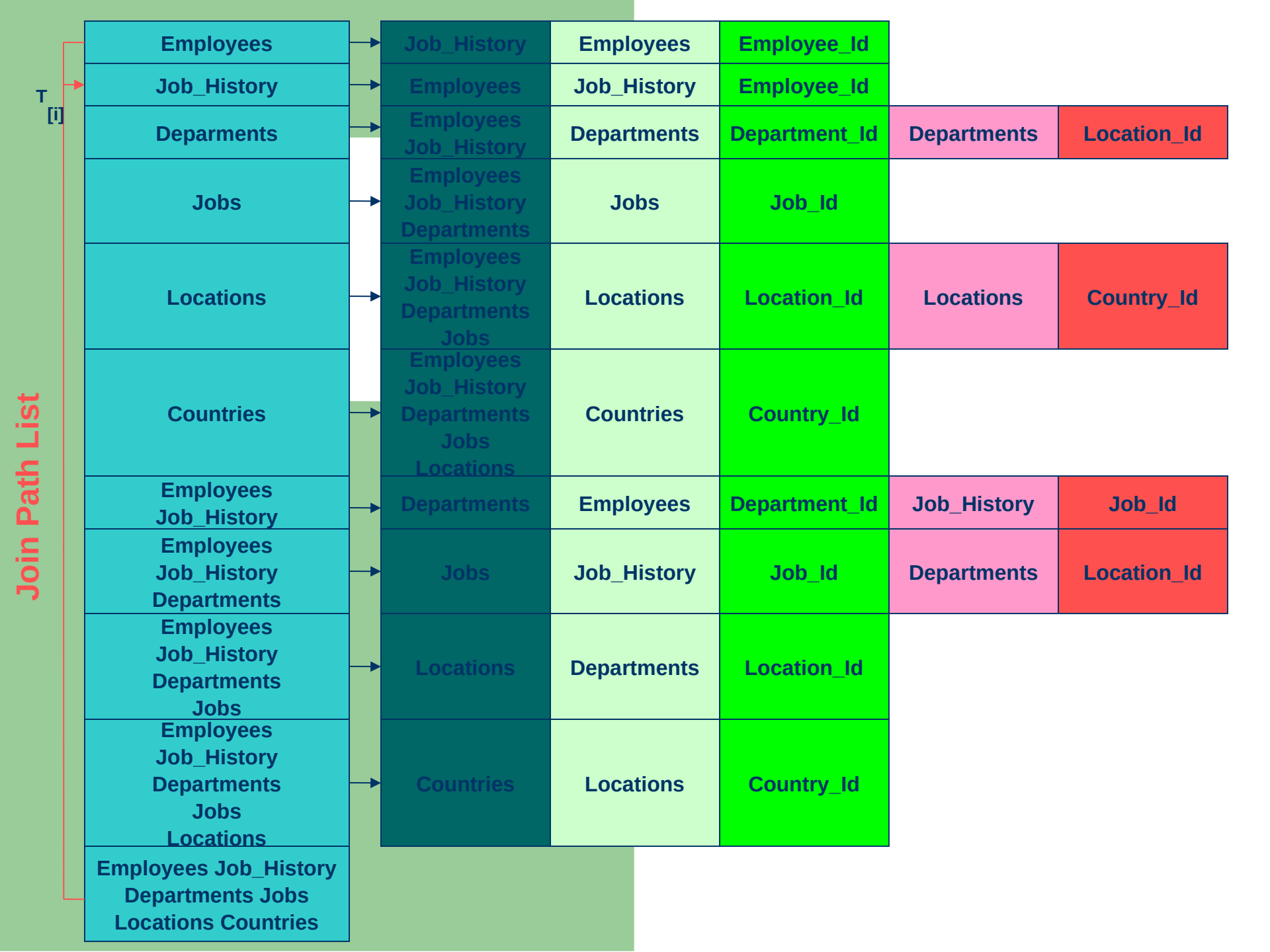
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then
InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)


Join Path List
T
[i]
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees Job_History Departments Jobs Locations Countries
Job_History
Employees
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees
Job_History
Departments
Jobs
Locations
Countries
Employee_Id
Employee_Id
Department_Id
Job_Id
Location_Id
Country_Id
Departments
Location_Id
Locations
Country_Id
Job_History
Job_Id
Departments
Location_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

