B$^{\bowtie}$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…R_z in T_k that satisfy the join condition with R_m and insert Rows references to $R_mR_{n....}$ R_mR_z in the virtual table T_{ik}.

The process should be repeated for $R_mR_{n....}$ R_mR_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:

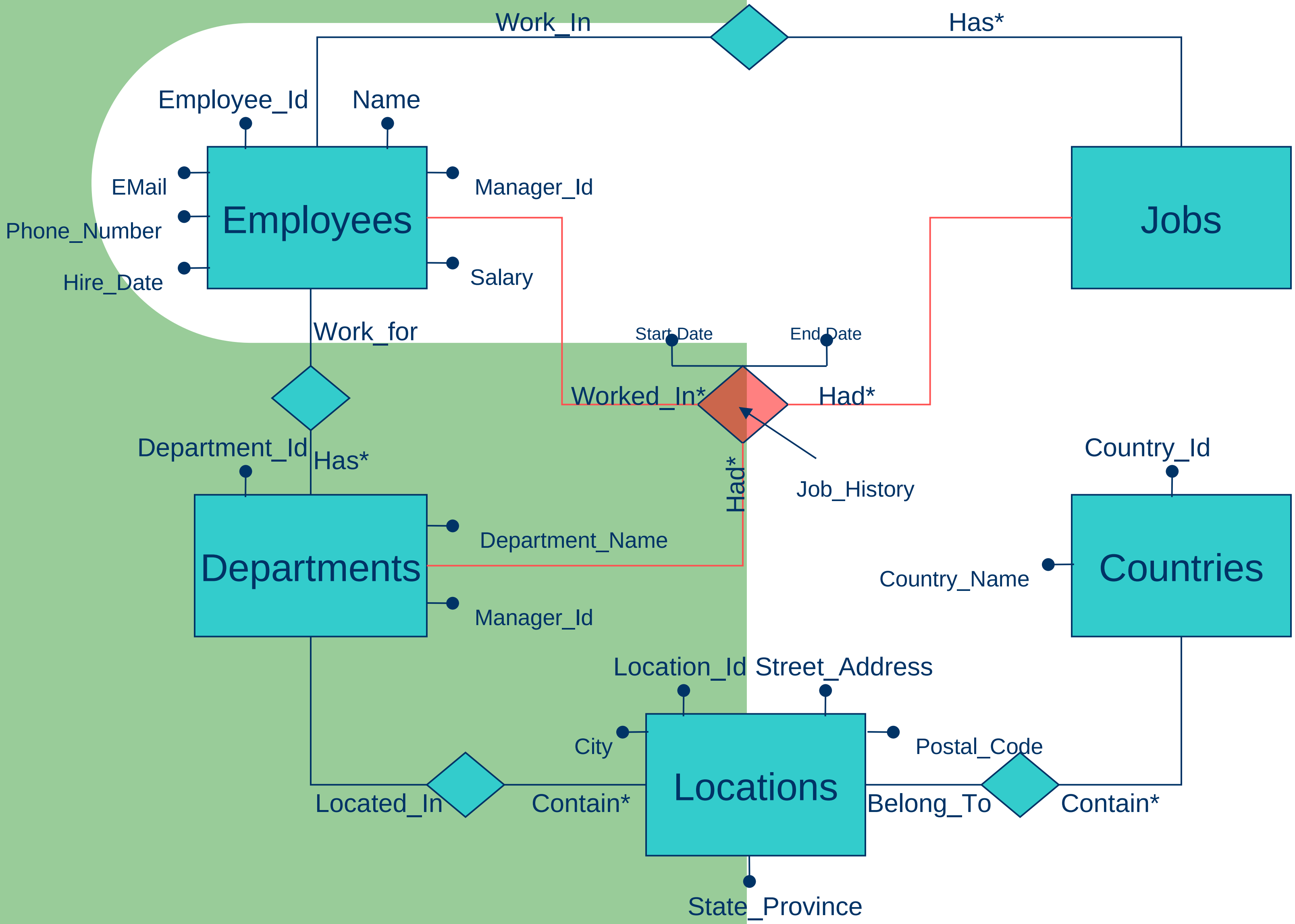
List for every table T_i in the database all those tables in direct Join with it.

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*
Job_History
Country_Id
Had*
Department_Name
Departments
Country_Name
Countries
Manager_Id
Location_Id
Street_Address
City
Postal_Code
Locations
Located_In
Contain*
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
```

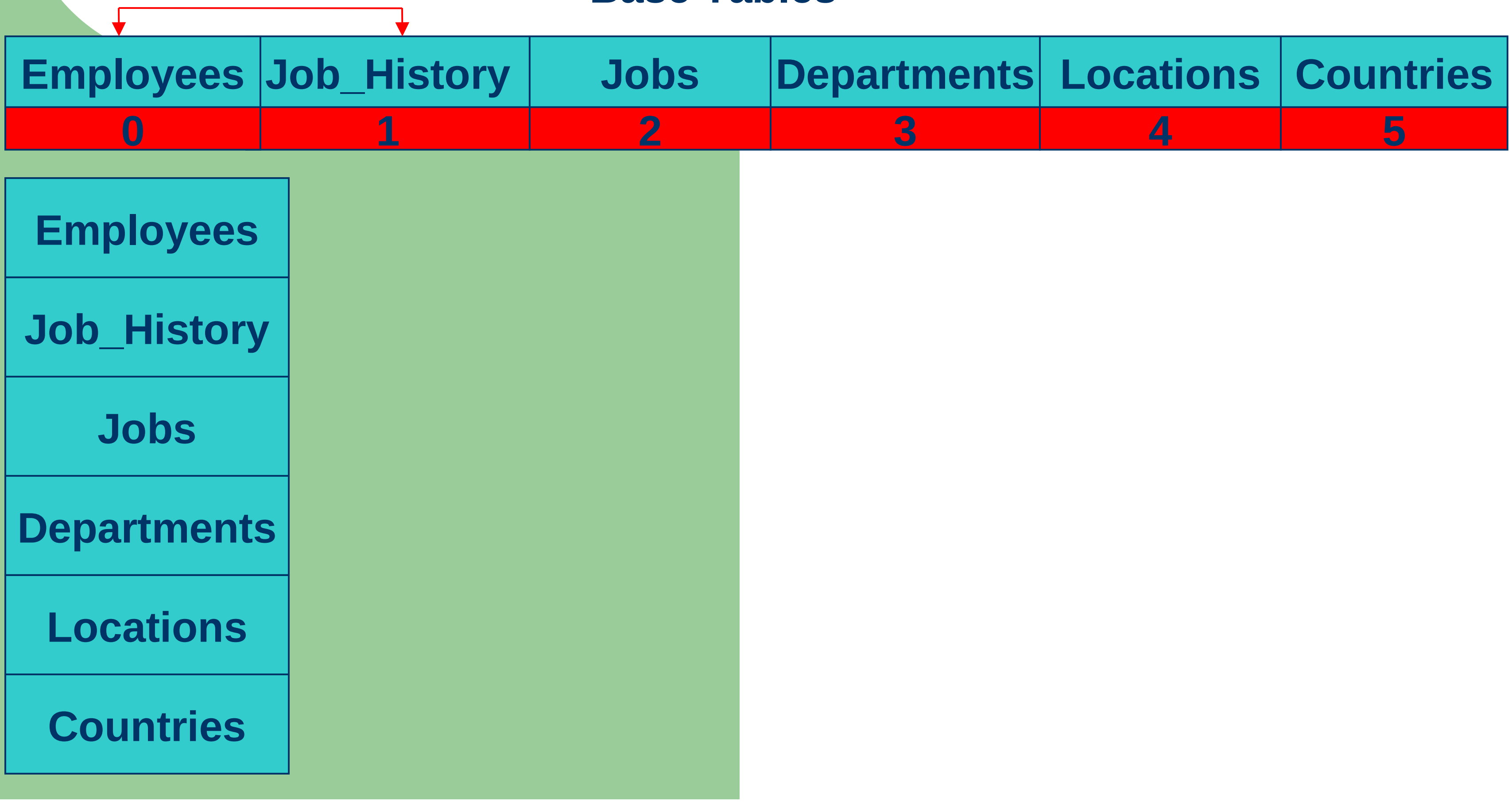
Base Tables

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

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



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

Employees

Job_History

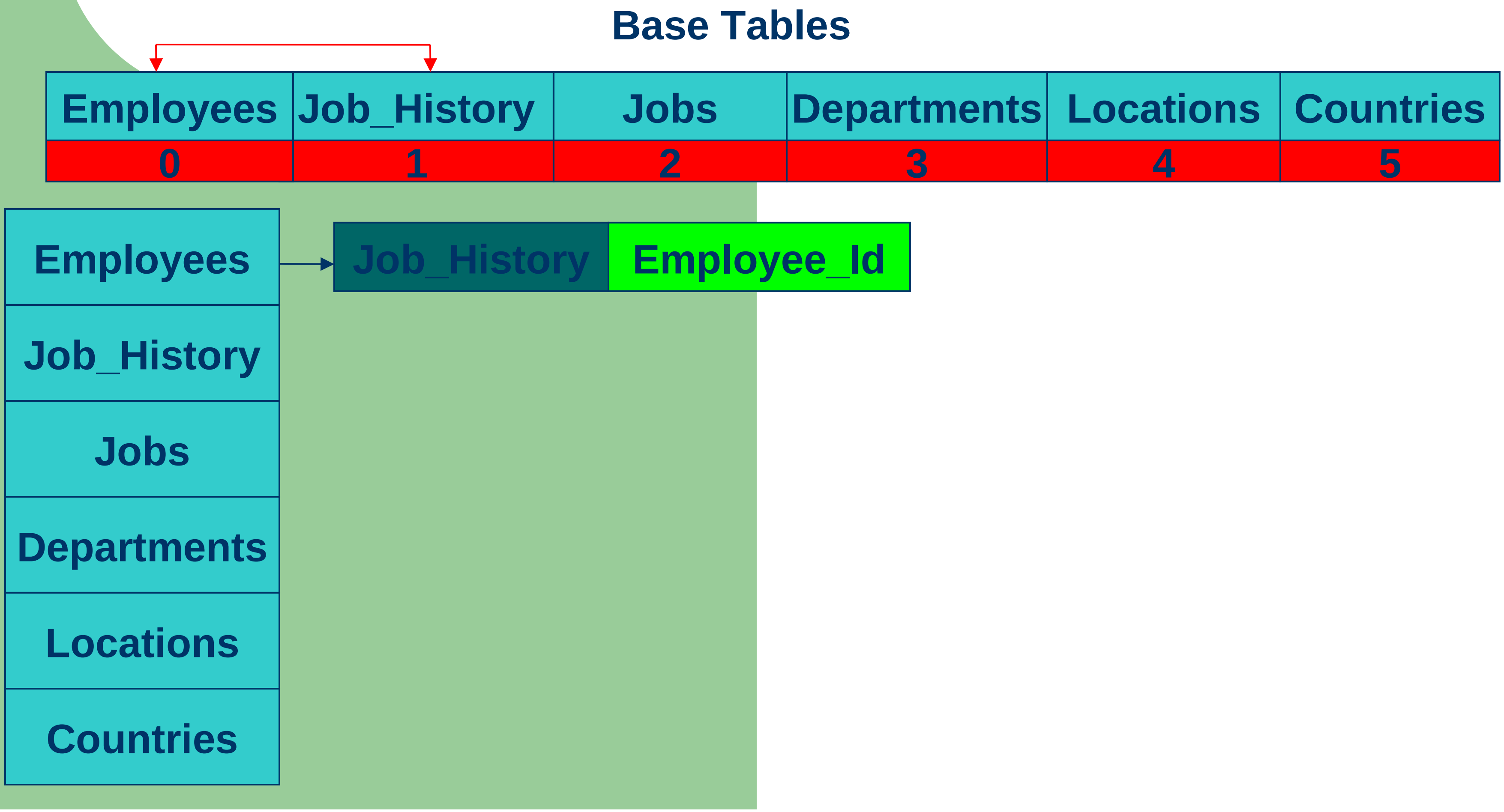
Jobs

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
```

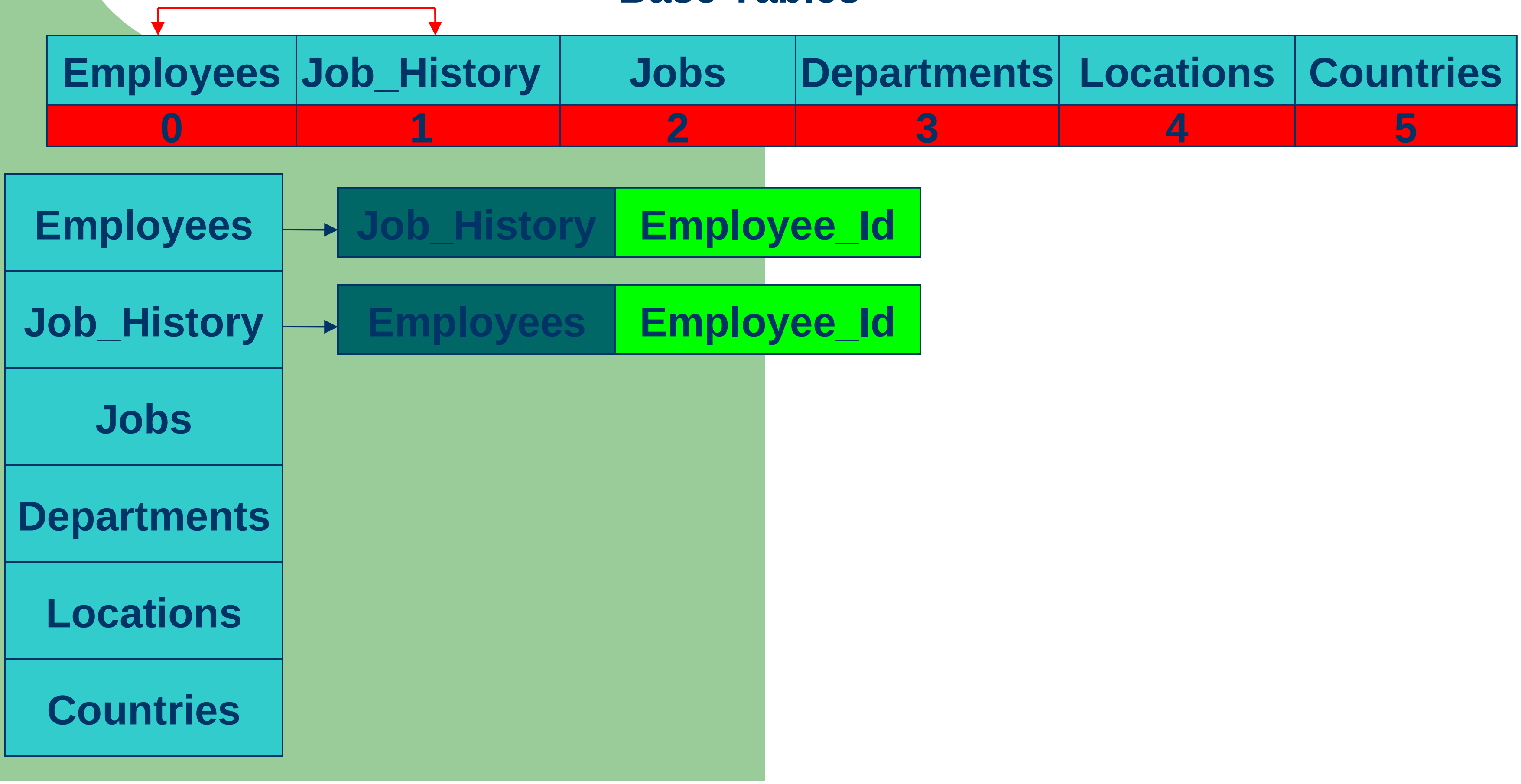


Base Tables

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
```

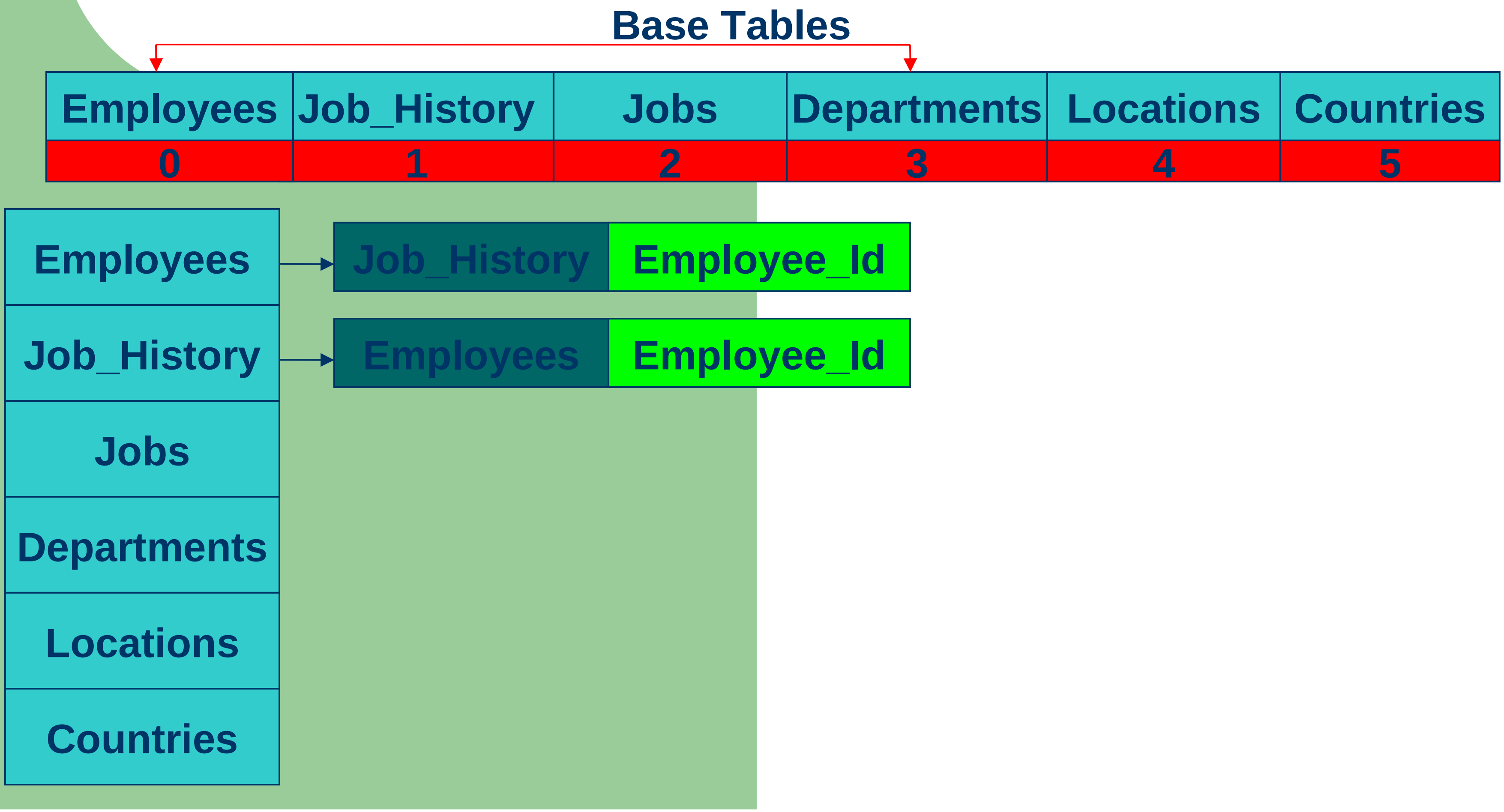
Base Tables



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

Employees	Job_History	Employee_Id
Job_History	Employees	Employee_Id
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
```

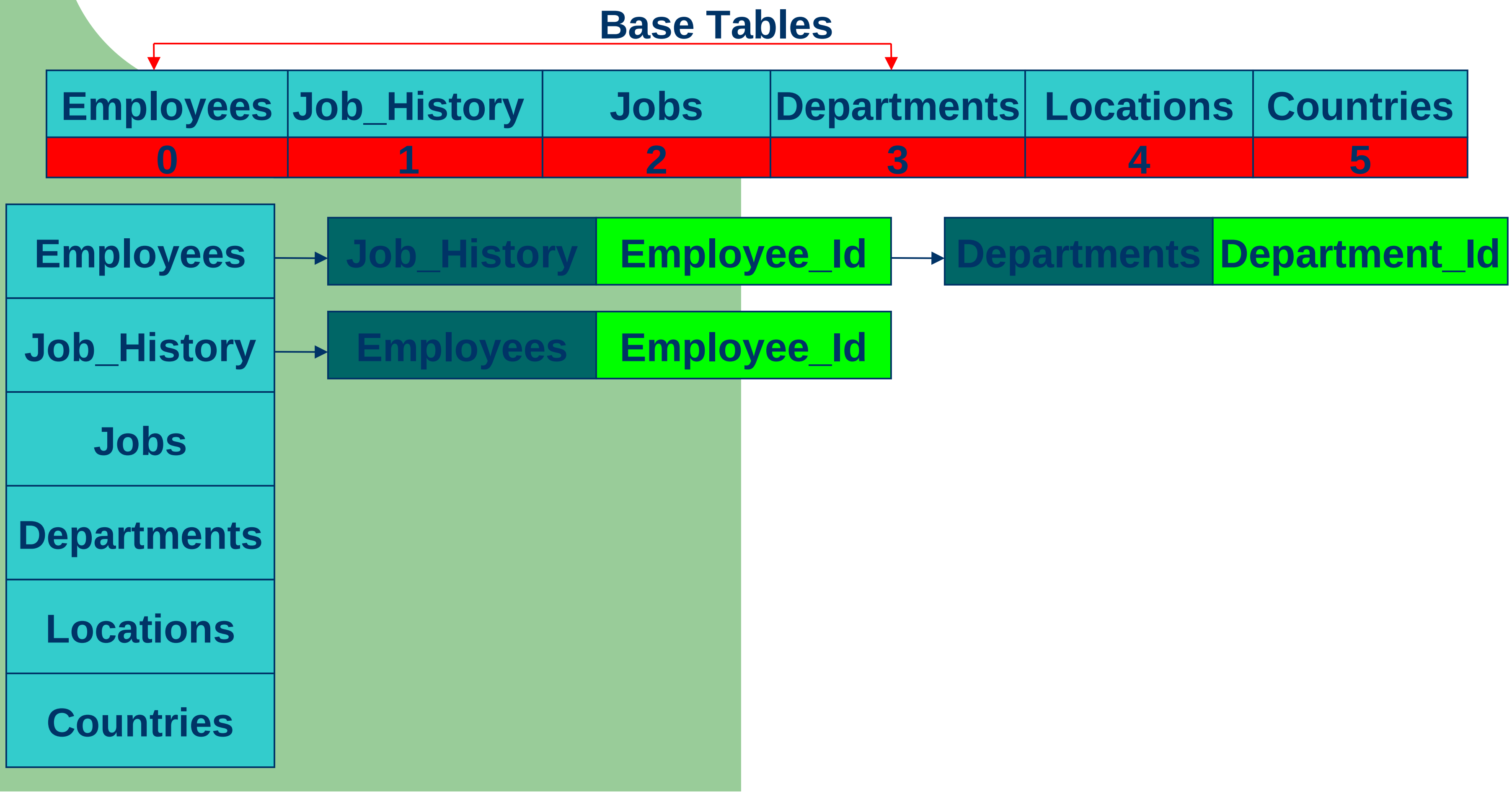


Base Tables

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

Employees	Job_History	Employee_Id
Job_History	Employees	Employee_Id
Jobs		
Departments		
Locations		
Countries		

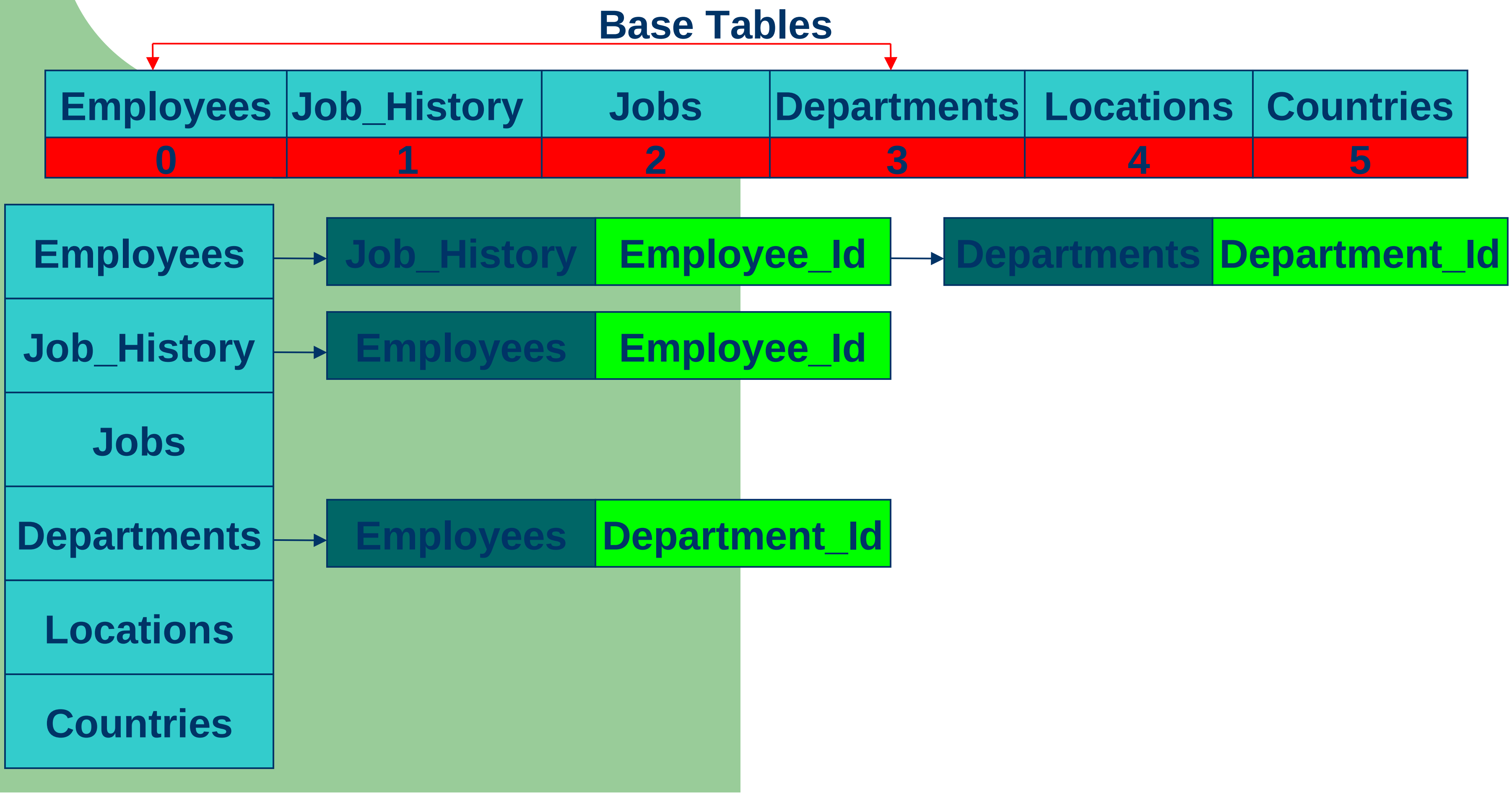
```
generateJoinGraph (in BaseTables; out JoinGraph)
insert the base tables as vertexes of the graph
for every direct join between 2 tables of the form $T_i$ and $T_k$ where $T_i$ is the table of order i and $T_k$ is the table of order k as defined by the DBA do
    AdjacentList[$T_i$] += $T_k$ follow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



Base Tables

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 | Employee_Id → Departments | Department_Id

Job_History → Employees | Employee_Id

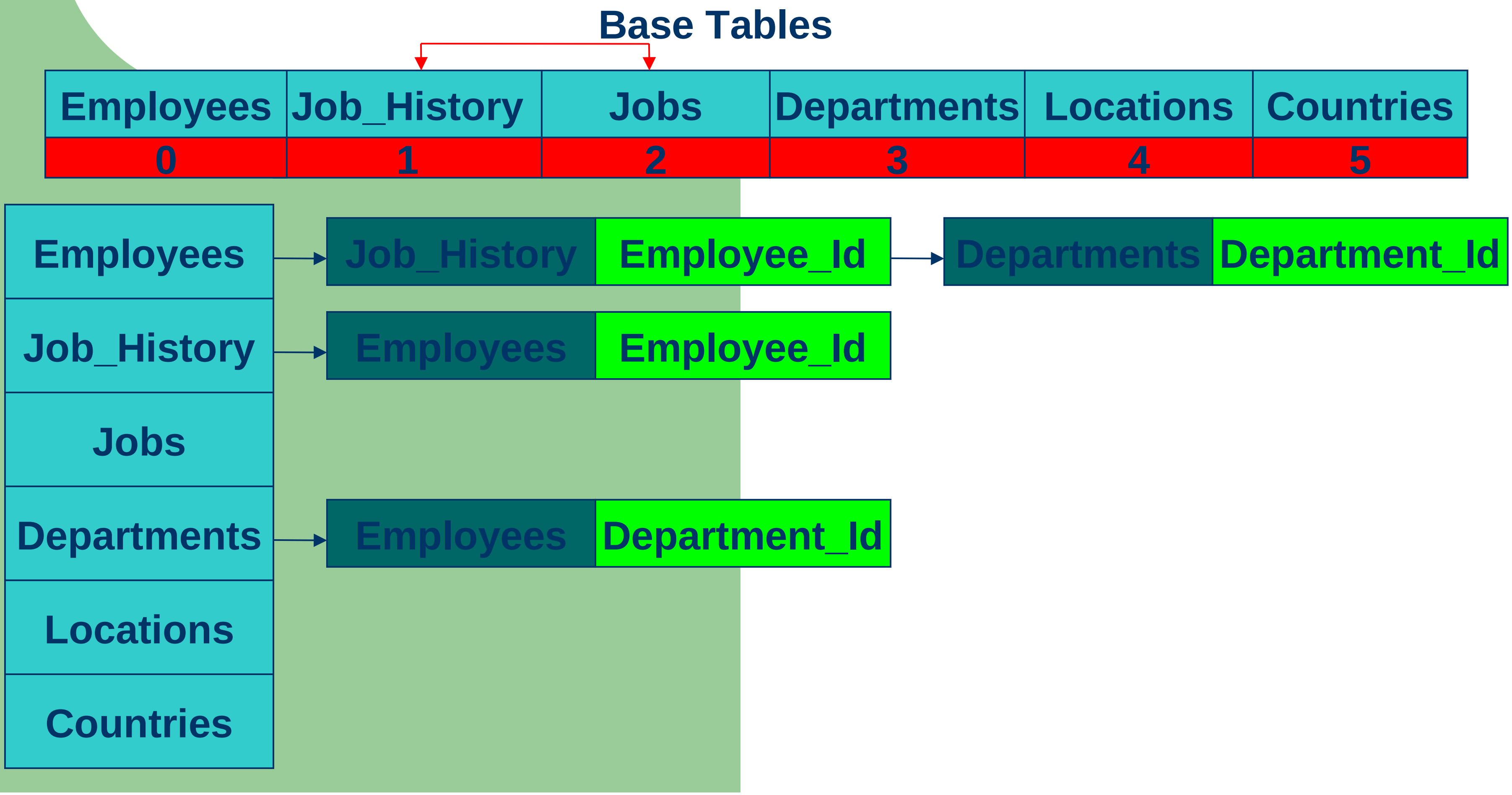
Jobs

Departments → Employees | Department_Id

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

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

Employees → Job_History Employee_Id → Departments Department_Id

Job_History → Employees Employee_Id

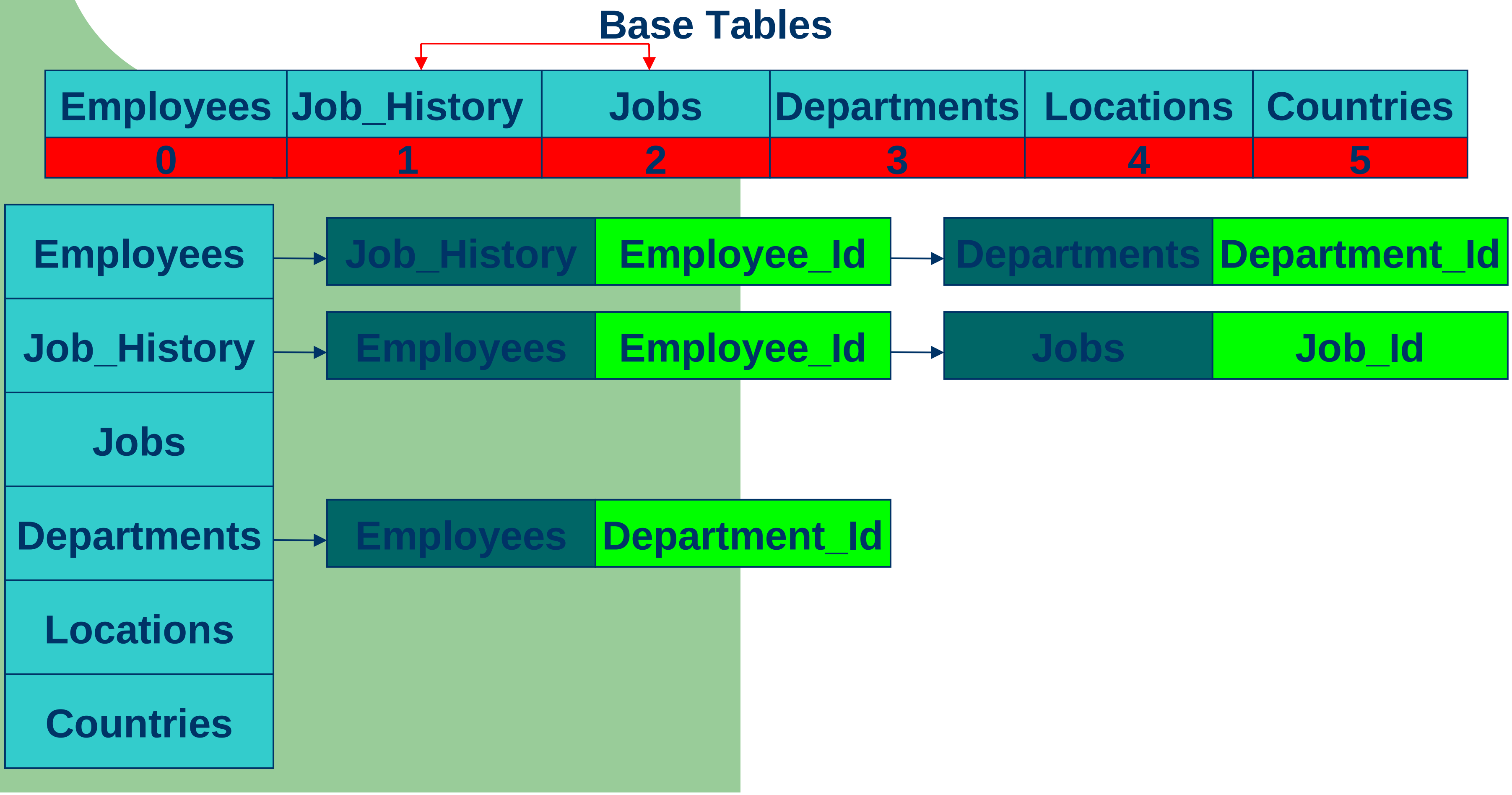
Jobs

Departments → Employees Department_Id

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

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

Employees → Job_History | Employee_Id → Departments | Department_Id

Job_History → Employees | Employee_Id → Jobs | Job_Id

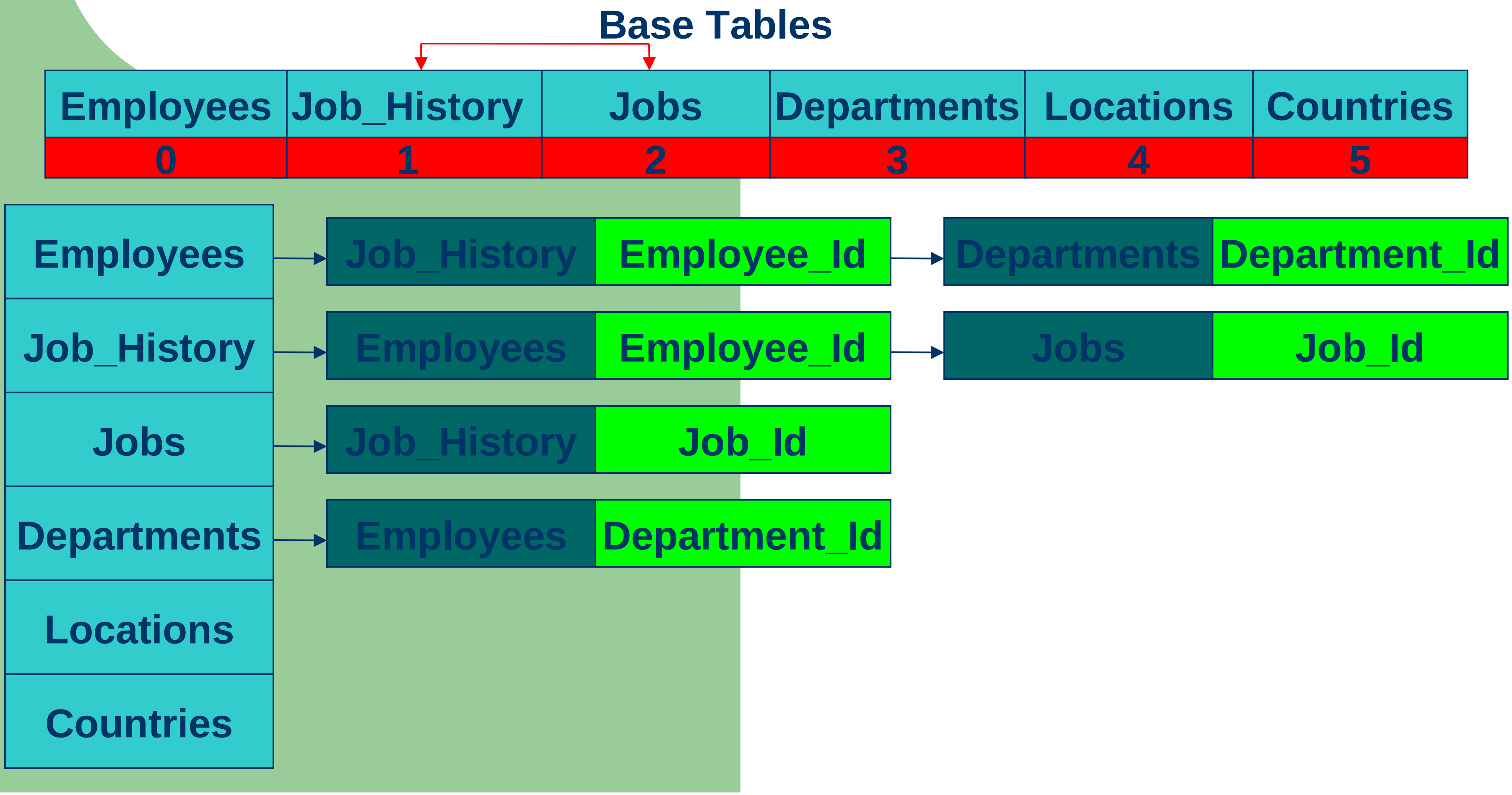
Jobs

Departments → Employees | Department_Id

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
```