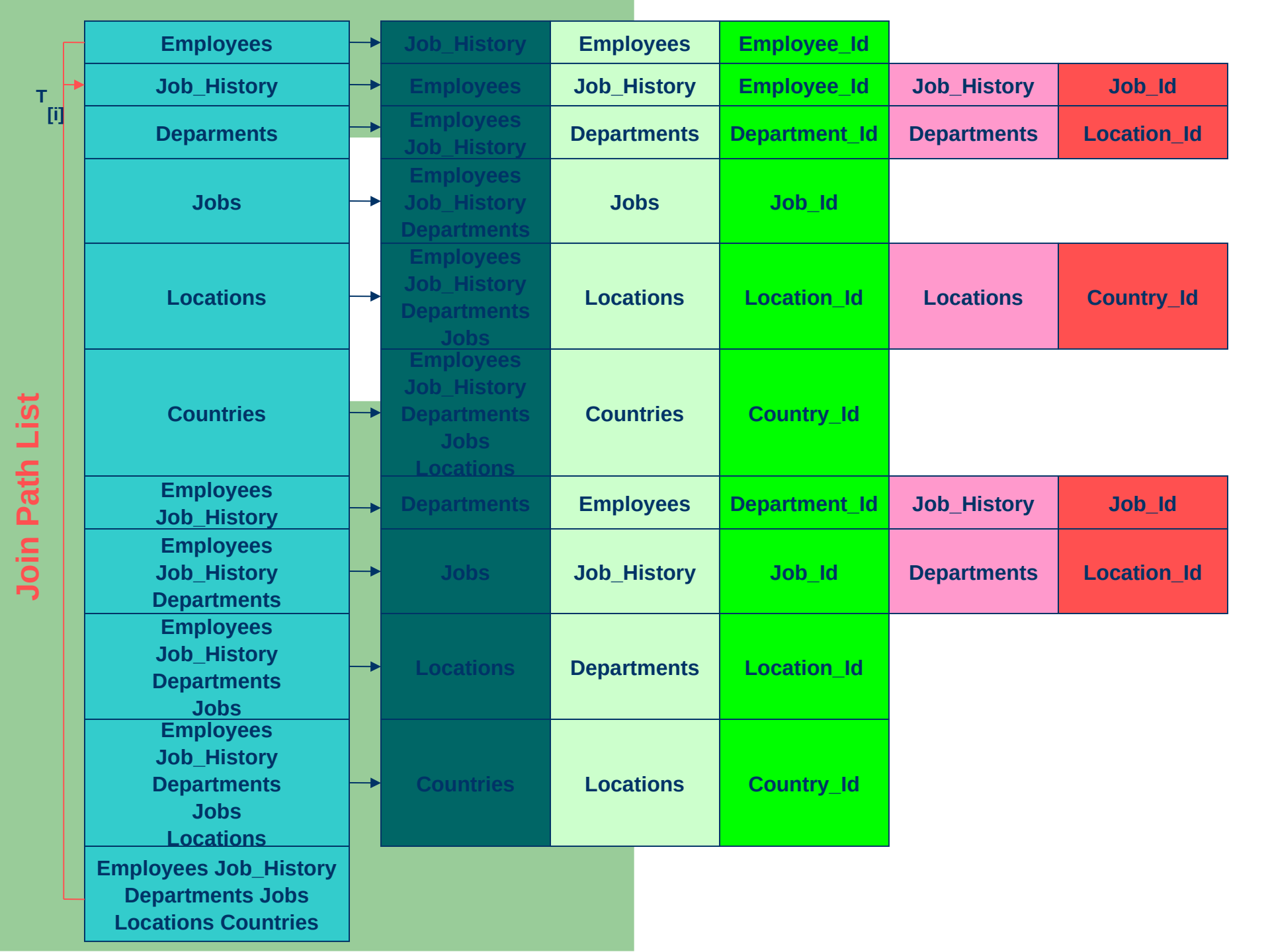
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then
InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)


Join Path List
T
[i]
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Job_History
Employees
Departments
Jobs
Locations
Countries
Employees
Job_History
Departments
Job_History
Job_Id
Departments
Location_Id
Jobs
Job_Id
Locations
Country_Id
Countries
Country_Id
Employees
Job_History
Departments
Jobs
Locations
Employees Job_History Departments Jobs Locations Countries