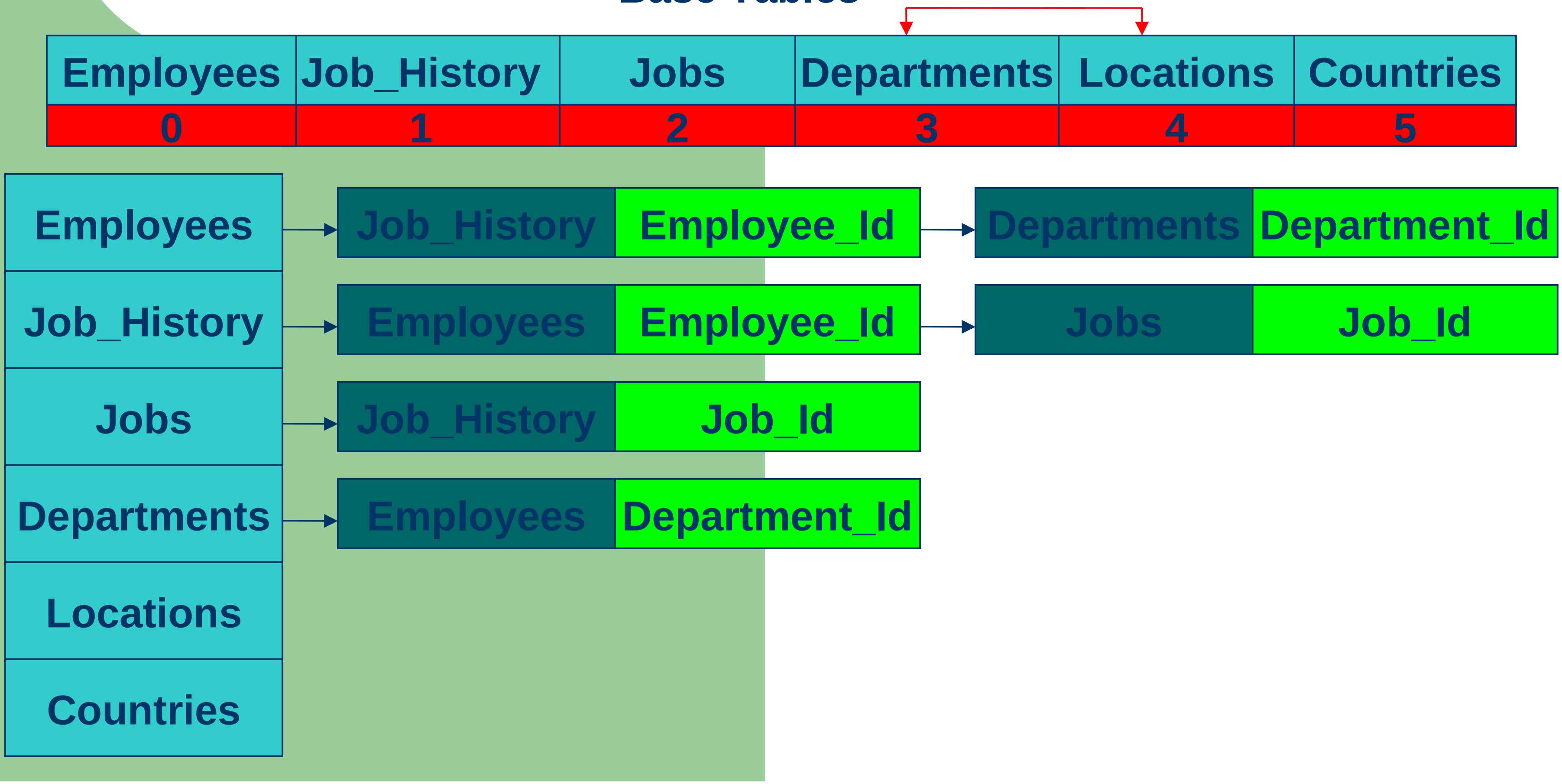
Base Tables

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

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				
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

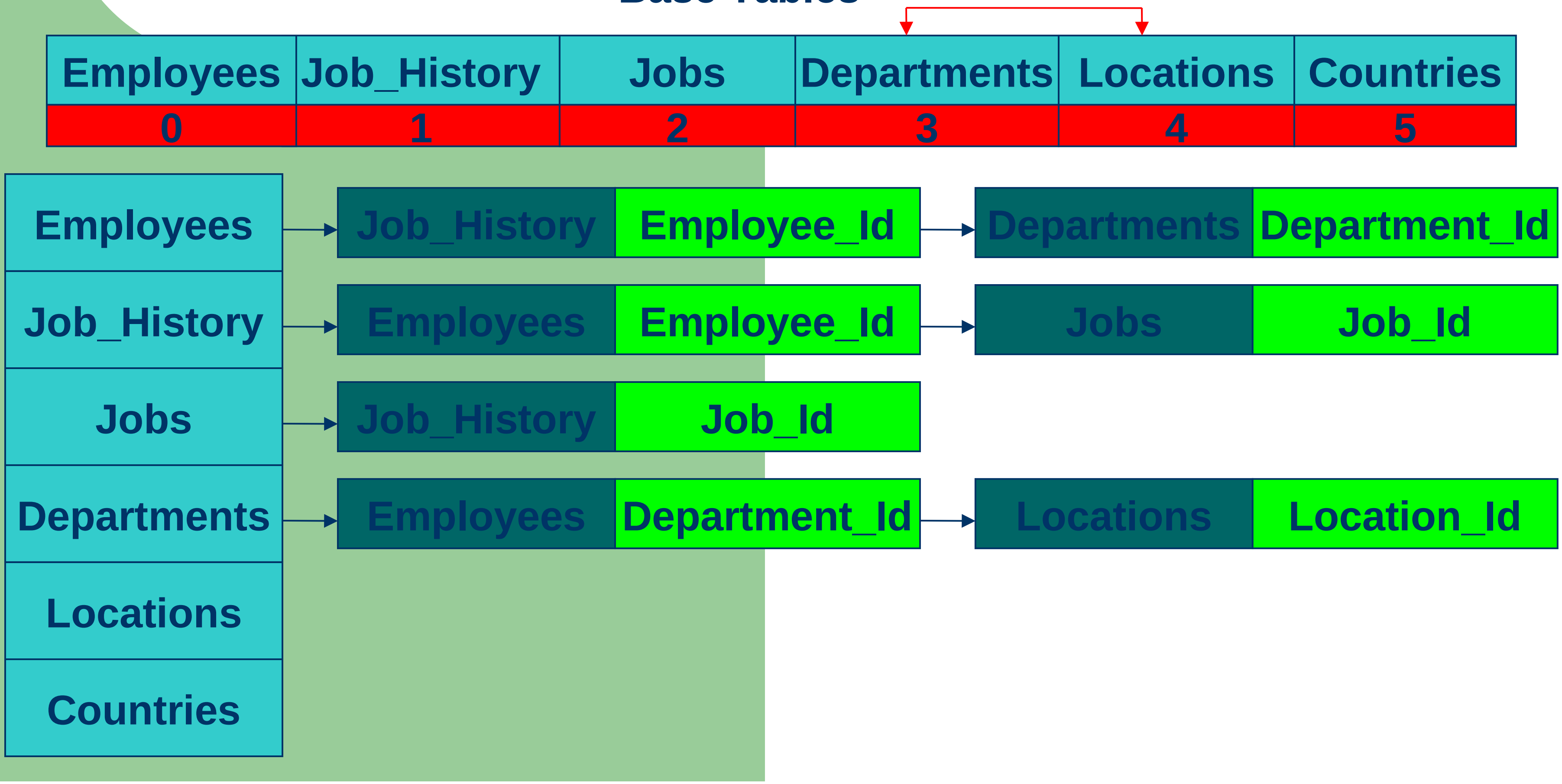


Employees	Job_History	Jobs	Departments	Locations	Countries
0	1	2	3	4	5

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				
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

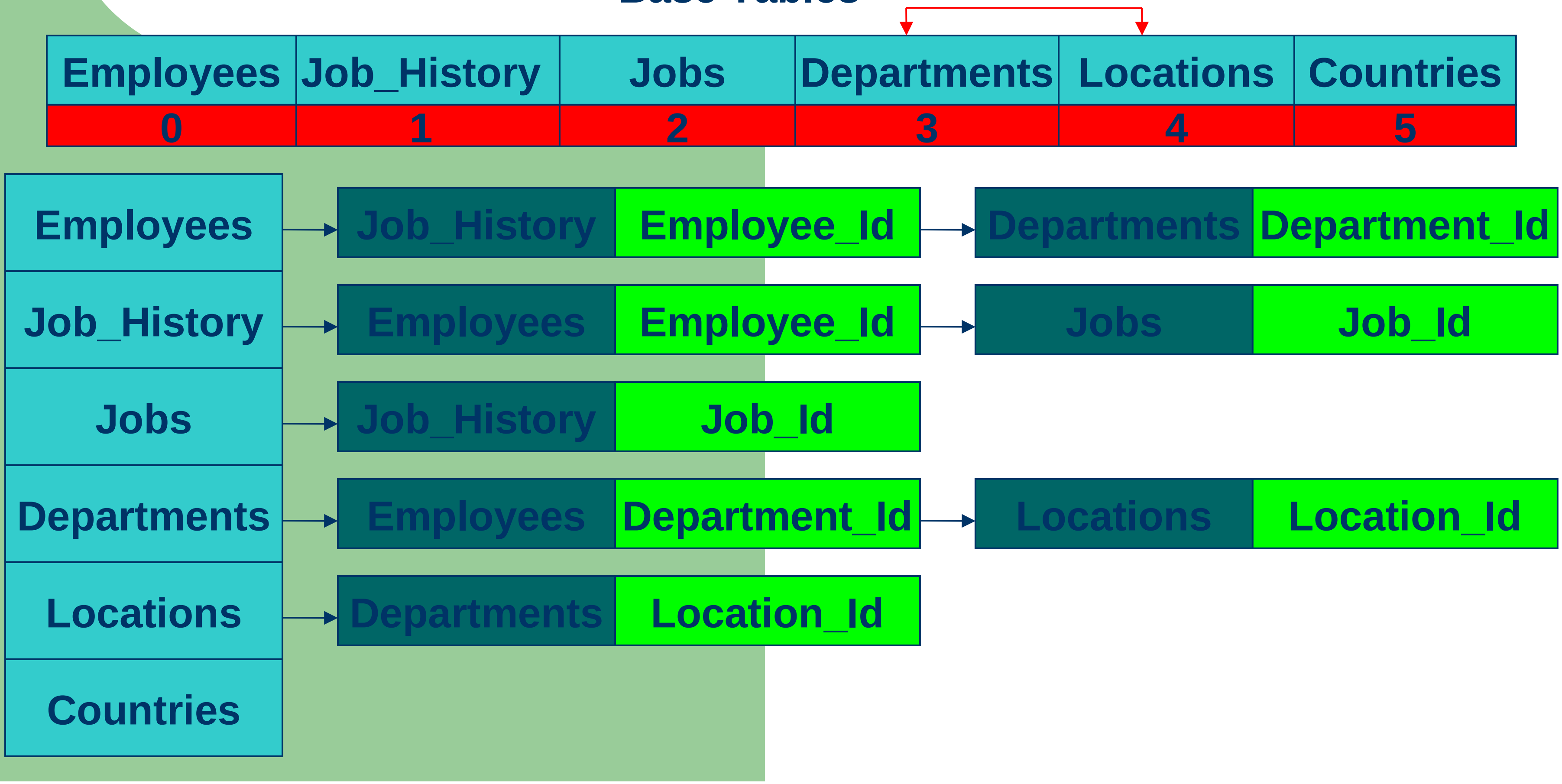


Employees	Job_History	Jobs	Departments	Locations	Countries
0	1	2	3	4	5

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				
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



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

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

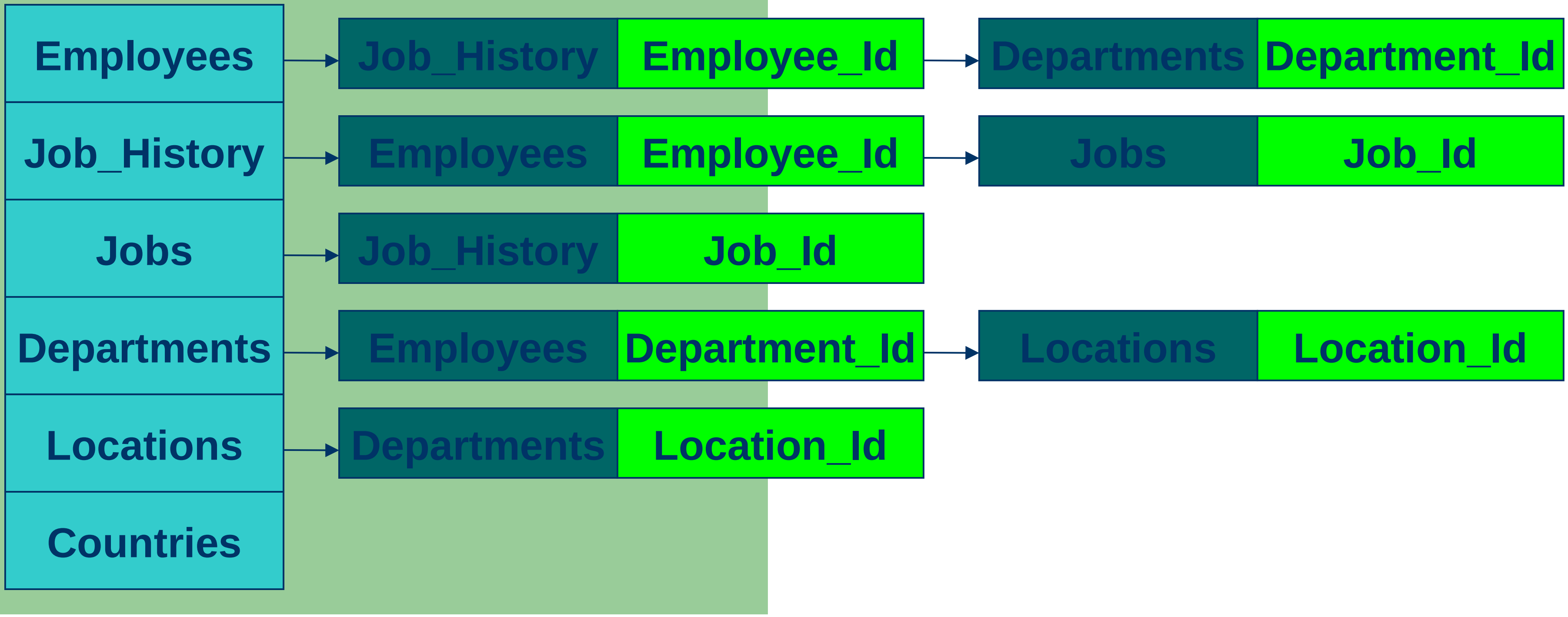
Locations → Departments | Location_Id

Countries

```
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 | 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

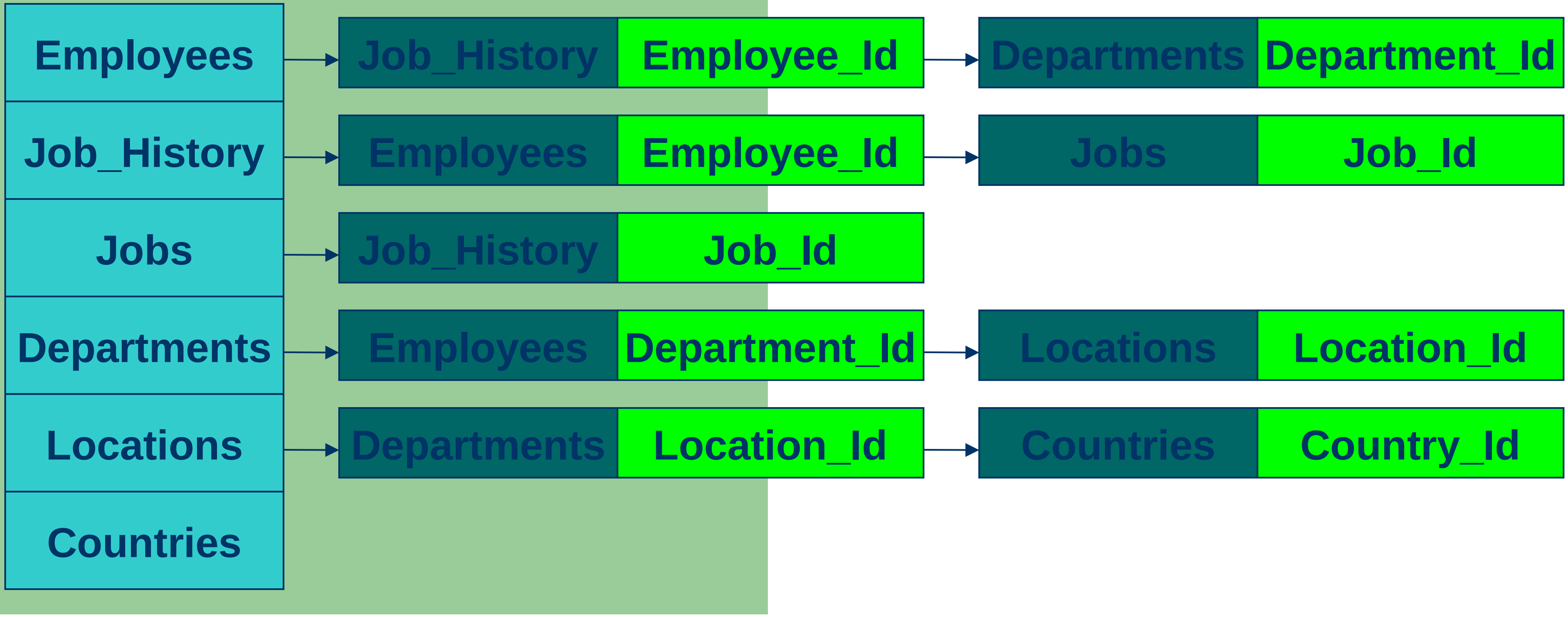
Locations → Departments | Location_Id

Countries

```
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 Employee_Id → Departments Department_Id

Job_History → Employees Employee_Id → Jobs Job_Id

Jobs → Job_History Job_Id

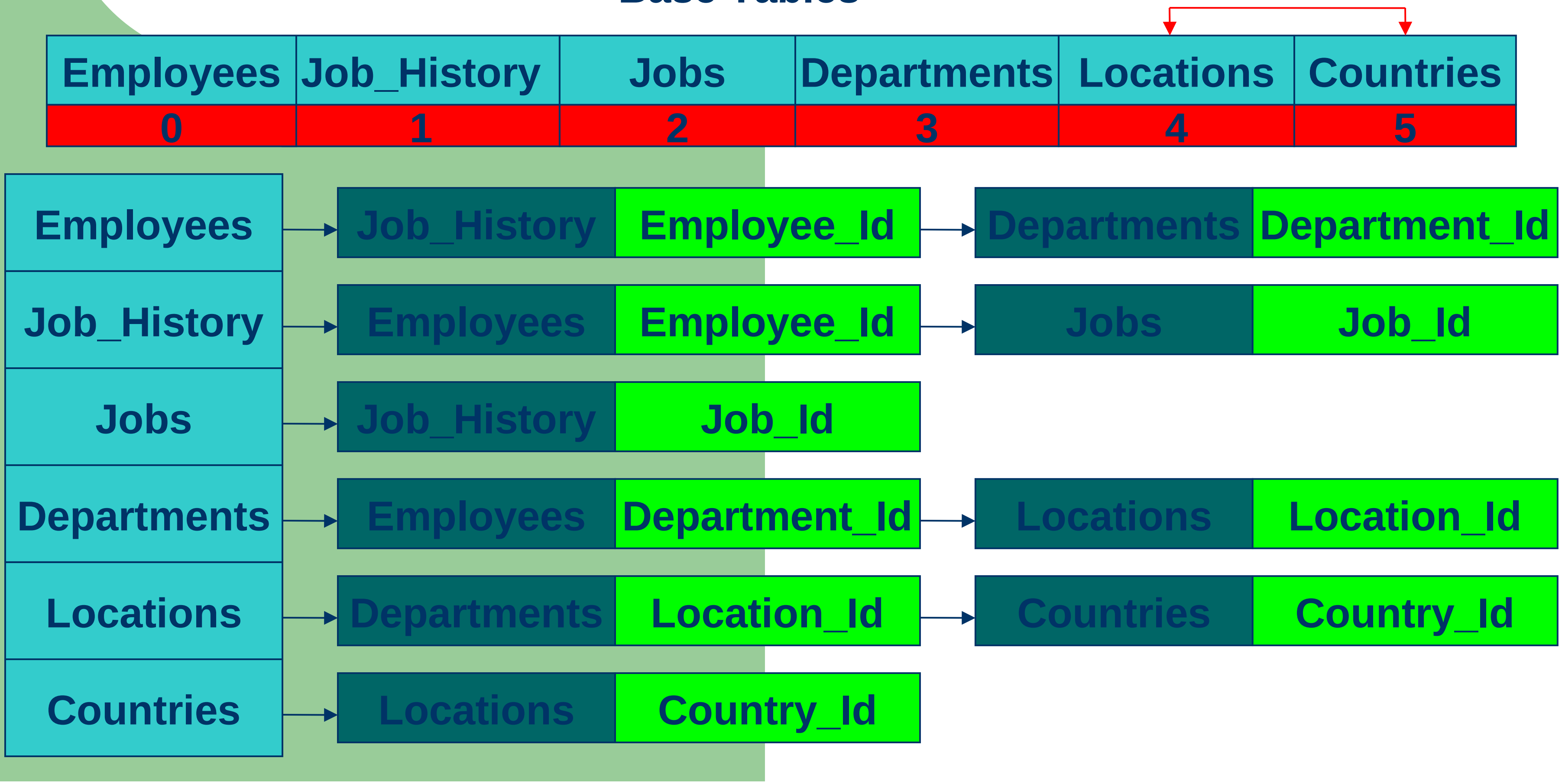
Departments → Employees Department_Id → Locations Location_Id

Locations → Departments Location_Id → Countries Country_Id

Countries

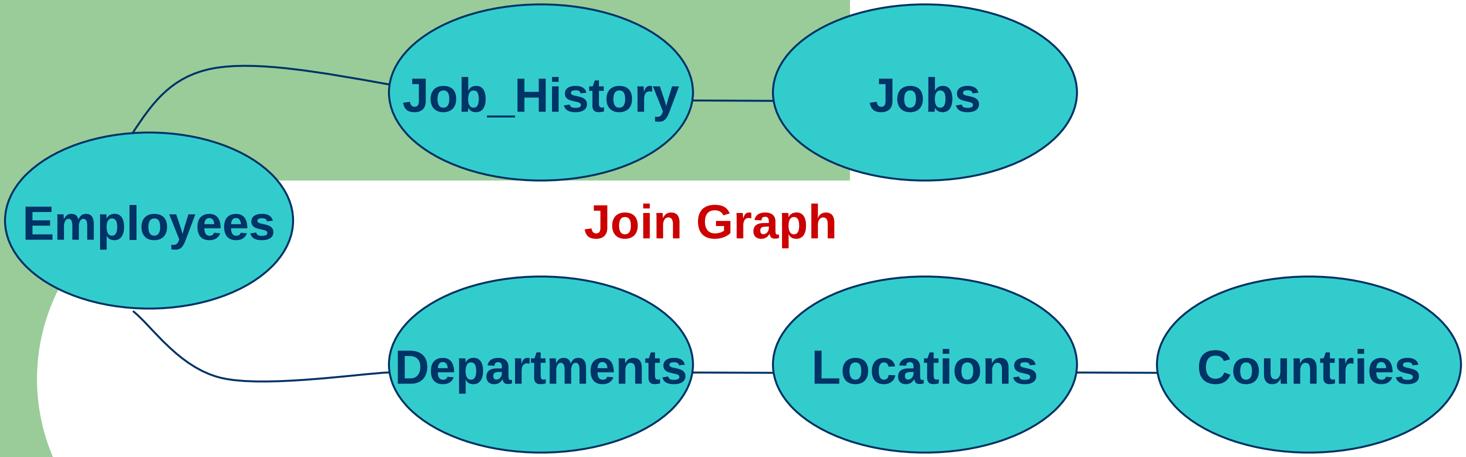
```
generateJoinGraph (in BaseTables; out JoinGraph)
insert the base tables as vertexes of the graph
for every direct join between 2 tables of the form $T_i$ and $T_k$ where $T_i$ is the table of order i and $T_k$ is the table of order k as defined by the DBA do
    AdjacentList[$T_i$] += $T_k$ follow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

Base Tables



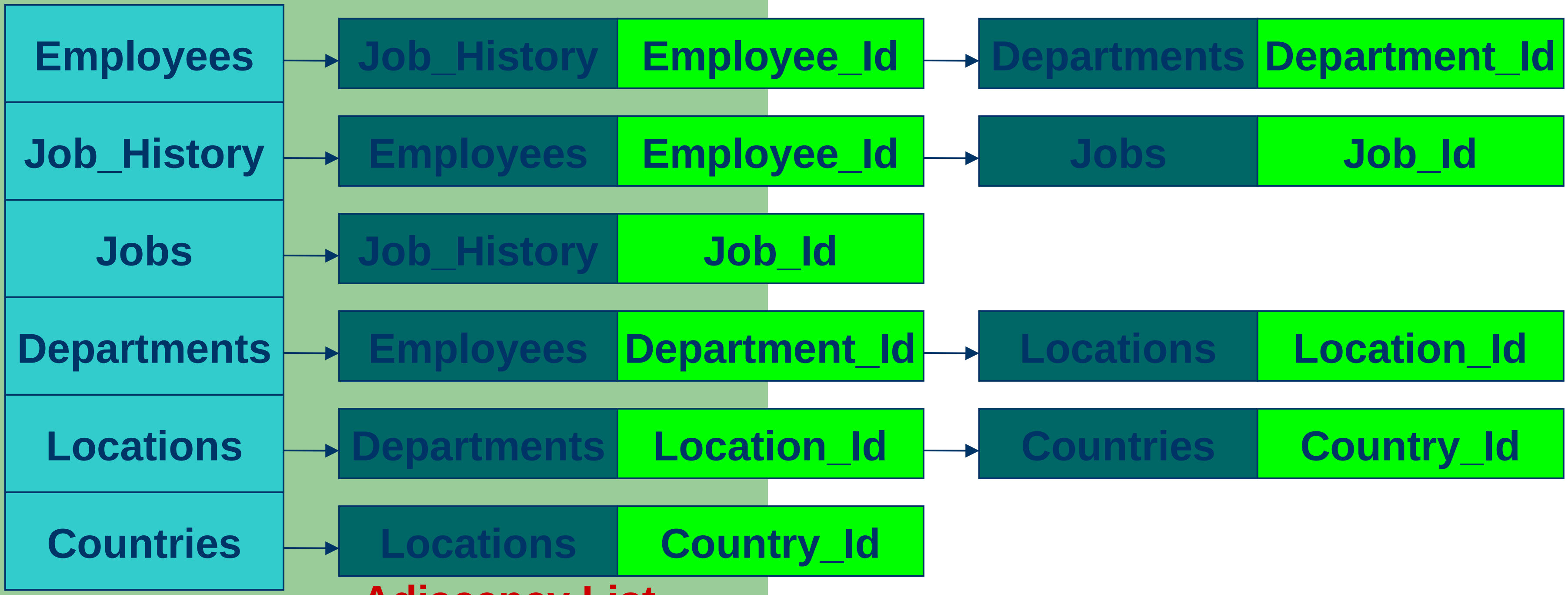
Employees	Job_History	Jobs	Departments	Locations	Countries
0	1	2	3	4	5

Employees	Job_History	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		



Join Graph

Linked List representation of the 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

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 $T_0 \ldots T_{m \text{ (continue)}}$

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 $T_0 \ldots T_{m \text{ (continue)}}$

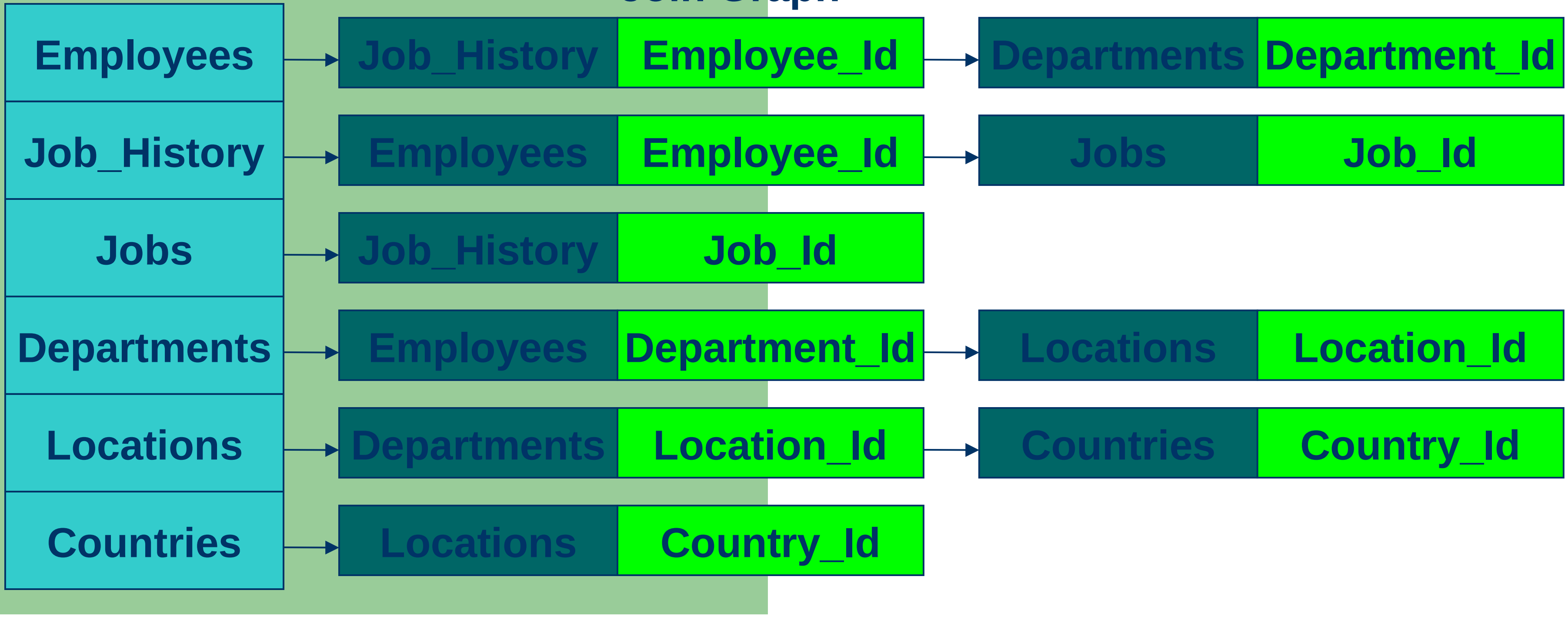
```
create a structure buf with 2 fields: Table and Key
for all the tables in JoinPathList going downward do
    take one $T_{[i]}$ at a time
    for all Base Tables in$T_{[i]}$ do
        take one $T_k$ at a time
        for every buf.Table = $T_k$ do
            if (buf.key != Key($T_{[i]}$) ) and (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 $T_{El}$ from queue
unt
```

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

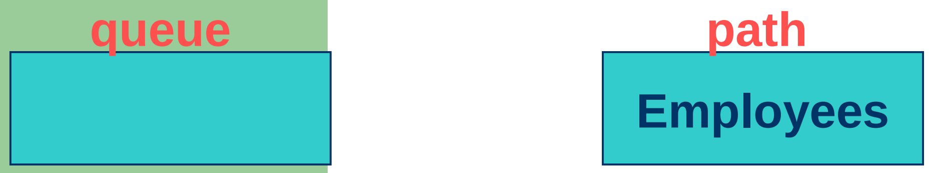
```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0$…$T_m$ be the base tables
create 2 dynamic arrays queue and path
insert $T_0$ into path
insert $T_0$ into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ 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 $T_0$…$T_m$ 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 $T_0$…$T_m$ 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
```

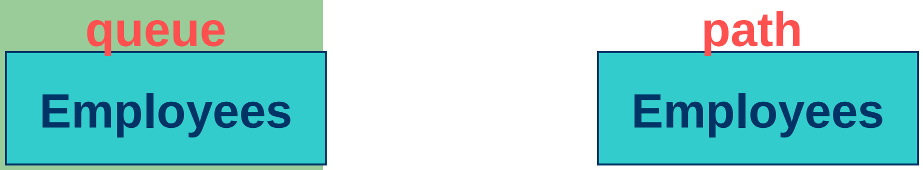


queue

path

Employees

```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0$…$T_m$ 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 $T_0 … T_m$ 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	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

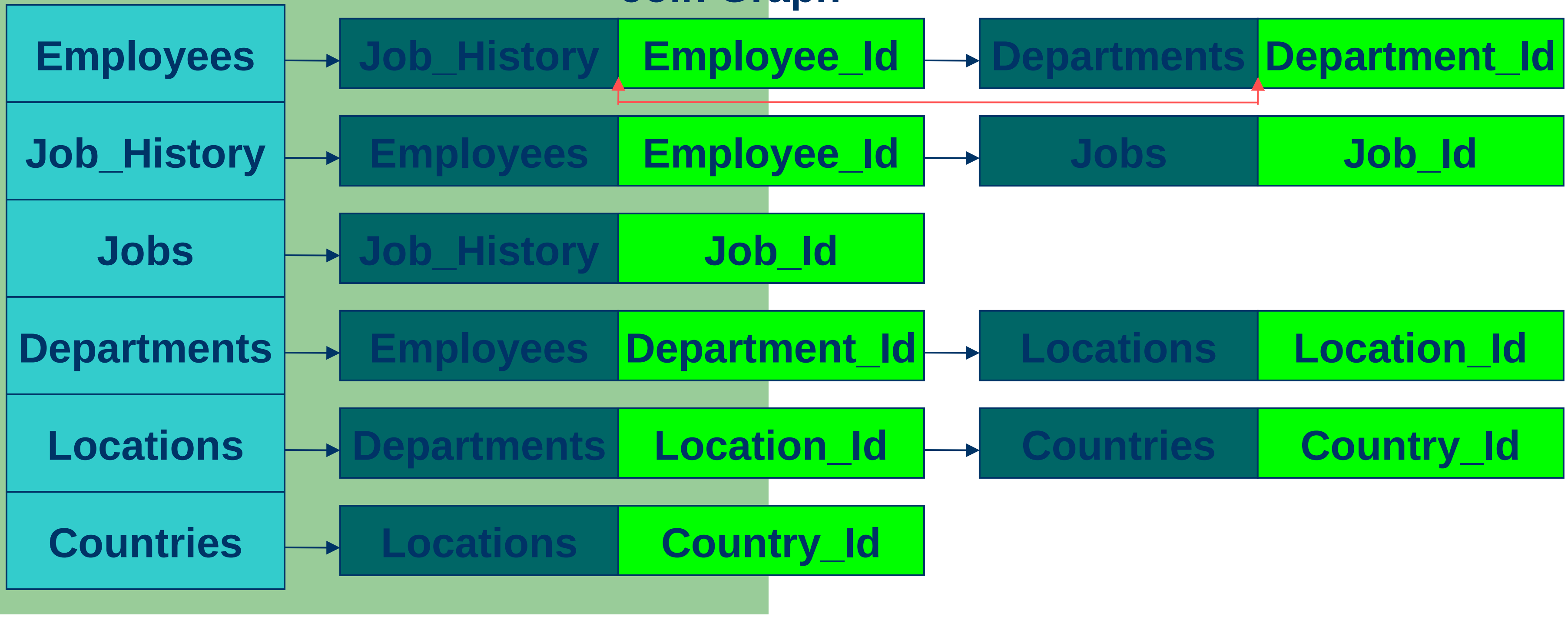
Employees

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
```

Join Graph



Employees → Job_History | Employee_Id → Departments | Department_Id

Job_History → Employees | Employee_Id → Jobs | Job_Id

Jobs → Job_History | Job_Id

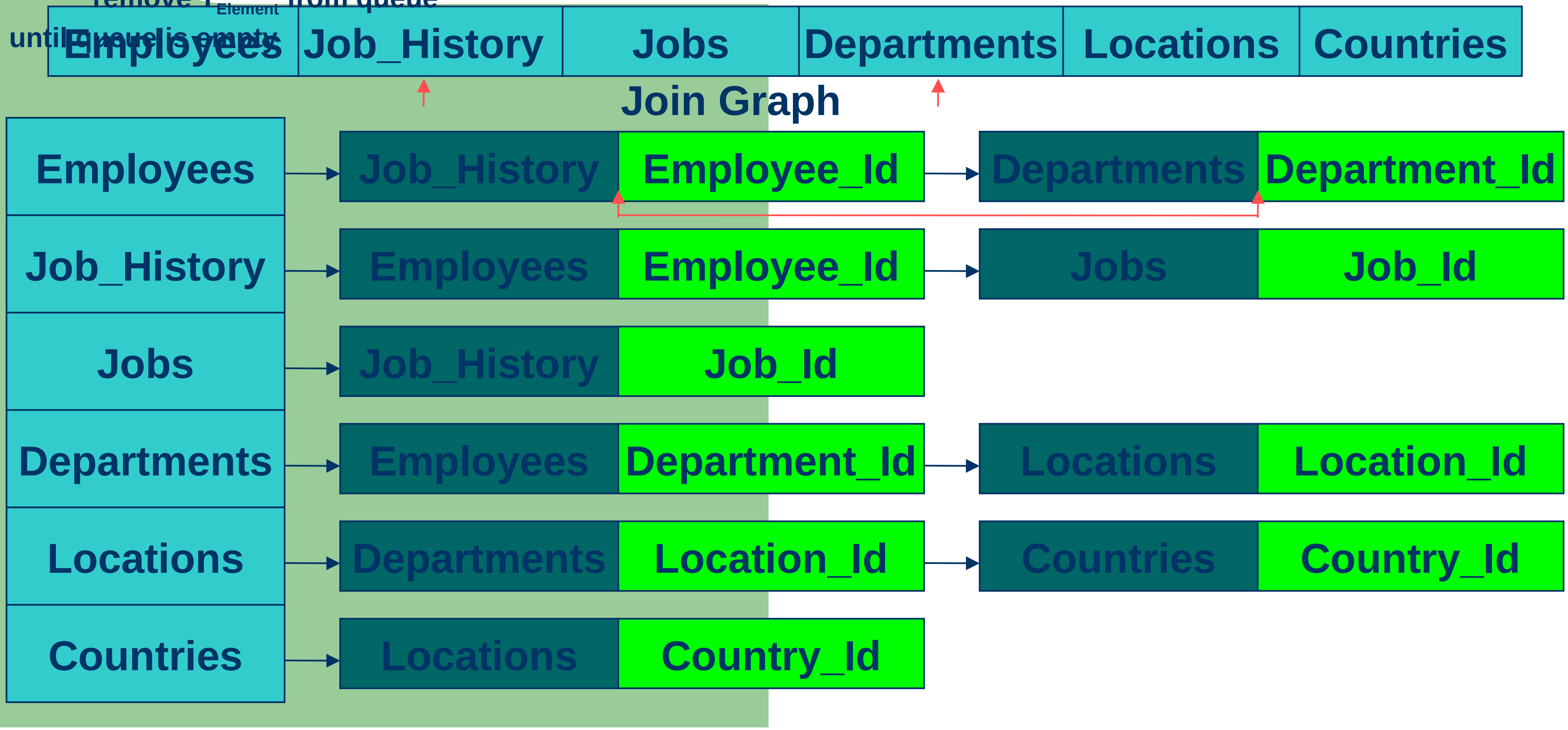
Departments → Employees | Department_Id → Locations | Location_Id

Locations → Departments | Location_Id → Countries | Country_Id

Countries → Locations | Country_Id

```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0…T_m$ 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