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

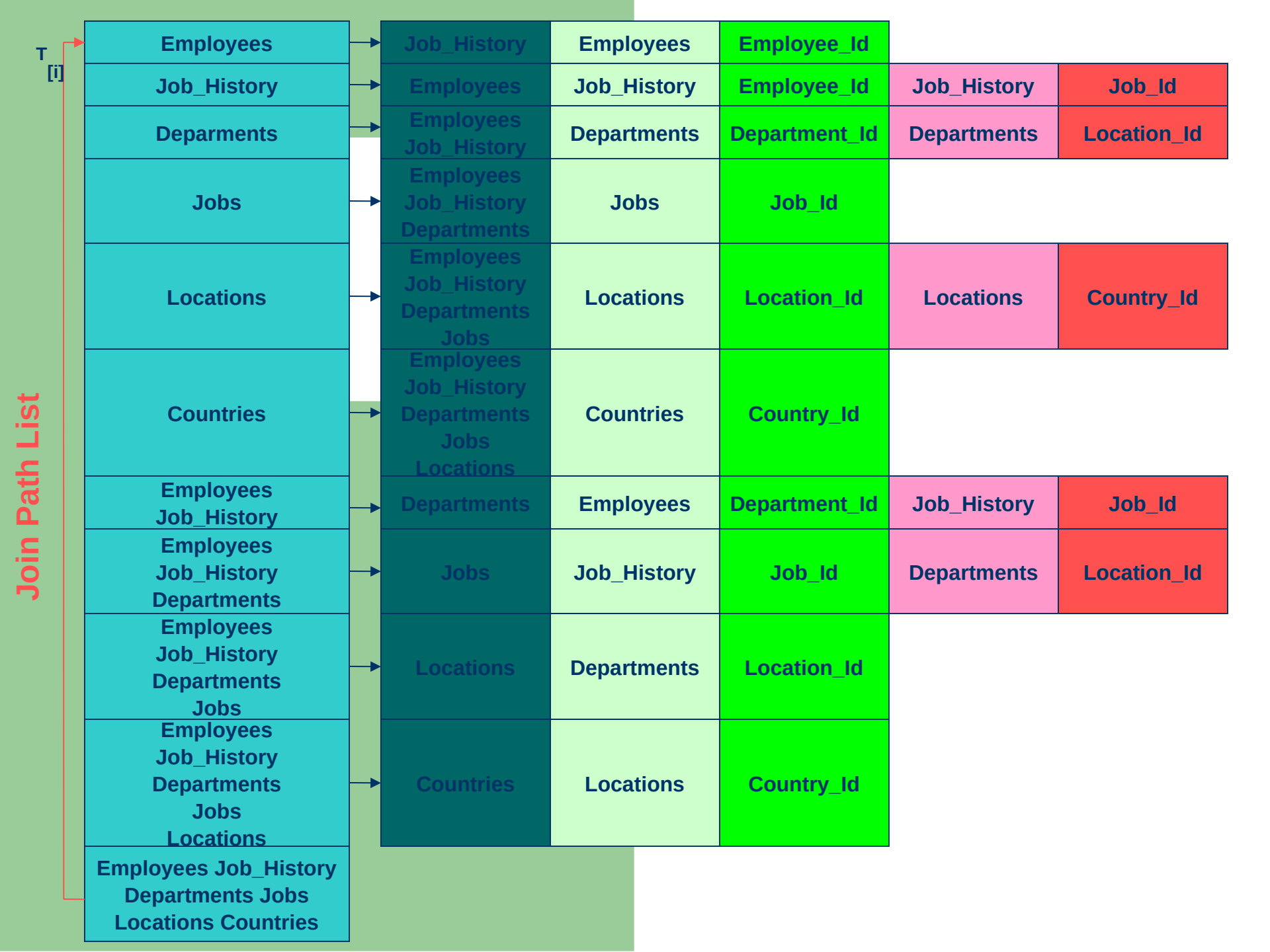
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then
InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)


T
[i]
Join Path List
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Job_History
Employees
Departments
Jobs
Locations
Countries
Employees
Job_History
Departments
Jobs
Locations
Employee_Id
Job_History
Department_Id
Job_Id
Location_Id
Country_Id
Job_History
Job_Id
Departments
Location_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

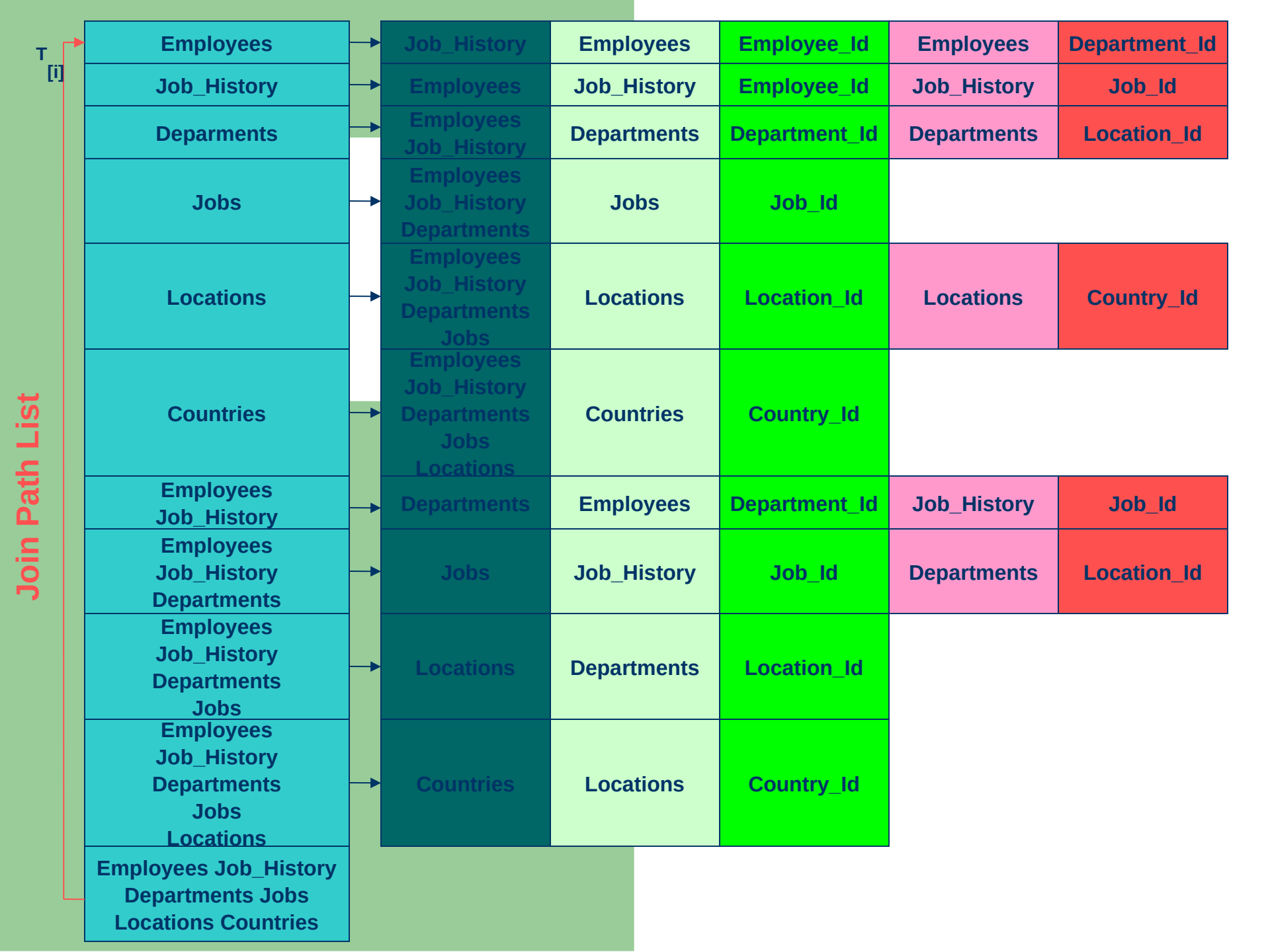
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then
InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)