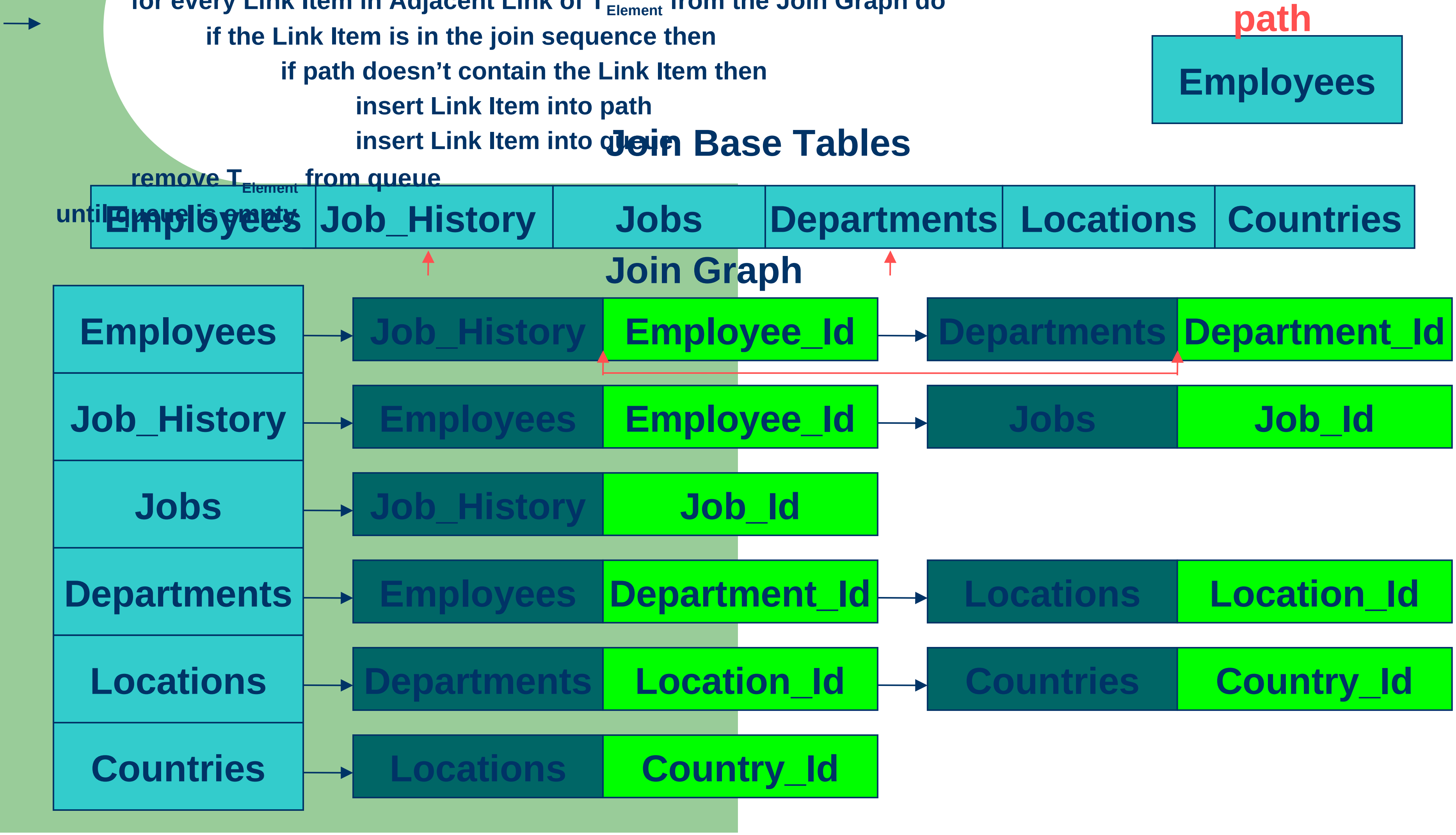


Employees	Job_History	Jobs	Departments	Locations	Countries

Join Graph

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 $T_0$…$T_m$ 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

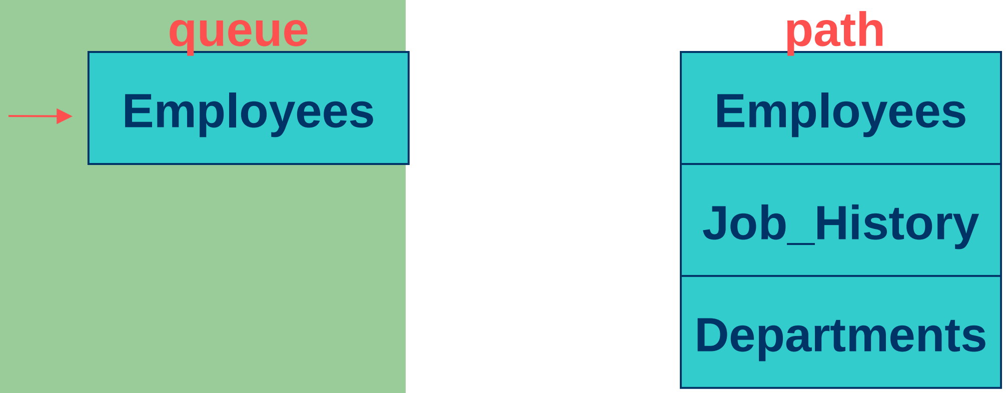
Join Base Tables

Employees	Job_History	Jobs	Departments	Locations	Countries

Join Graph

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
```


queue

Employees

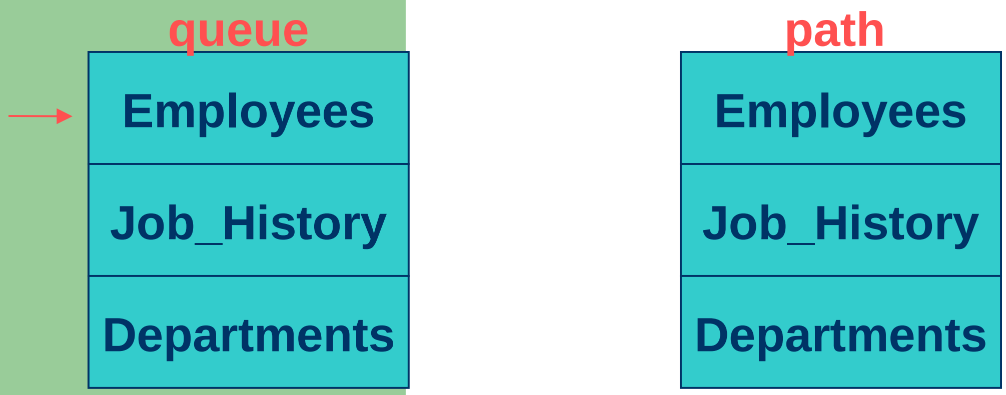
path

Employees

Job_History

Departments

```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0$…$T_m$ 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 $T_0$…$T_m$ 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

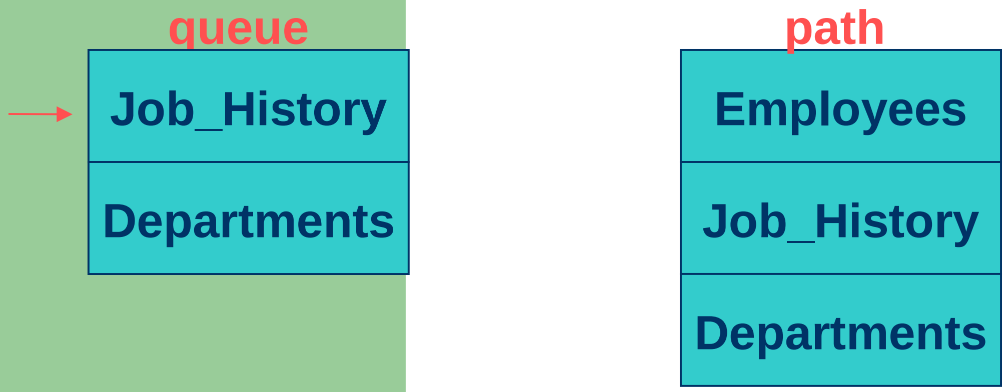
path

Employees

Job_History

Departments

```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0$…$T_m$ 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

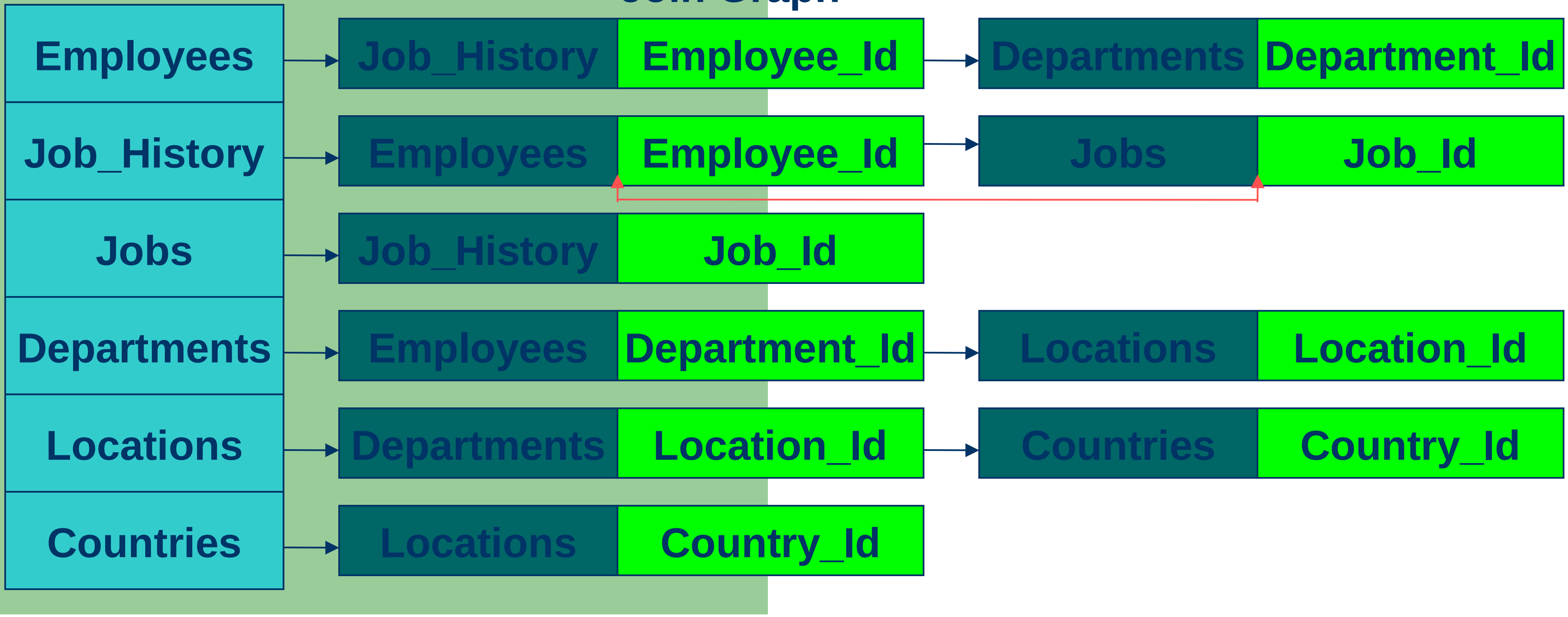
Employees

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
```

Join Graph



Employees → Job_History | Employee_Id → Departments | Department_Id

Job_History → Employees | Employee_Id → Jobs | Job_Id

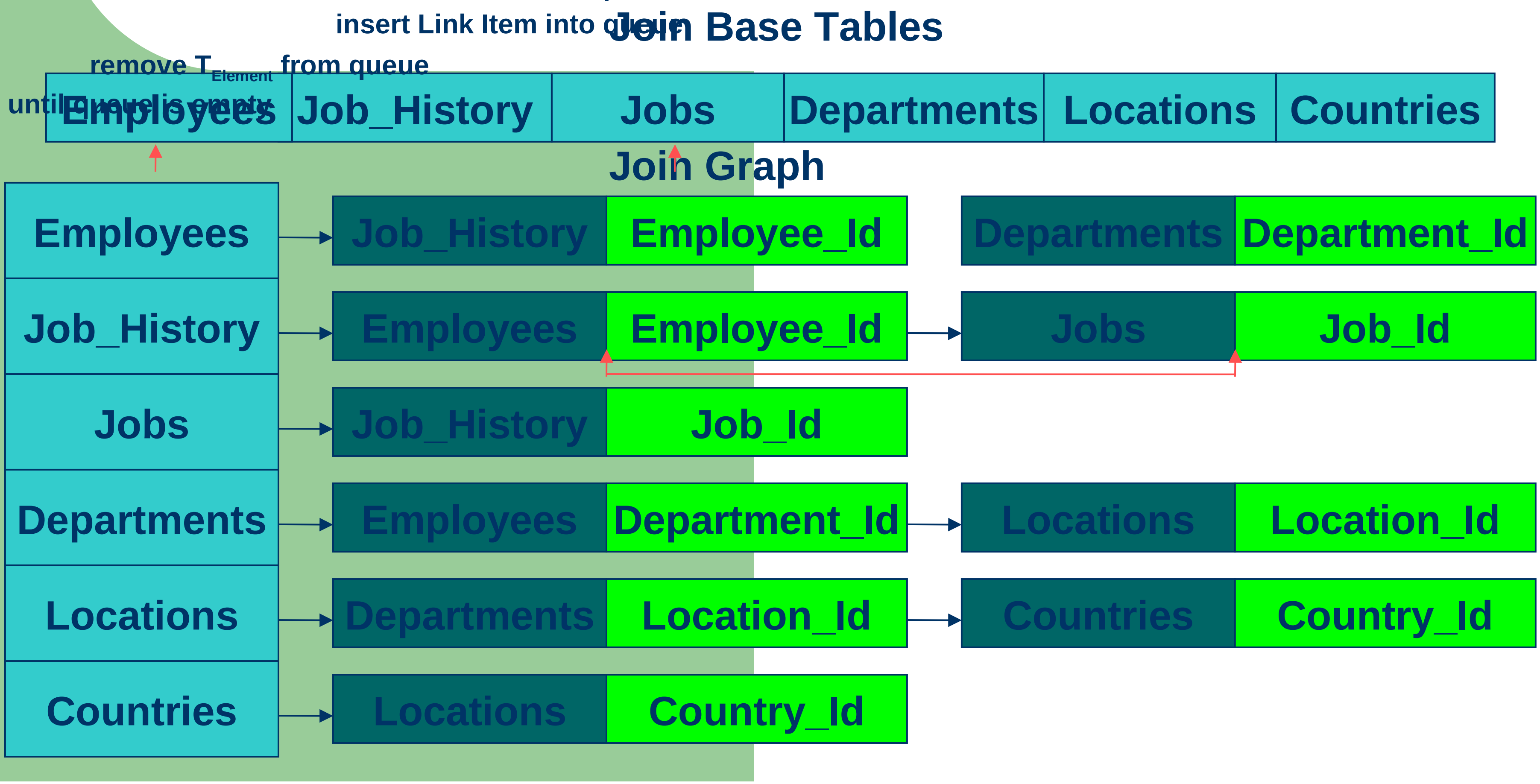
Jobs → Job_History | Job_Id

Departments → Employees | Department_Id → Locations | Location_Id

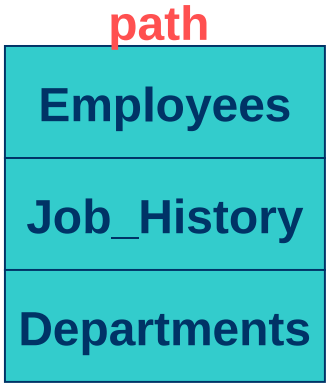
Locations → Departments | Location_Id → Countries | Country_Id

Countries → Locations | Country_Id

```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0…T_m$ 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
```



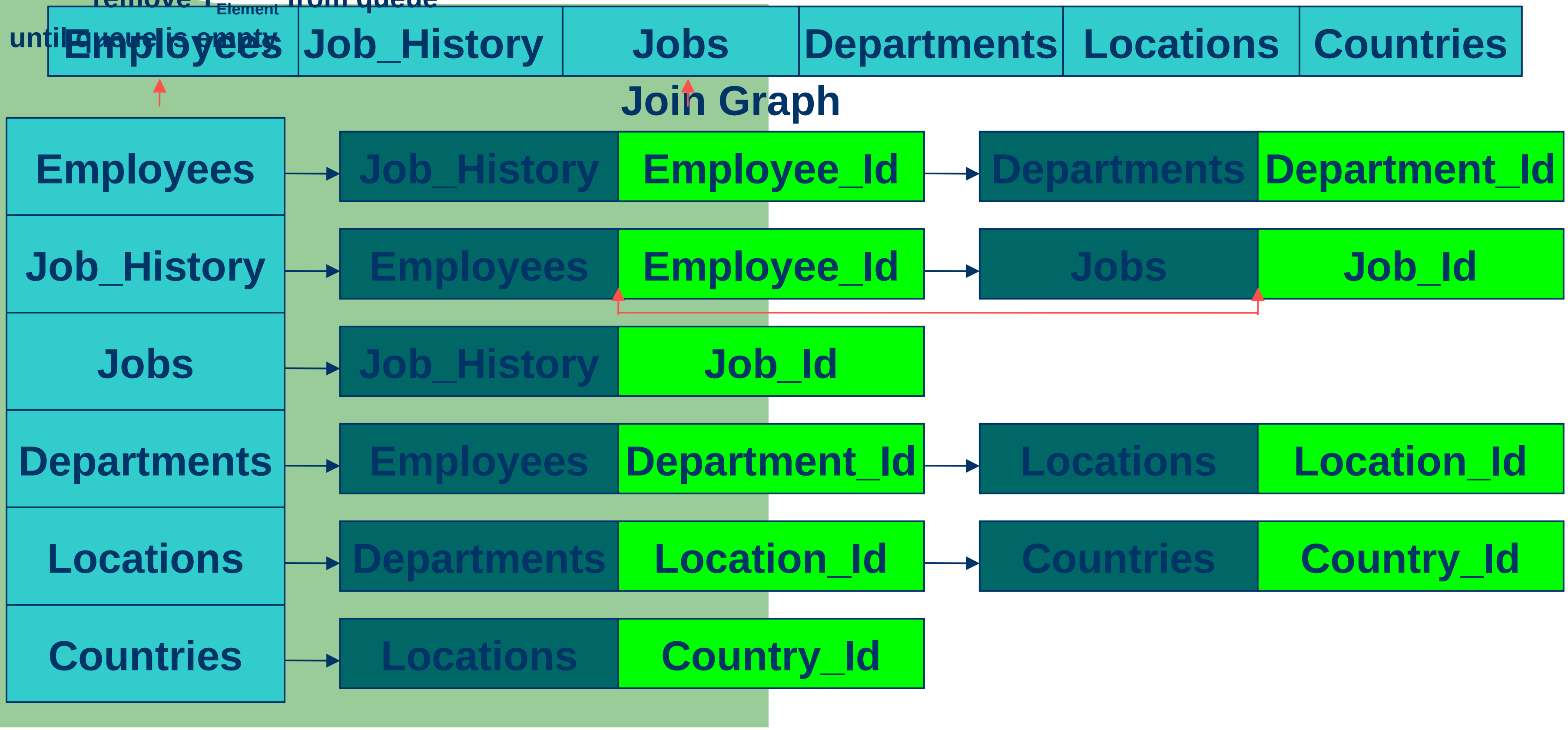
```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert $T_0$ into path
insert $T_0$ into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ 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
```



path

Employees
Job_History
Departments

Join Base Tables

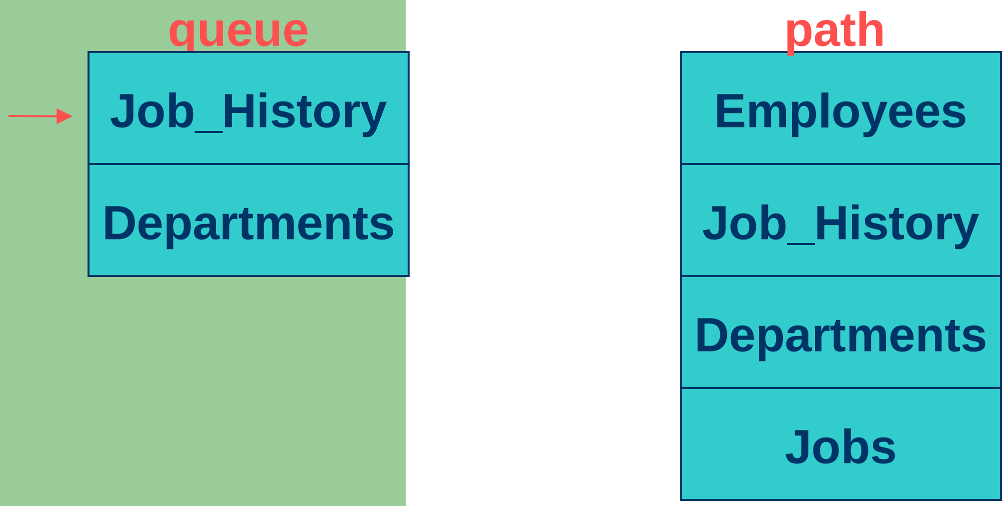


Employees	Job_History	Jobs	Departments	Locations	Countries

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
```



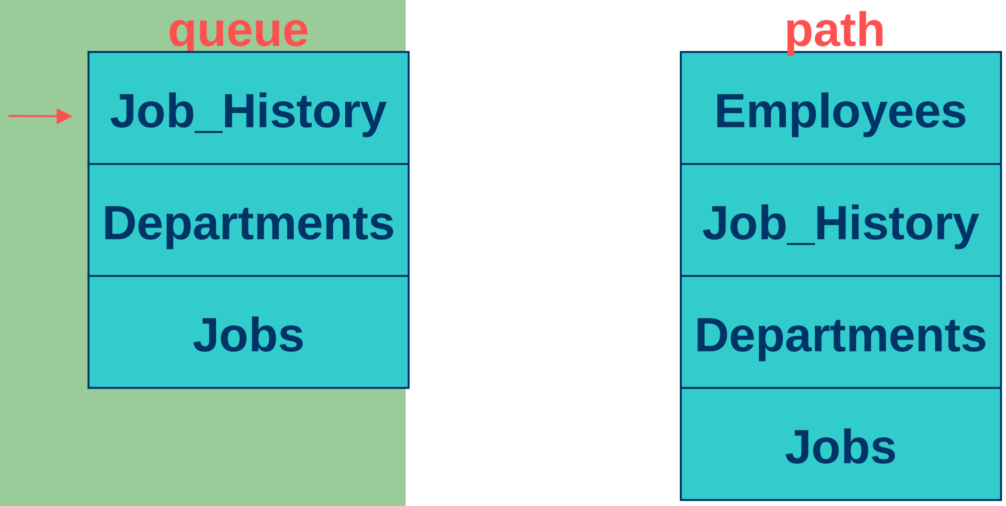
queue

Job_History
Departments

path

Employees
Job_History
Departments
Jobs