T
[i]
Join Path List
Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments
Employees Job_History Departments Jobs
Employees Job_History Departments Jobs Locations
Employees Job_History Departments Jobs Locations Countries
Job_History
Employees
Departments
Jobs
Locations
Countries
Employees
Job_History
Departments
Jobs
Locations
Employees
Job_History
Departments
Jobs
Locations
Employee_Id
Job_History
Department_Id
Job_Id
Location_Id
Country_Id
Employees
Job_History
Departments
Jobs
Locations
Countries
Department_Id
Job_Id
Location_Id
Country_Id

create a structure buf with 2 fields: Table and Key

for all the tables in JoinPathList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in $T_{[i]}$ do

take one T_k at a time

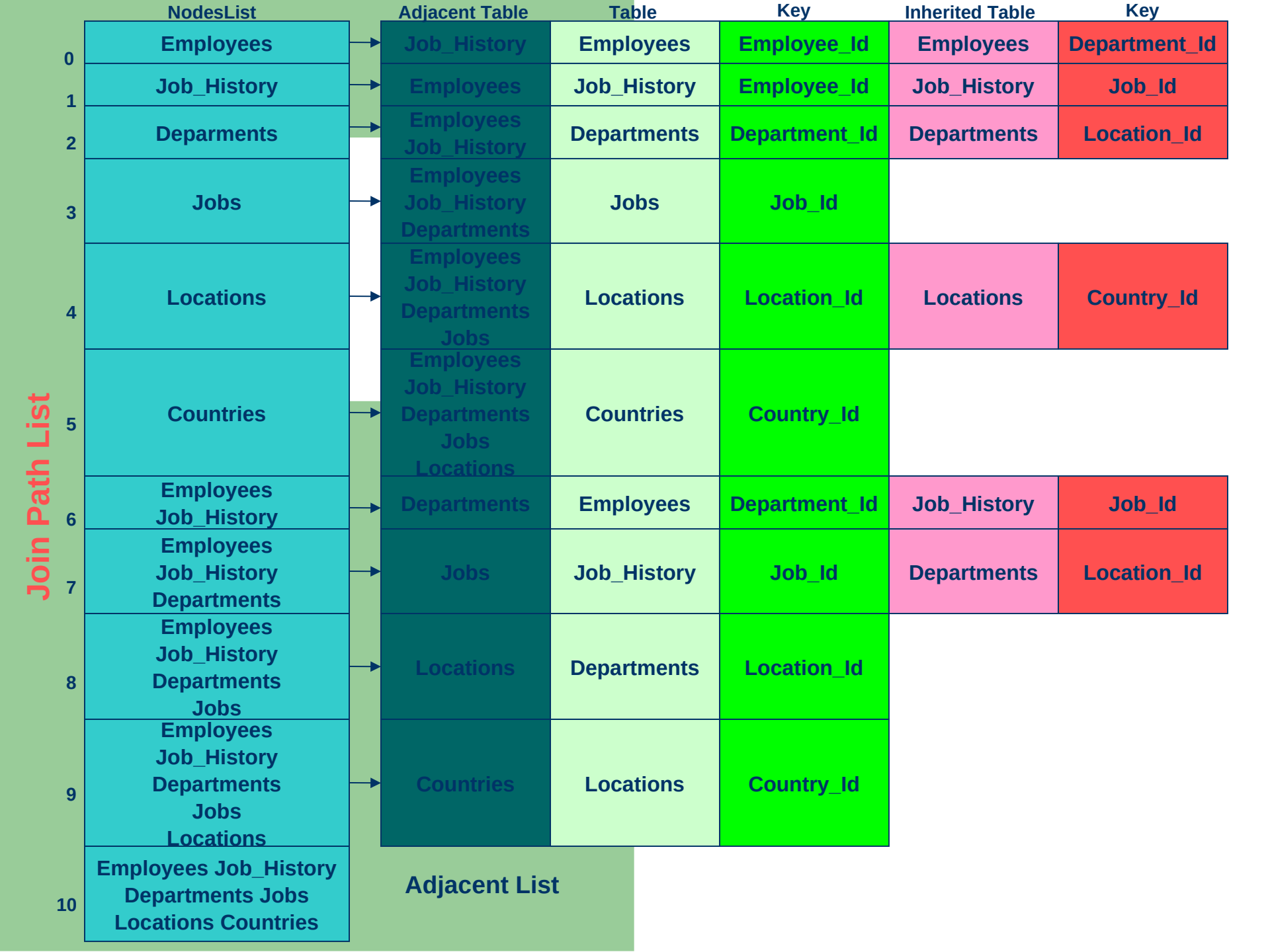
for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in InheritedKey($T_{[i]}$)) then
InheritedKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)


Join Path List
NodesList
Adjacent Table
Table
Key
Inherited Table
Key
0
Employees
Job_History
Employees
Employee_Id
Employees
Department_Id
1
Job_History
Employees
Job_History
Employee_Id
Job_History
Job_Id
2
Deparments
Employees
Job_History
Departments
Department_Id
Departments
Location_Id
3
Jobs
Employees
Job_History
Departments
Jobs
Job_Id
4
Locations
Employees
Job_History
Departments
Jobs
Locations
Location_Id
Locations
Country_Id
5
Countries
Employees
Job_History
Departments
Jobs
Locations
Countries
Country_Id
6
Employees
Job_History
Departments
Employees
Department_Id
Job_History
Job_Id
7
Employees
Job_History
Departments
Jobs
Job_History
Job_Id
Departments
Location_Id
8
Employees
Job_History
Departments
Jobs
Locations
Departments
Location_Id
9
Employees
Job_History
Departments
Jobs
Locations
Countries
Locations
Country_Id
10
Employees Job_History
Departments Jobs
Locations Countries
Adjacent List

Give a general name for the B⋈Tree.

Give for every entry in the JoinPathList a B+Tree index with name as the B⋈Tree + the PathJoinList entry number.

About the last virtual table, it index has no keys, it works because we consider pairs of < keys, Data Pointers > so they are ordered by their data pointers. Scanning the index we get all the sequences of joined data pointers.

Non Terminal has repeated empty keys they point to different pages. When comes a key it would be inserted in the last page.

Duplicate keys are inserted and when a page is full , the key is repeated in the non terminal.

In any case we can incorporate any key of our choice from the tables forming the virtual table.