```
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

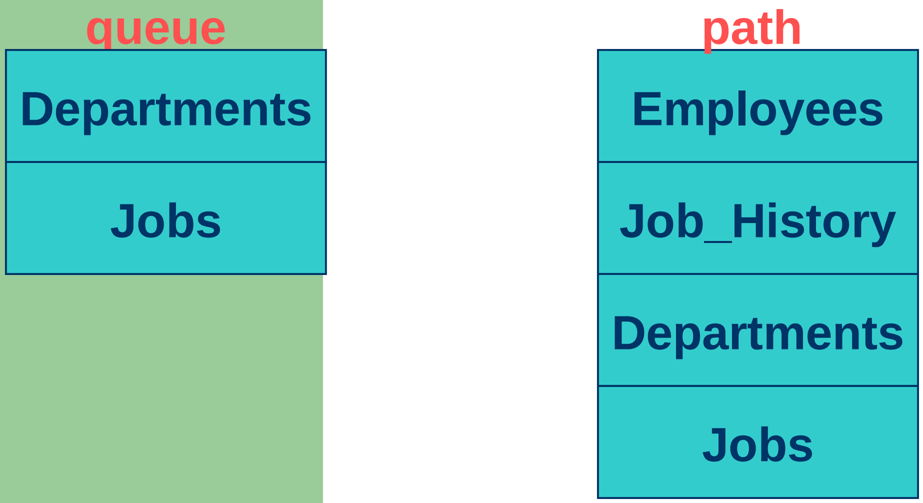
Job_History
Departments
Jobs

path

Employees
Job_History
Departments
Jobs

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

Departments

Jobs

path

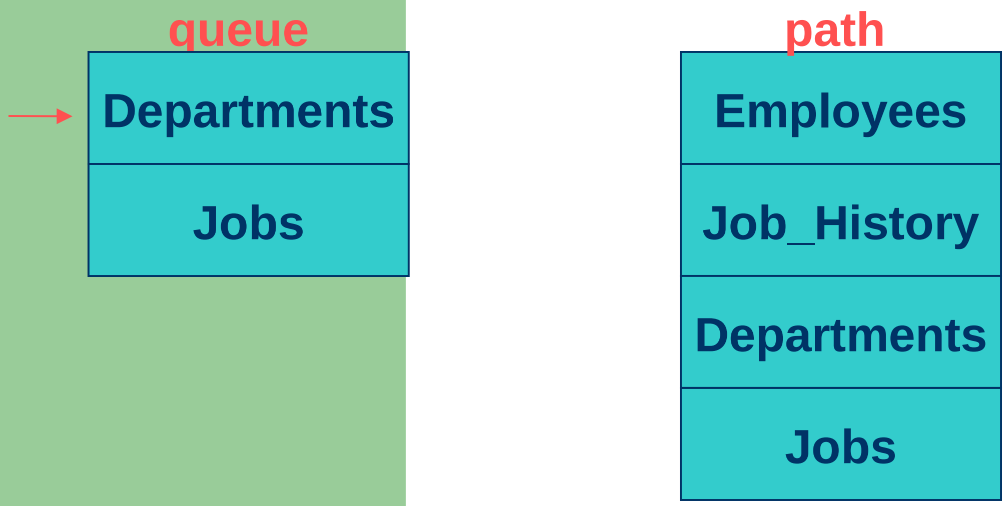
Employees

Job_History

Departments

Jobs

```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0$…$T_m$ 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

Departments

Jobs

path

Employees

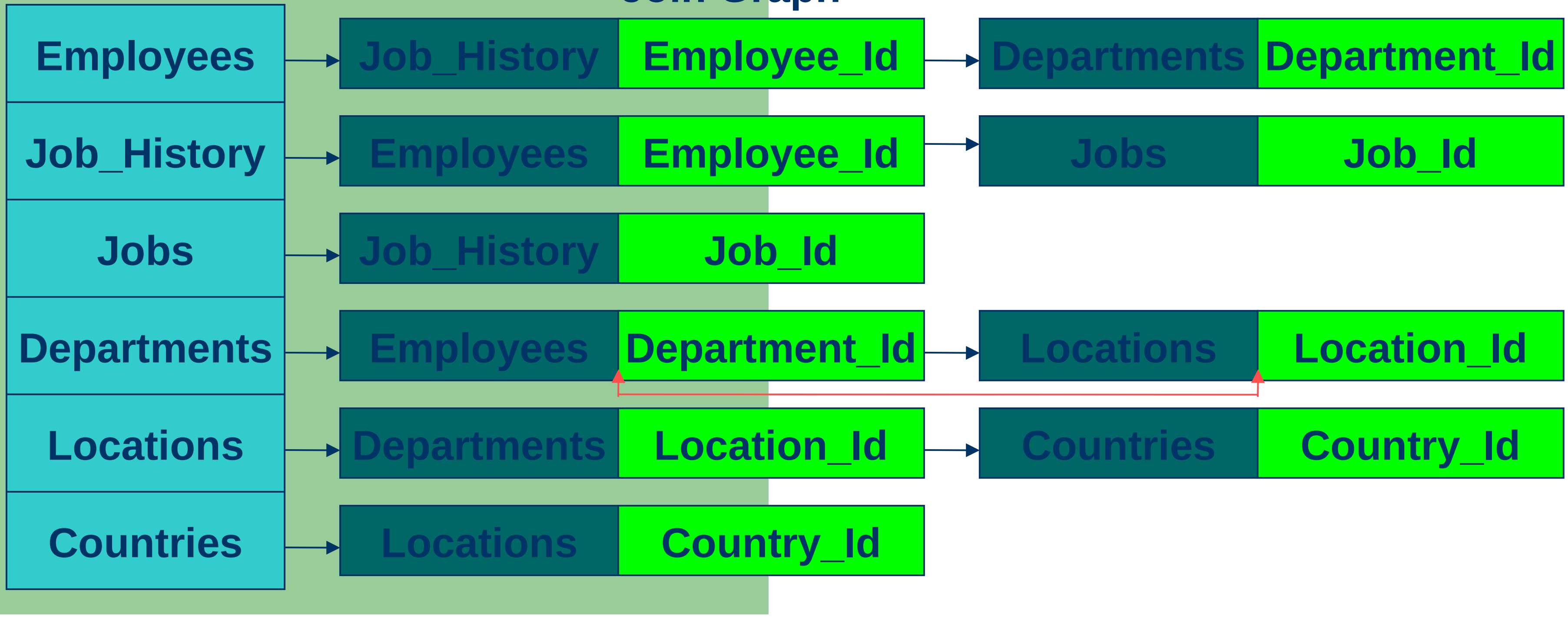
Job_History

Departments

Jobs

```
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



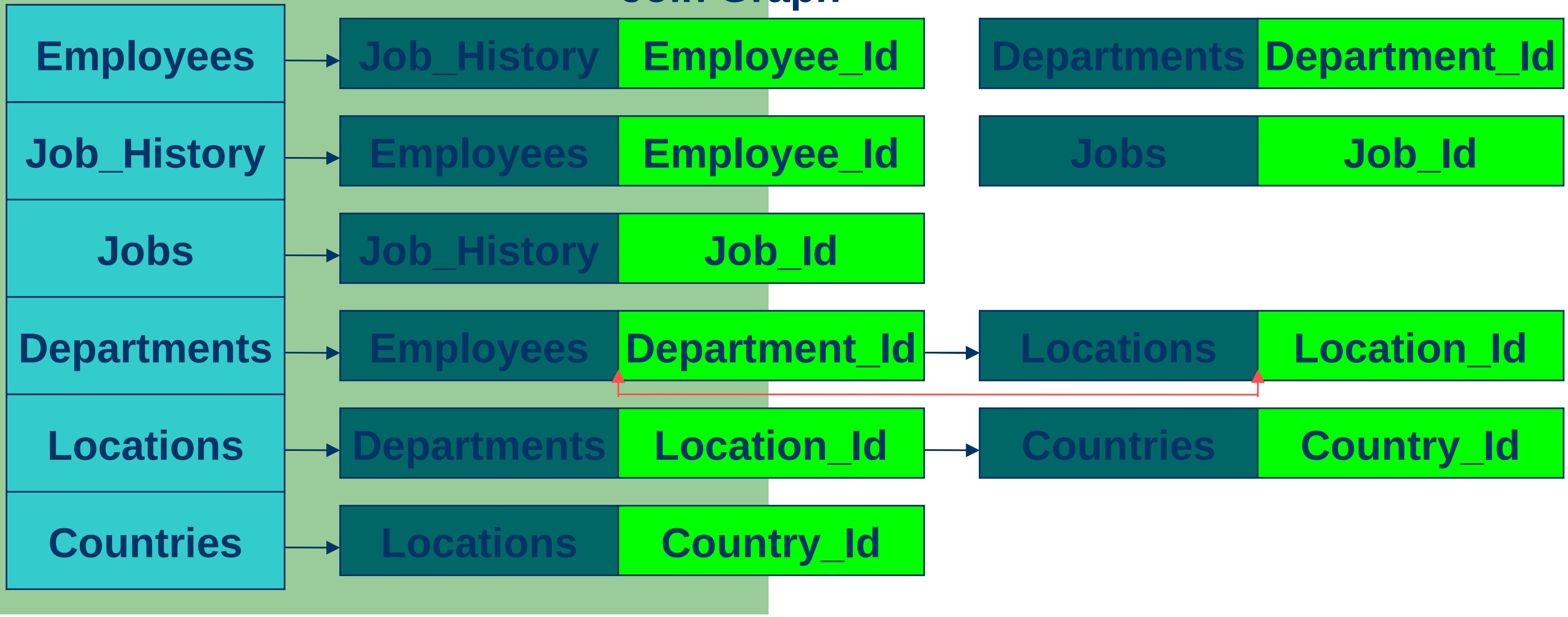
Employees	Job_History	Employee_Id	Departments	Department_Id
Job_History	Employees	Employee_Id	Jobs	Job_Id
Jobs	Job_History	Job_Id		
Departments	Employees	Department_Id	Locations	Location_Id
Locations	Departments	Location_Id	Countries	Country_Id
Countries	Locations	Country_Id		

```
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
```

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 $T_0$…$T_m$ 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
```

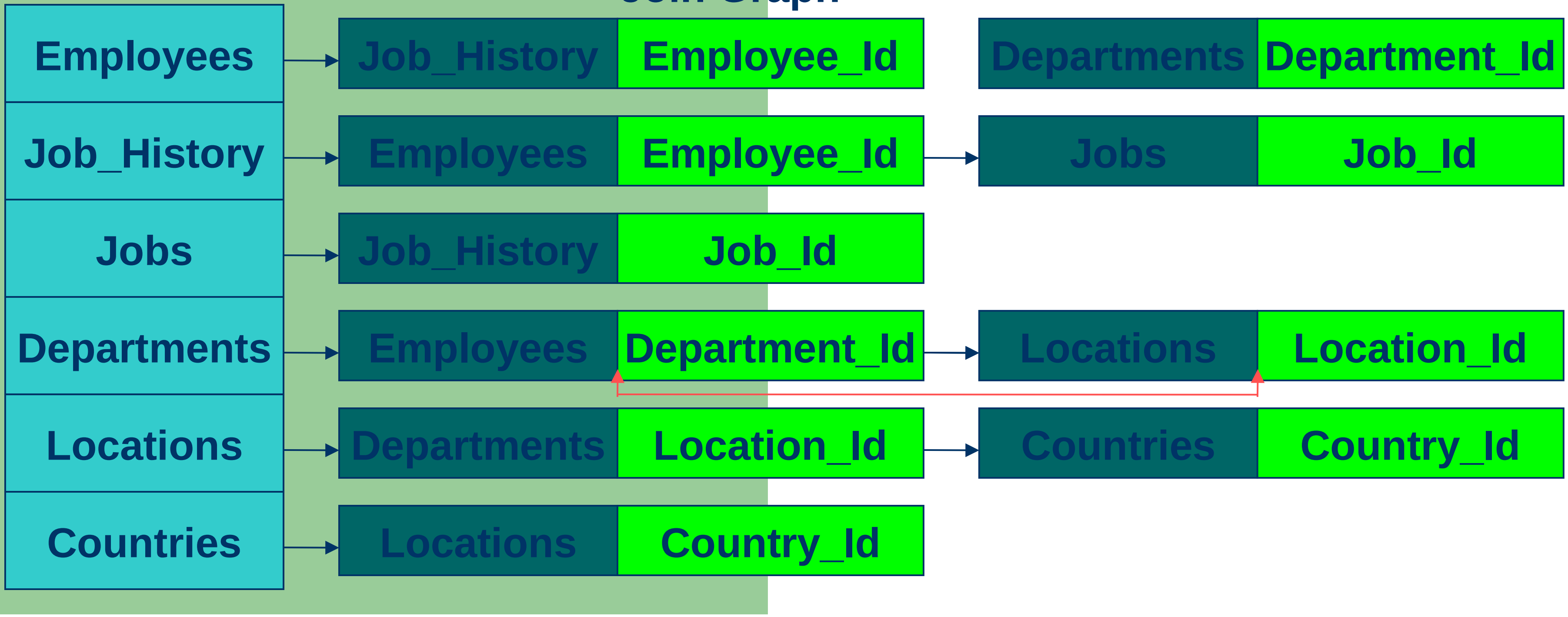
path

Employees	Jobs
Job_History	
Departments	

Join Base Tables

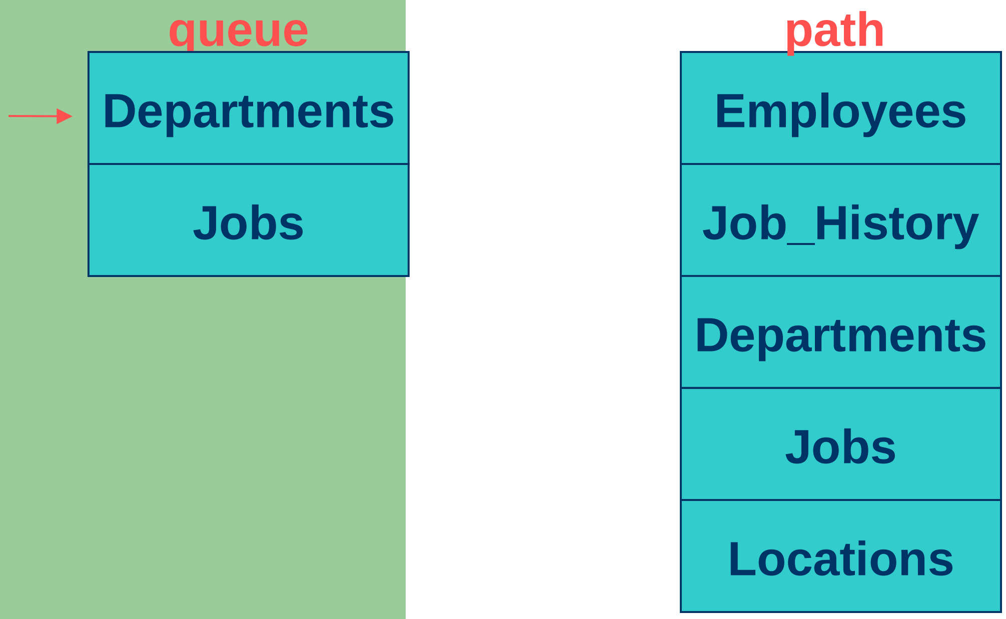
Employees	Job_History	Jobs	Departments	Locations	Countries

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		

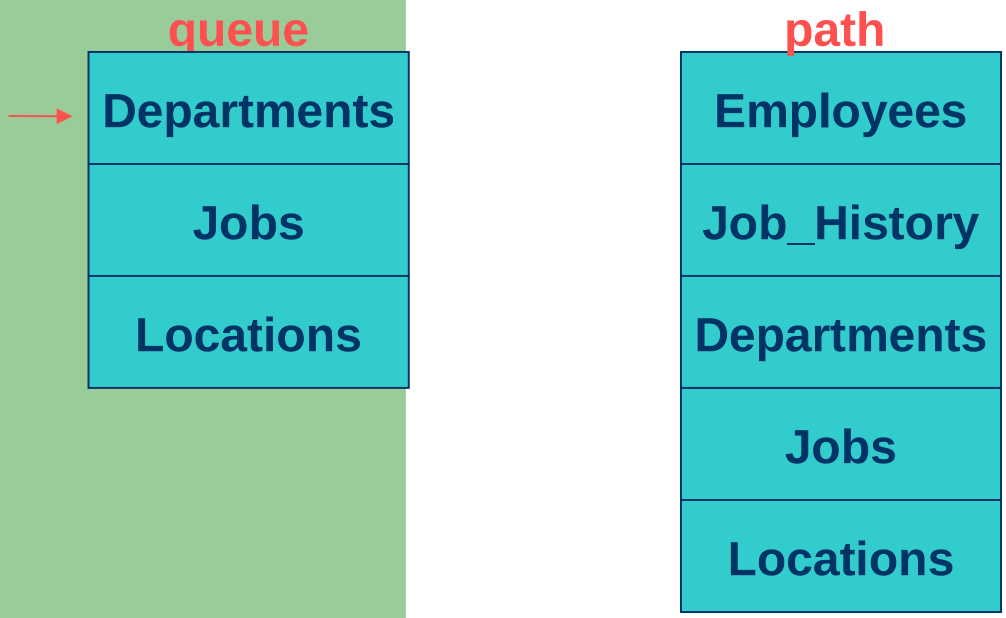
```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0$…$T_m$ be the base tables
create 2 dynamic arrays queue and path
insert $T_0$ into path
insert $T_0$ into queue
repeat
          $T_{Element}$ = First Table in queue
          for every Link Item in Adjacent Link of $T_{Element}$ 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
Departments
Jobs

path
Employees
Job_History
Departments
Jobs
Locations

```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0$…$T_m$ 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

Departments

Jobs

Locations

path

Employees

Job_History

Departments

Jobs

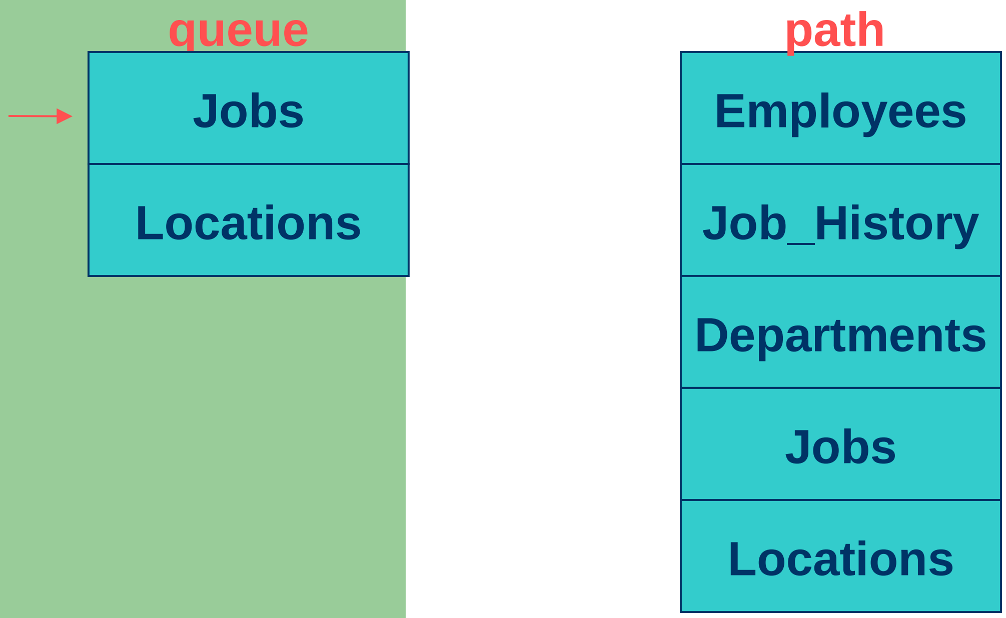
Locations