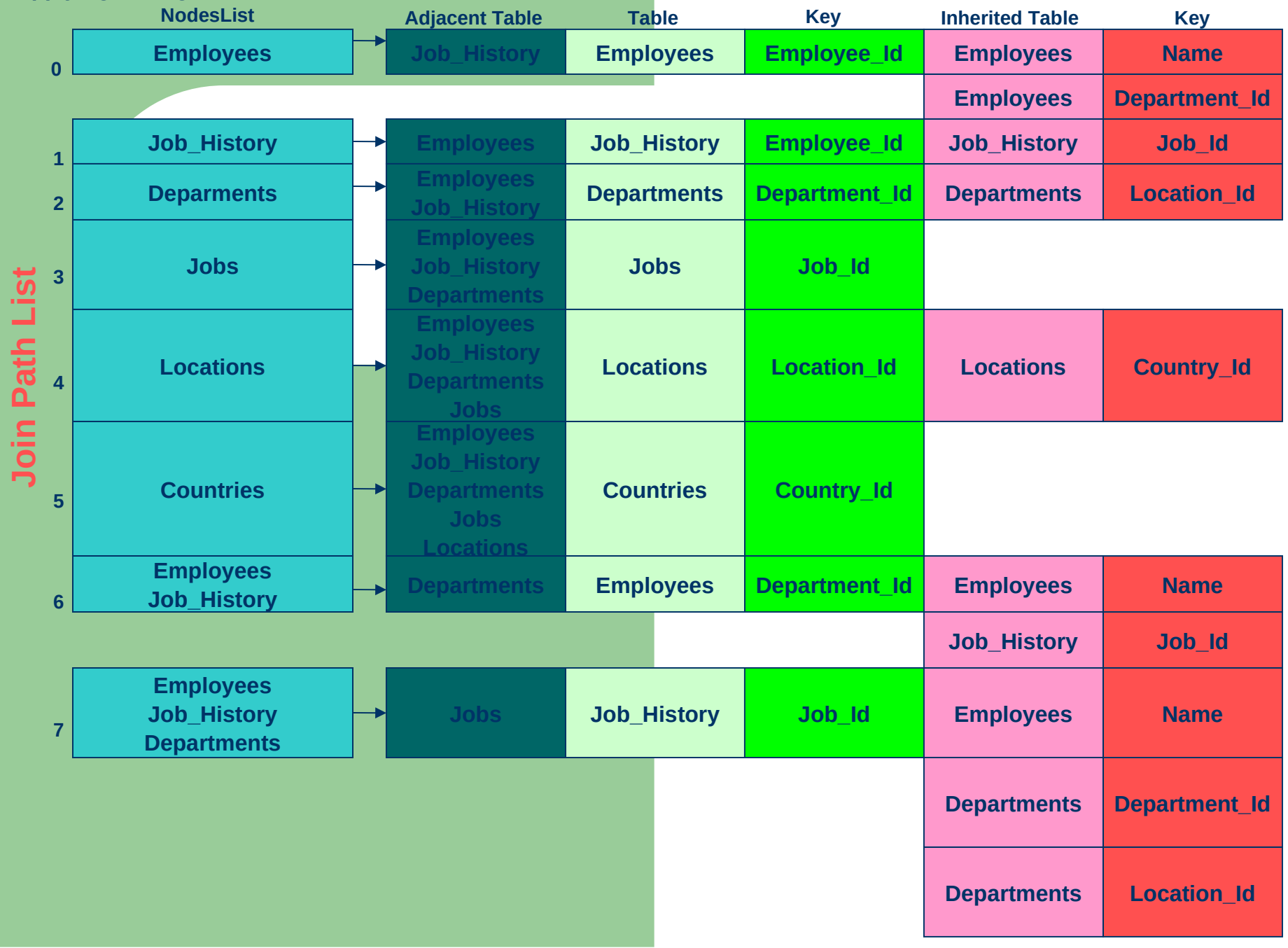
If the table is in join with itself, consider the table twice as aliases.

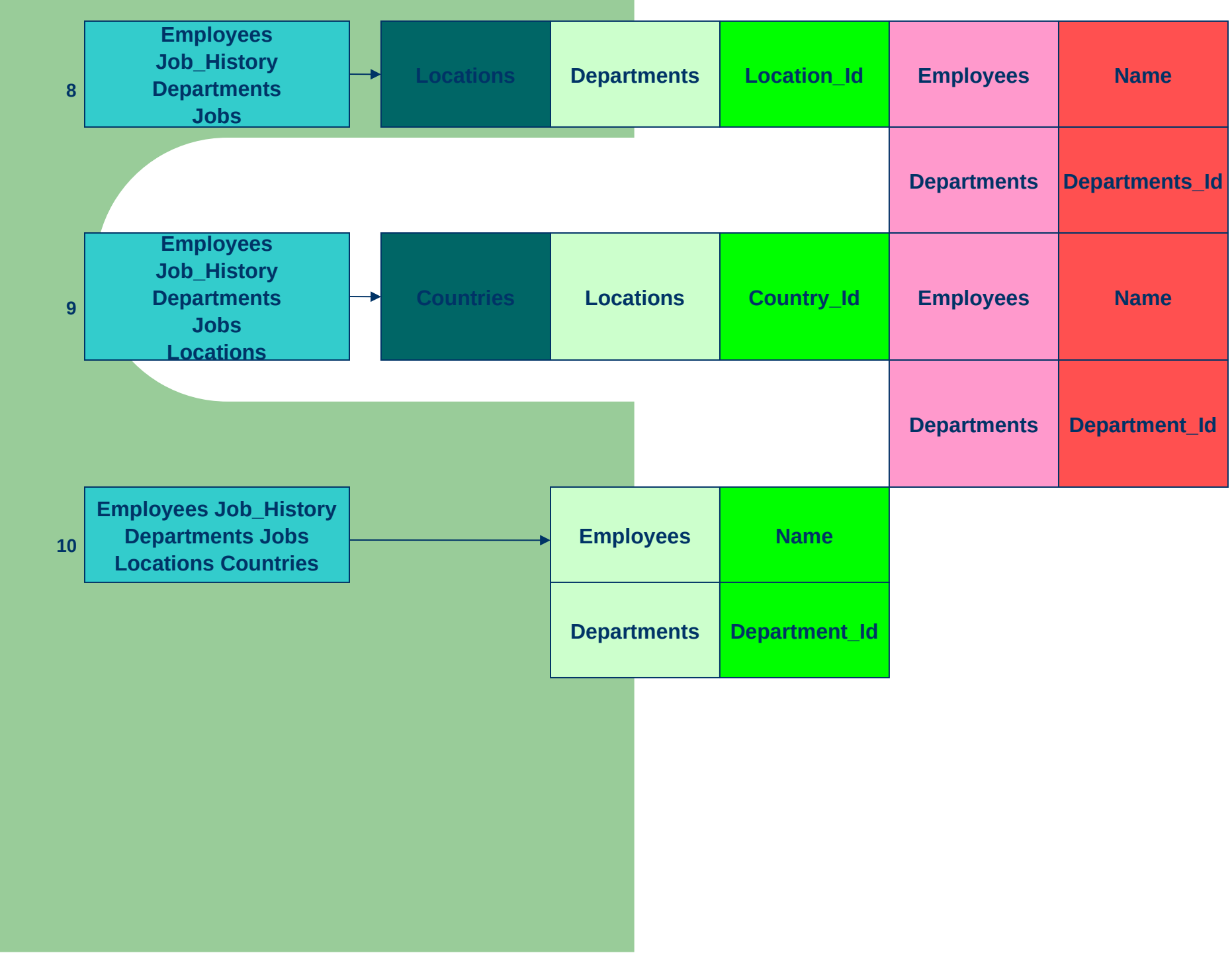
Define a Create Join Index (IndexName, Eventual columns for the last virtual table representing tables in join)

Implementation:

Use a big buffer and from the Data Dictionary divide it by the keys length, inherited keys length and space for the number of Data Pointers.

Suppose we wants the join sequence of the tables ordered by Employees.Name and Departments.Department_Id, applying the algorithm, the JoinPathList becomes:




8
Employees
Job_History
Departments
Jobs
Locations
Departments
Location_Id
Employees
Name
Departments
Departments_Id
9
Employees
Job_History
Departments
Jobs
Locations
Countries
Locations
Country_Id
Employees
Name
Departments
Department_Id
10
Employees Job_History
Departments Jobs
Locations Countries
Employees
Name
Departments
Department_Id

Insert routine

When a new row R_m from table T_i get inserted do the following:

- Locate the entry of T_i in the JoinPathList
- From its adjacent List, locate the definition of the keys and inherited keys
- From Row R_m get the columns constituting the keys and the inherited keys
- Call AddJoinKey (T_i, Keys, InheritedKeys, DP_i) where DP_i is the row id of row R_m.

AddJoinKey ($T_{[i]}$, $[DP_i]$)

- Call AddKey ($B^+Tree(T_{[i]})$, $keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$) for the index of table $T_{[i]}$
- Locate the entry of $T_{[i]}$ in the JoinPathList
- From its adjacent List, locate the Table $T_{[k]}$ adjacent to it and do the following:
  - Locate the entry of $T_{[k]}$ in the JoinPathList
  - FindKey($B^+Tree(T_{[k]})$, $Keys_{[i]}$)
  - While found(keys[i]) do
    - ReturnKeys($B^+Tree(T_{[k]})$, $keys_{[k]}$, $InheritedKeys_{[k]}$, $[DP_k]$)
    - Locate the entry of $T_{[ik]}$ in the JoinPathList
    - From its adjacent List, locate the definition of the keys and inherited keys
    - From $keys_{[i],}$ $inheritedkeys_{[i]}$, $keys_{[k],}$ $inheritedkeys_{[k]}$ get the keys and inherited keys of $T_{[ik]}$
    - AddJoinKey ($T_{[ik]}$, $Keys_{[ik]}$, $InheritedKeys_{[ik]}$, $[DP_{ik}]$)
    - NextKey($B^+Tree(T_{[k]})$, $Keys_{[i]}$)

AddJoinKey ($T_{[i]}$, [DP_i])

In the same fashion when using an ordinary B^+Tree and one row get inserted, so we check the definition of the B^+Tree to get the necessary keys from the row to insert them, with B^{Join}Tree we check the definition to get the keys and the inherited keys.

Call AddjoinKey(T_i,[keys],[DP_i])

Delete routine

Call DelJoinKey (T_i, DP_i) where DP_i is the data pointer of the deleted row from the base table T_i.

DelJoinKey ($T_{[i]}$, $[DP_i]$)

- Call DelKey ($B^+Tree(T_{[i]})$, $keys_{[i]}$, $[DP_i]$) for the index of table $T_{[i]}$
- Locate the entry of $T_{[i]}$ in the JoinPathList
- From its adjacent List, locate the Table $T_{[k]}$ adjacent to it and do the following:
  - Locate the entry of $T_{[k]}$ in the JoinPathList
  - FindKey($B^+Tree(T_{[k]})$, $Keys_{[i]}$)
  - While found(keys[i]) do

    ReturnKeys($B^+Tree(T_{[k]})$, $keys_{[k]}$, $InheritedKeys_{[k]}$, $[DP_k]$)

    Locate the entry of $T_{[ik]}$ in the JoinPathList

    From its adjacent List, locate the definition of the keys and inherited keys

    From $keys_{[i],}$ $inheritedkeys_{[i]}$, $keys_{[k],}$ $inheritedkeys_{[k]}$ get the keys and inherited keys of $T[_{ik]}$

    DelJoinKey ($T_{[ik]}$, $Keys_{[ik]}$, $[DP_{ik}]$)

    NextKey($B^+Tree(T_{[k]})$, $Keys_{[i]}$)

B⋈Tree with incremental Join

Due to the fact that join is commutative and associative and we are working on Virtual Tables and using indexes on them; it is possible instead of calculating all the join combinations to calculate incrementally the join.