```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0…T_m$ 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
Jobs
Locations

path
Employees
Job_History
Departments
Jobs
Locations

```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0…T_m$ 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

Jobs

Locations

path

Employees

Job_History

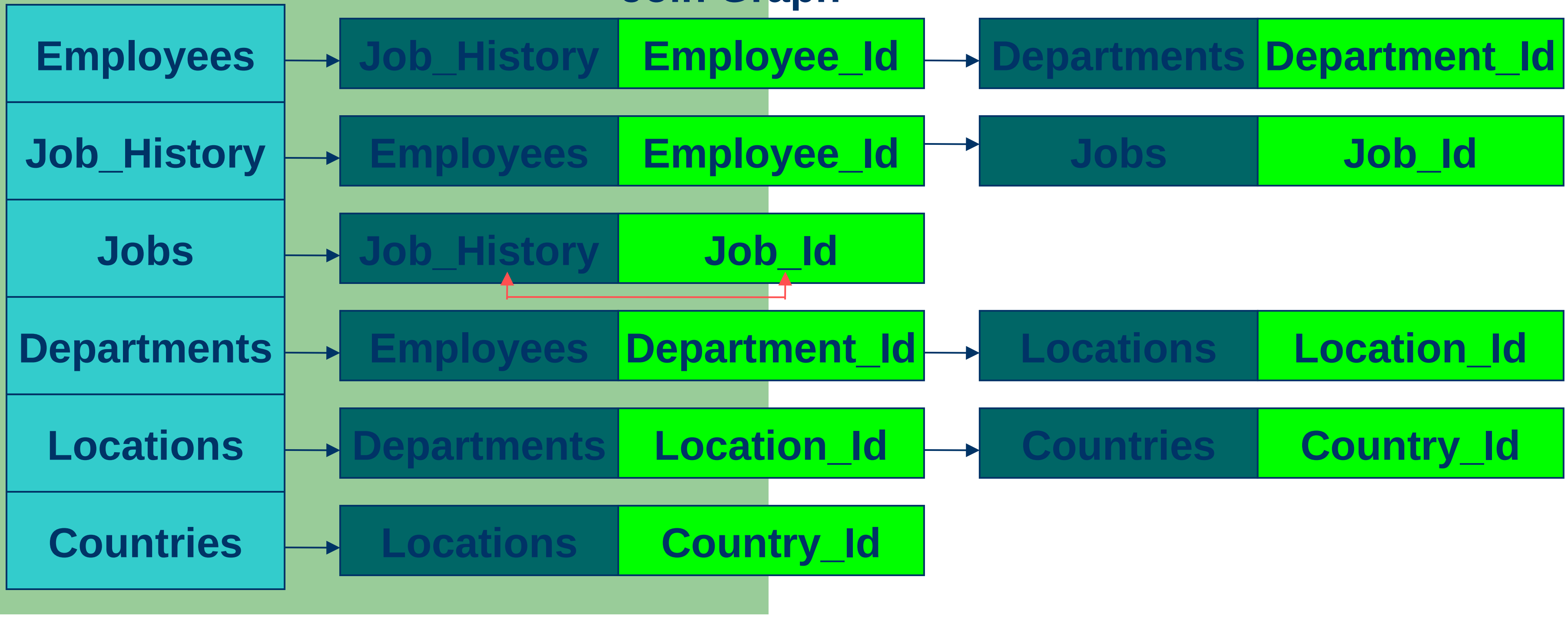
Departments

Jobs

Locations

```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0$…$T_m$ 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



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

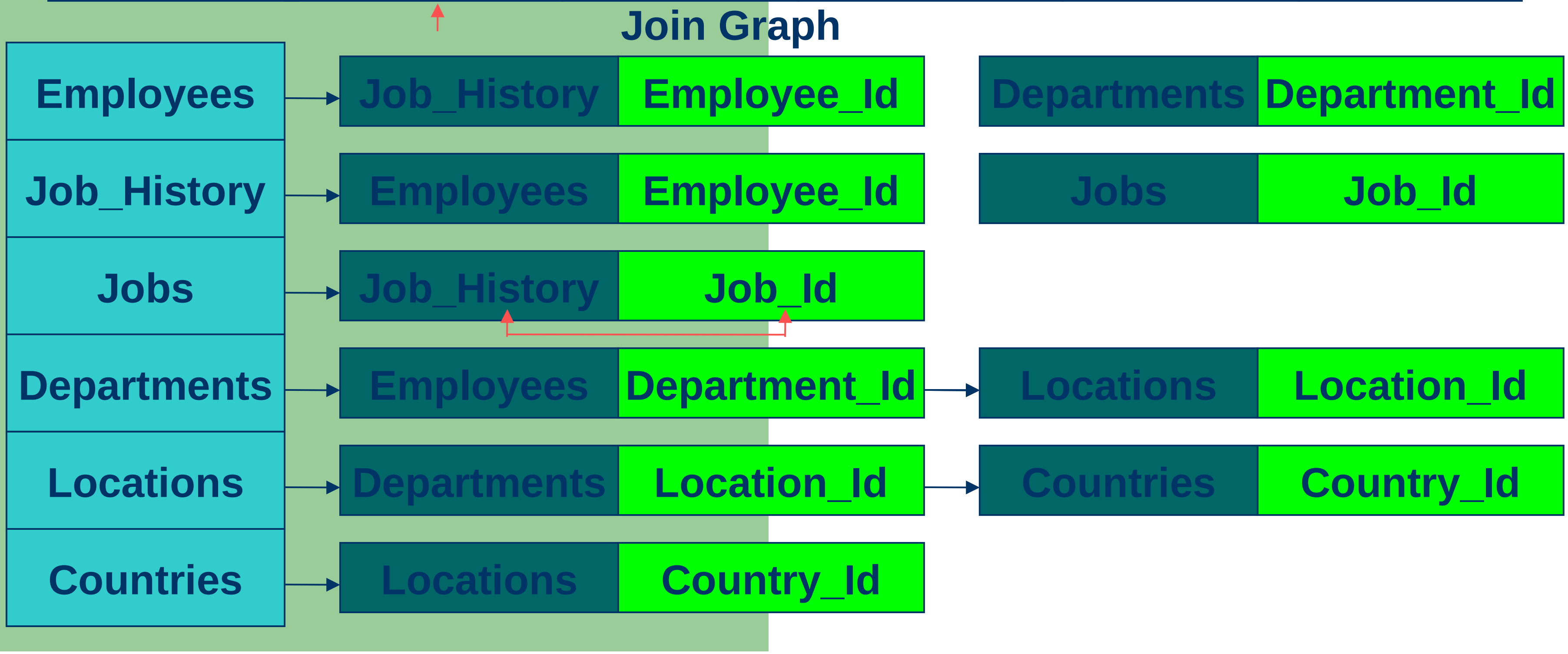
Countries → Locations | Country_Id

```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0…T_m$ 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



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		

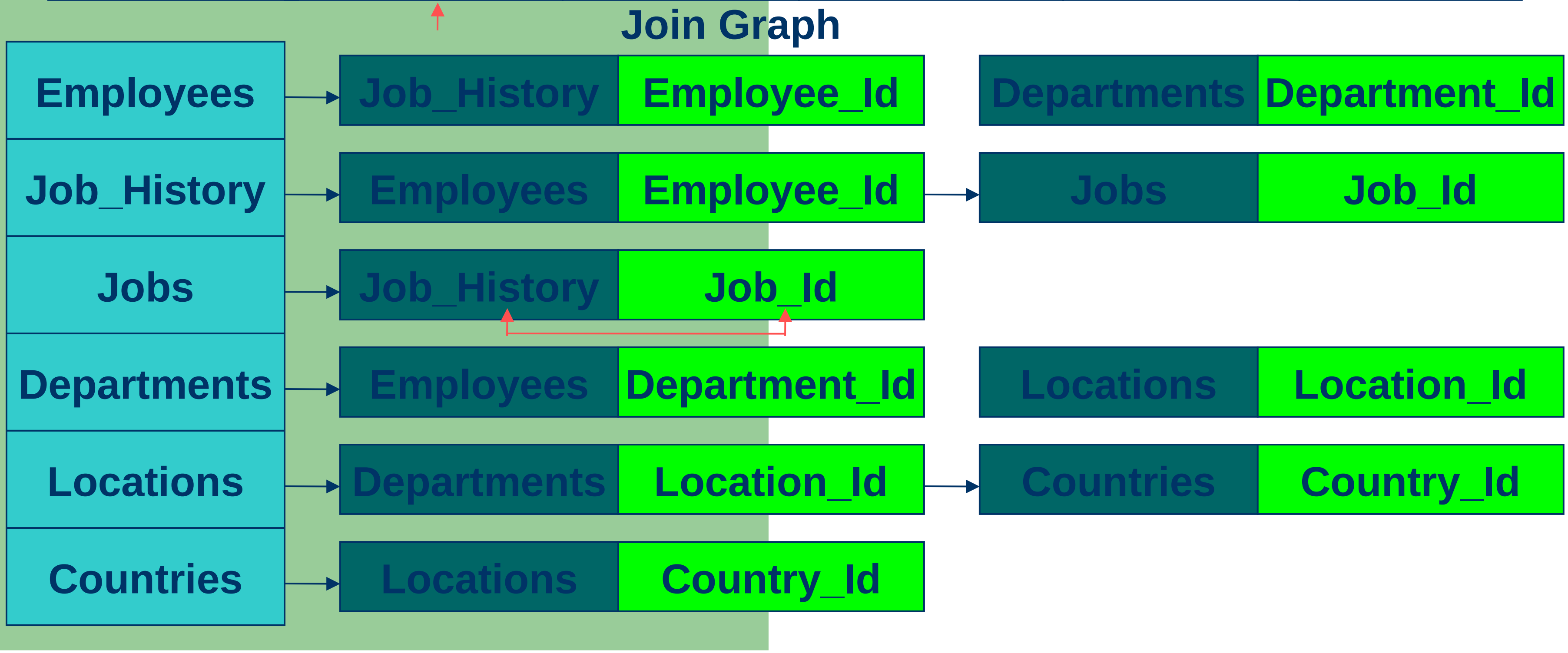
```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0$…$T_m$ be the base tables
create 2 dynamic arrays queue and path
insert $T_0$ into path
insert $T_0$ into queue
repeat
          $T_{Element}$ = First Table in queue
          for every Link Item in Adjacent Link of $T_{Element}$ 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

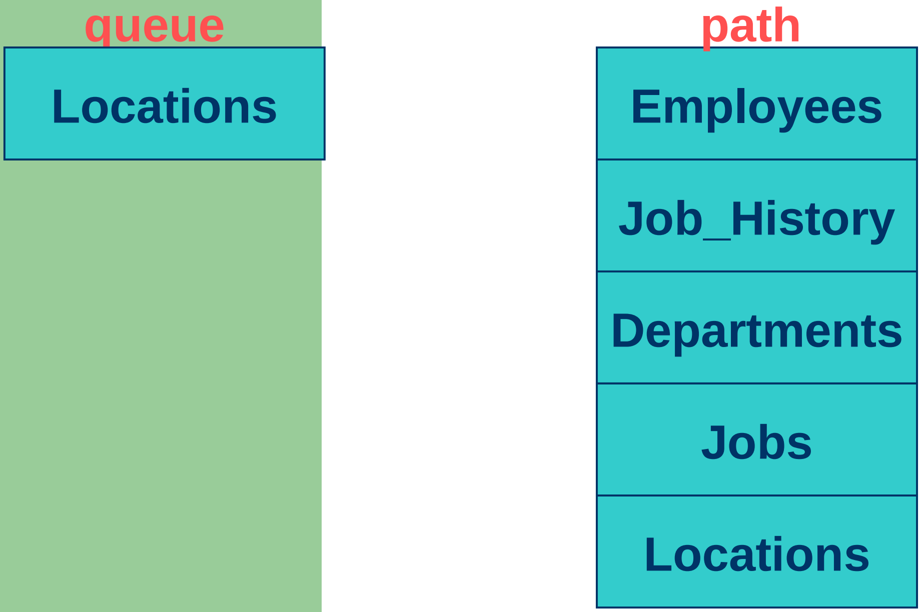
Employees	Job_History	Jobs	Departments	Locations	Countries



Join Graph

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 $T_0…T_m$ 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

Locations

path

Employees

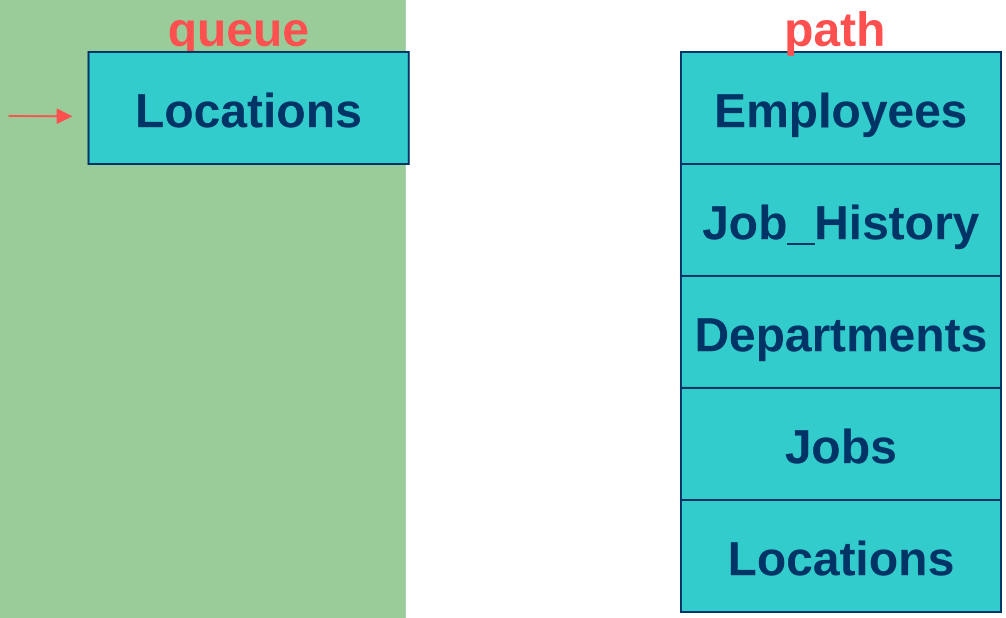
Job_History

Departments

Jobs

Locations

```
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

Locations

path

Employees

Job_History

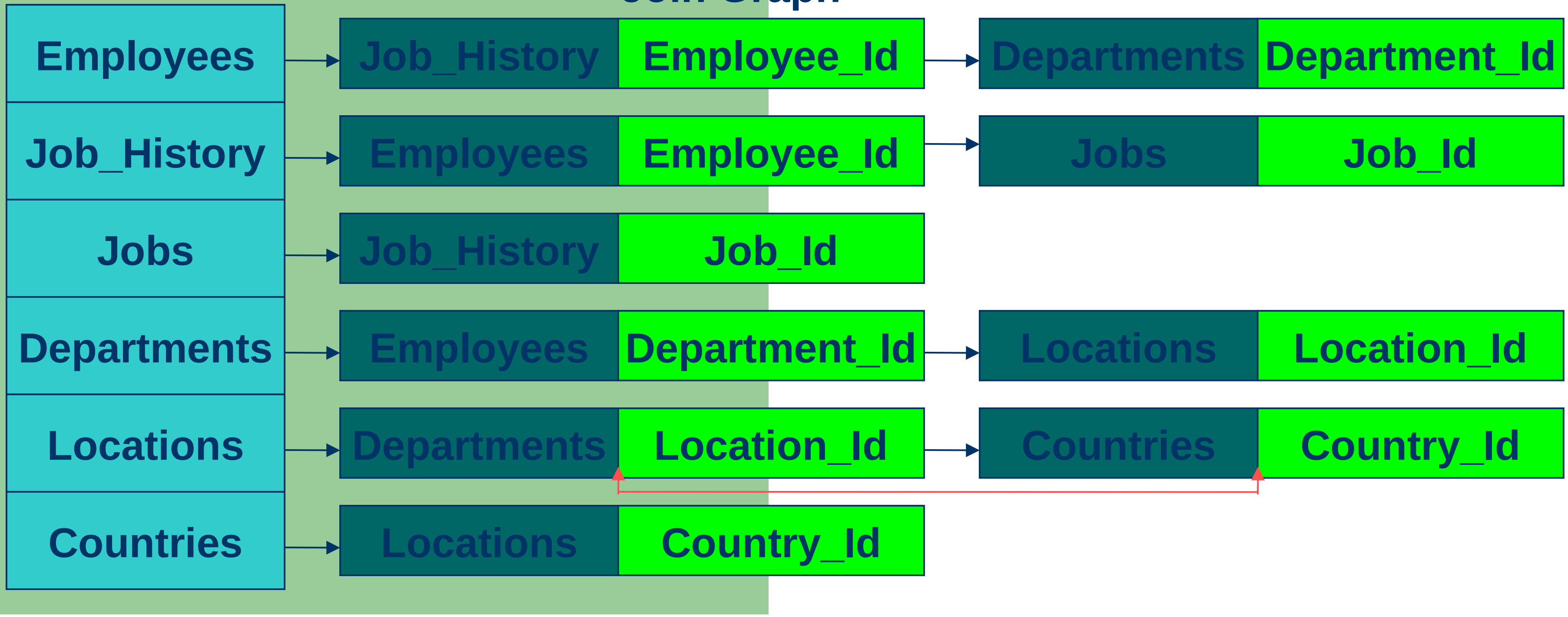
Departments

Jobs

Locations

```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0$…$T_m$ 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