This issue works just when the n tables are in direct path join between them but if they are not we are not interested.

Giving a casual order for the tables.

Beginning from Table 0, get a table T_i in direct join with it.

A Join Path List comes out with 2 entries from T_o to T_i and from T_i to T_0.

The index number start always with 0.

Repeat, with T_{0i} and get a next table that is in direct join with T_0 or with T_i, the process continue till we scan all the tables.

This algorithm is linear, is 2*n – 1.

Complexity of the algorithm for the creation of JoinPathList.

The complexity for the creation of JoinPathList structure is:
2*n-1 where n is the number of tables in join.

Proof:
We can prove it by induction on the number of tables in join.

For m = 1:

The complexity should be 2*1-1 = 1 in fact it is the only table that get inserted in the JoinPathList.

For m = n-1:

Suppose that the number of tables in JoinPathList is 2*(n-1)-1.

For m = n:

The n^{th} table get inserted as a Vertex in the JoinPathList at the beginning of the algorithm.

The n^{th} table get inserted in queue and path dynamic arrays because the n tables are in join and at least there is one table in the (n-1) remaining table that is in join with the n^{th} table.

So when the algorithm run at certain point should execute:

$$\mathbf{T_{[buf]} + = T_i}$$

$$\mathbf{Insert\ NodesList[T_{[buf]}] = T_{[buf]}}$$

where T_l is $T_{n,}$ so the number of tables in JoinPathList are: 2*(n-1)-1 + 1 + 1 = 2*n-1

Complexity of the algorithm for the insertion and deletion.

Delete is symmetric to insert in the algorithm in the sense where there is an insert we use a delete, so they have both the same complexity.

When inserting a new row in the database we use the $B^{Join}Tree$ structure to drive us in the insert for the join.

Suppose that the order of the $B^{+}Tree$ is m, the number of elements in the (2*n-1) $B^{+}Trees$ is p_i where i is the index of the B+Tree and that in average there is l_i elements satisfying the join between every pair of tables.

In the worst case when get inserted row with the lowest order tables T_0 and T_1 in this case we call recursively the insert procedure for (2*n – 1) – (n - 1) = n times.

The complexity will be: $Ord(n * l_i * \log_{m/2}(p_i))$

Complexity of the algorithm for the other operation.

The only B^+Tree of our interest for the scan is the one with the latest index that have the join of the tables inside it.

Suppose that the number of elements for the latest index is $p_{(2*n-1)}$ so the other operations on this B^+Tree for find, search, prev, next,… are the same as for normal B^+Tree with the same number of elements.

Steps to generate Join Path List for the join sequence $T_0 \ldots T_m$

insert all the Names of Base Tables from the join sequence $T_0 \ldots T_m$ as vertexes in the PathJoinList

create 2 dynamic arrays queue and path

insert T_0 into path

insert T_0 into queue

repeat

- **$T_{Element}$ = First Table in queue**
- **for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do**
  - **if the Link Item is in the join sequence then**
    - **if path doesn't contain the Link Item then**
      - **insert Link Item into path**
      - **insert Link Item into queue**
- **remove $T_{Element}$ from queue**

until queue is empty

Steps to generate Join Path List for the join sequence $T_0 \ldots T_m$ (continue)

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

- **take one T_i at a time**
- **PathJoinAdjacentList(T_i) = $T_{[buf]}$**
- **Key(T_i) = getFirstAdjacentListKey(T_i,$T_{[buf]}$)**
- **PathJoinAdjacentList($T_{[buf]}$) = T_i**
- **Key($T_{[buf]}$) = getFirstAdjacentListKey ($T_{[buf]}$,T_i)**
- **$T_{[buf]}$ + = T_i**
- **Insert NodesList[$T_{[buf]}$] = $T_{[buf]}$**

Steps to generate Join Path List for the join sequence $T_0 \ldots T_m$ (continue)

create a structure buf with 2 fields: Table and Key

for all the tables in PathJoinList going downward do

take one $T_{[i]}$ at a time

for all Base Tables in$T_{[i]}$ do

take one T_k at a time

for every buf.Table = T_k do

if (buf.key != Key($T_{[i]}$)) and (buf.Key not in AuxKey($T_{[i]}$)) then AuxKey($T_{[i]}$) += buf.key

if T_l is the table from which comes Key($T_{[i]}$) then

buf.Table = T_l

buf.key = Key($T_{[i]}$)