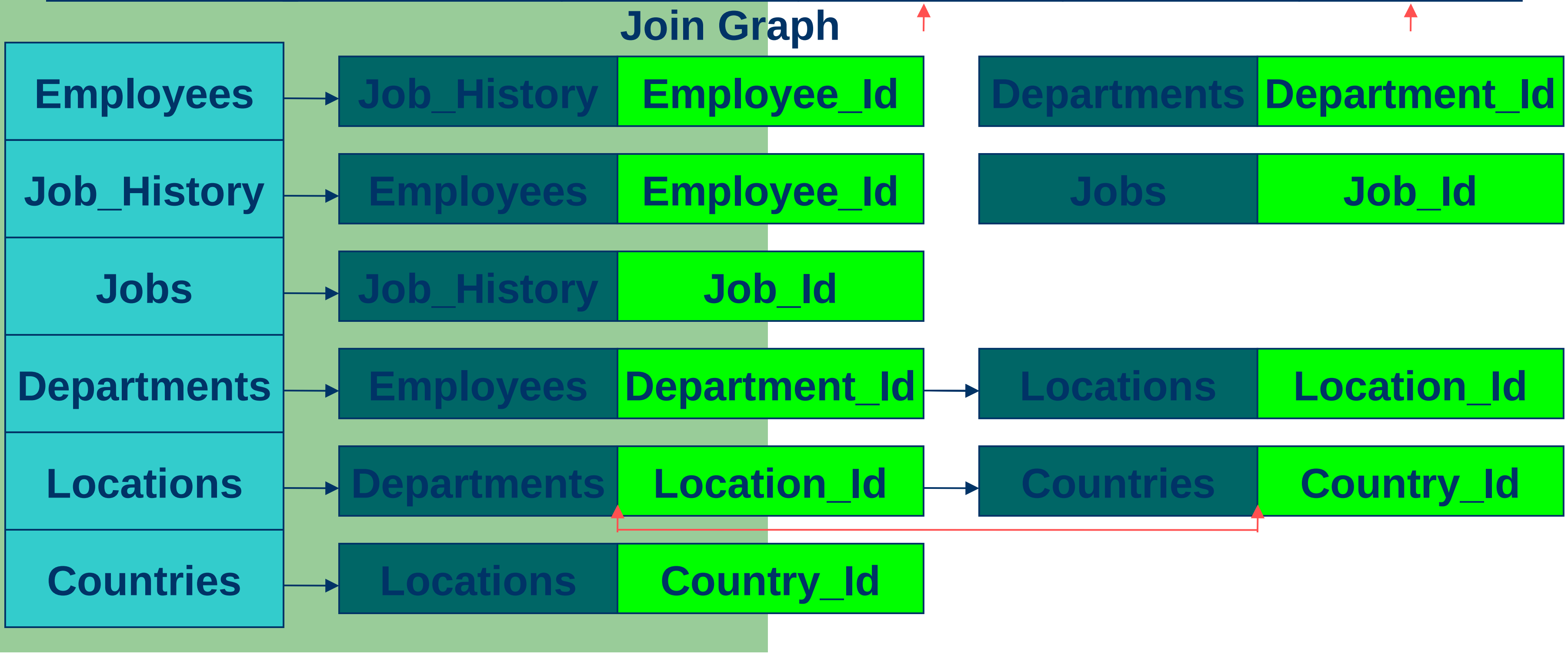
Employees	Job_History	Employee_Id	Departments	Department_Id
Job_History	Employees	Employee_Id	Jobs	Job_Id
Jobs	Job_History	Job_Id		
Departments	Employees	Department_Id	Locations	Location_Id
Locations	Departments	Location_Id	Countries	Country_Id
Countries	Locations	Country_Id		

```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0$…$T_m$ 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



Employees → Job_History Employee_Id   Departments Department_Id

Job_History → Employees Employee_Id   Jobs Job_Id

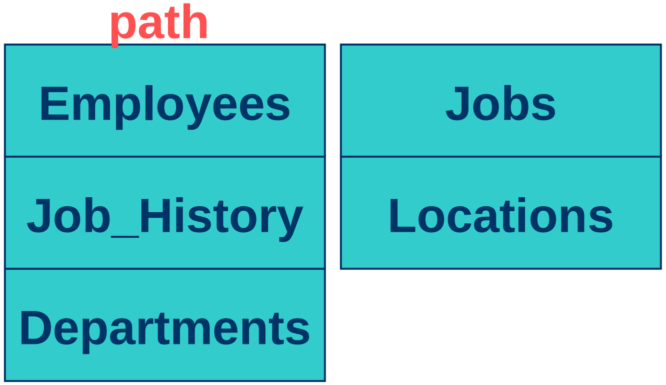
Jobs → Job_History Job_Id

Departments → Employees Department_Id → Locations Location_Id

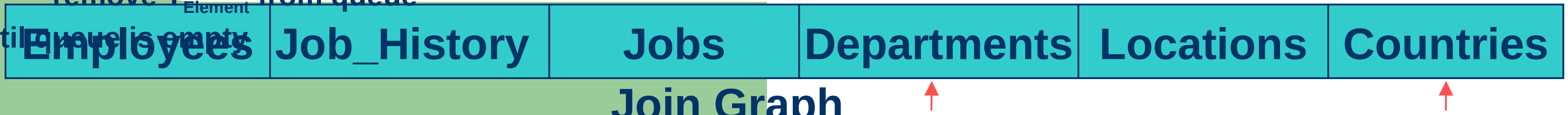
Locations → Departments Location_Id → Countries Country_Id

Countries → Locations Country_Id

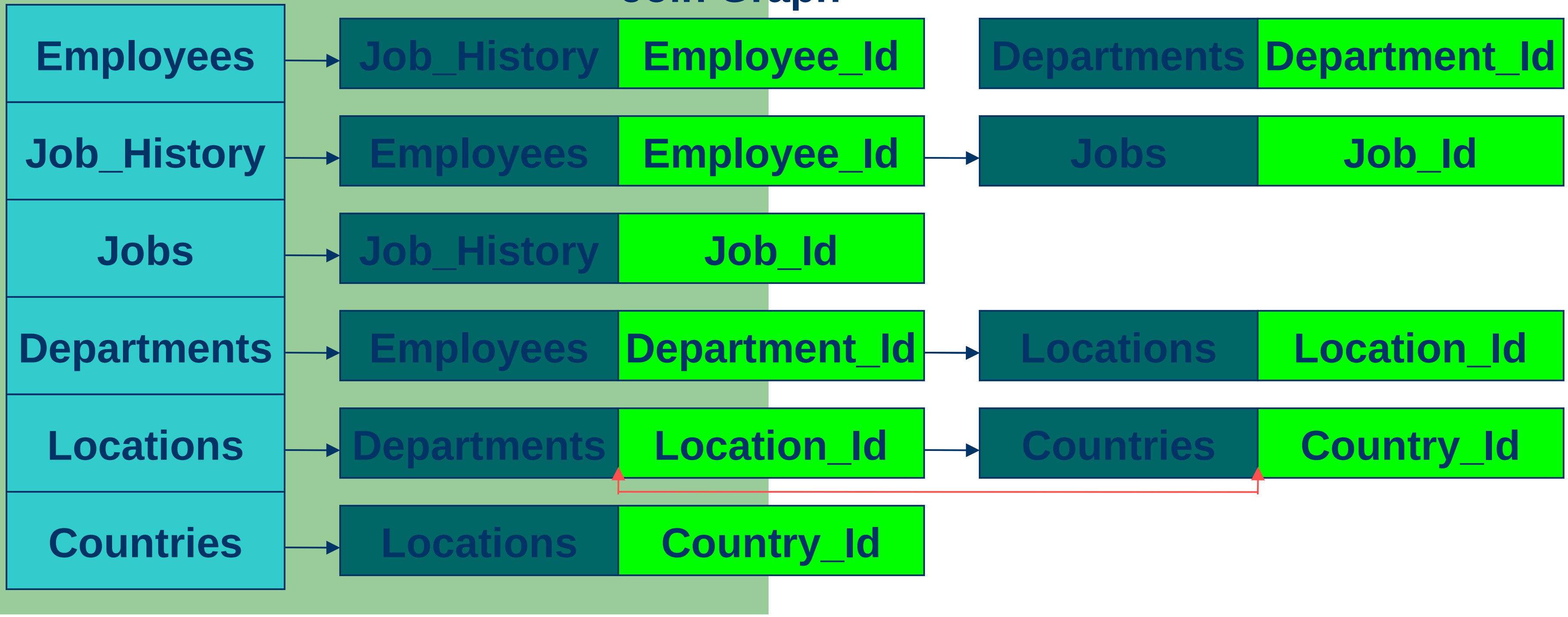
```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0 \ldots T_m$ be the base tables
create 2 dynamic arrays queue and path
insert $T_0$ into path
insert $T_0$ into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ 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
```



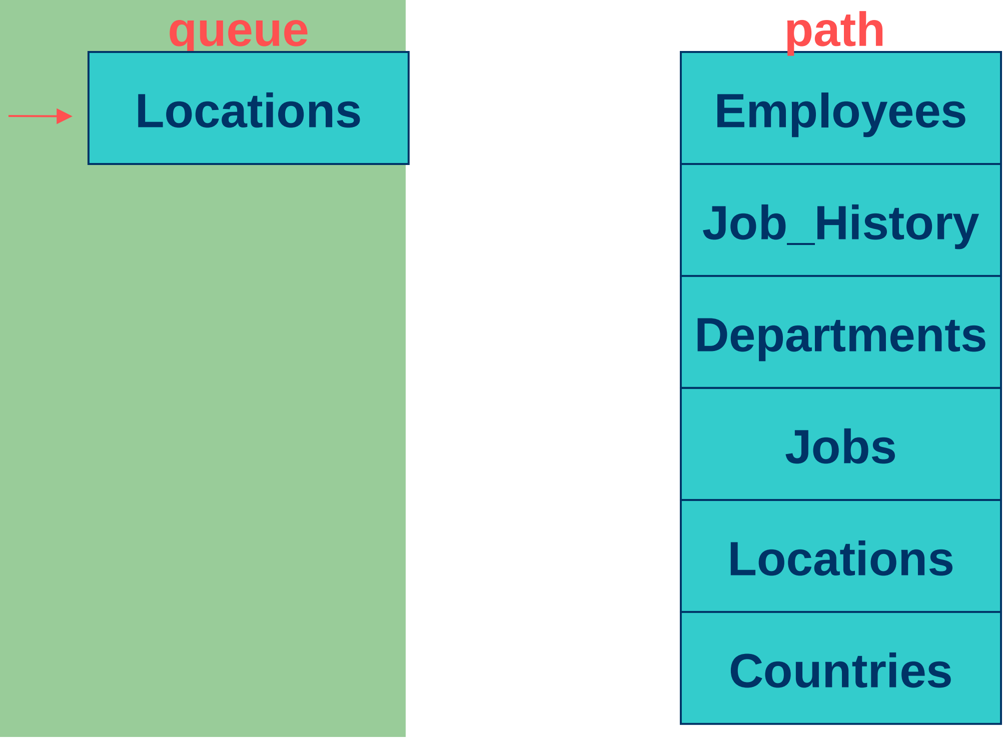
Join Base Tables



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
```



queue

Locations

path

Employees

Job_History

Departments

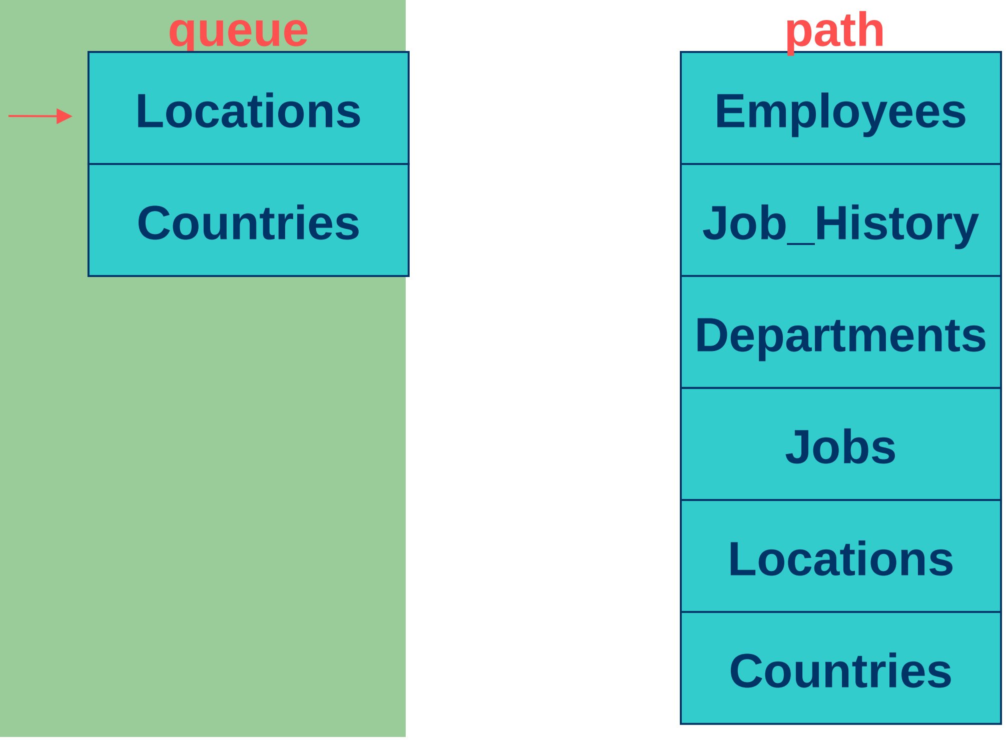
Jobs

Locations

Countries

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

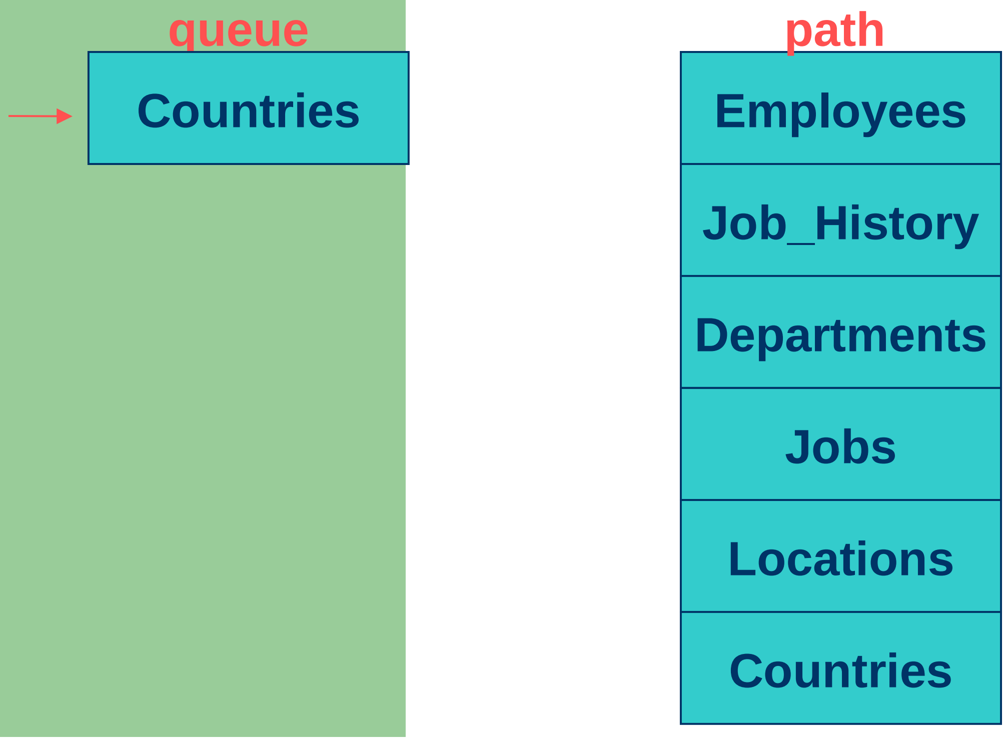
```
let $T_0$…$T_m$ be the base tables
create 2 dynamic arrays queue and path
insert $T_0$ into path
insert $T_0$ into queue
repeat
    $T_{Element}$ = First Table in queue
    for every Link Item in Adjacent Link of $T_{Element}$ from the Join Graph do
        if the Link Item is in the join sequence 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	path
Countries	Employees
	Job_History
	Departments
	Jobs
	Locations
	Countries

```
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

Countries

path

Employees

Job_History

Departments

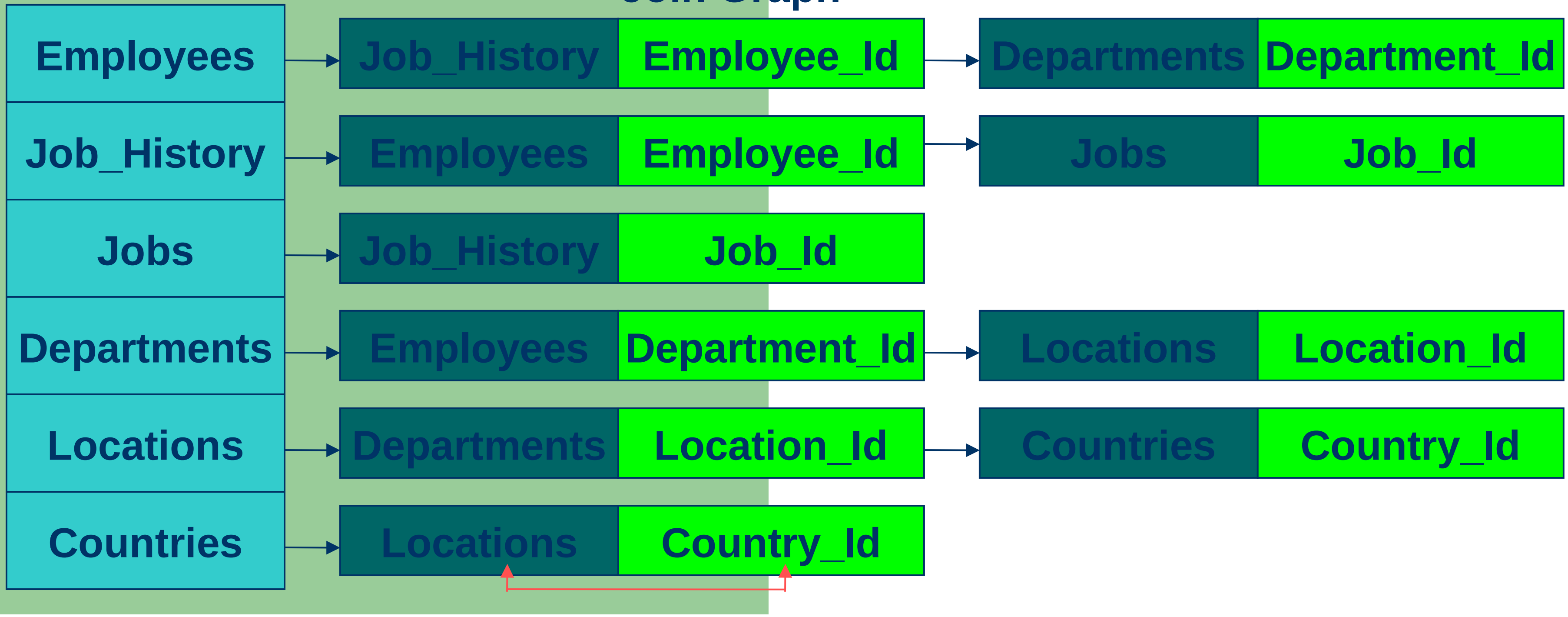
Jobs

Locations

Countries

```
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



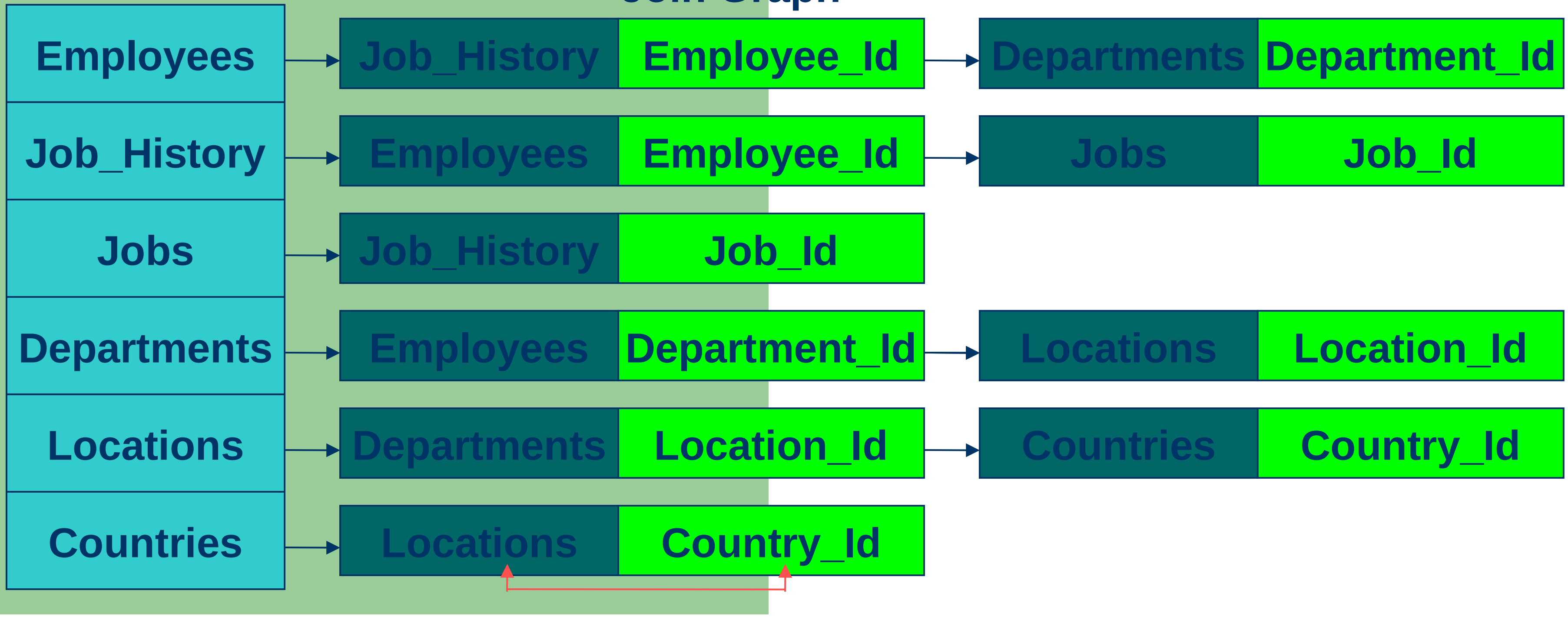
Employees	Job_History	Employee_Id	Departments	Department_Id
Job_History	Employees	Employee_Id	Jobs	Job_Id
Jobs	Job_History	Job_Id		
Departments	Employees	Department_Id	Locations	Location_Id
Locations	Departments	Location_Id	Countries	Country_Id
Countries	Locations	Country_Id		

```
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
```

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 $T_0…T_m$ 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
```

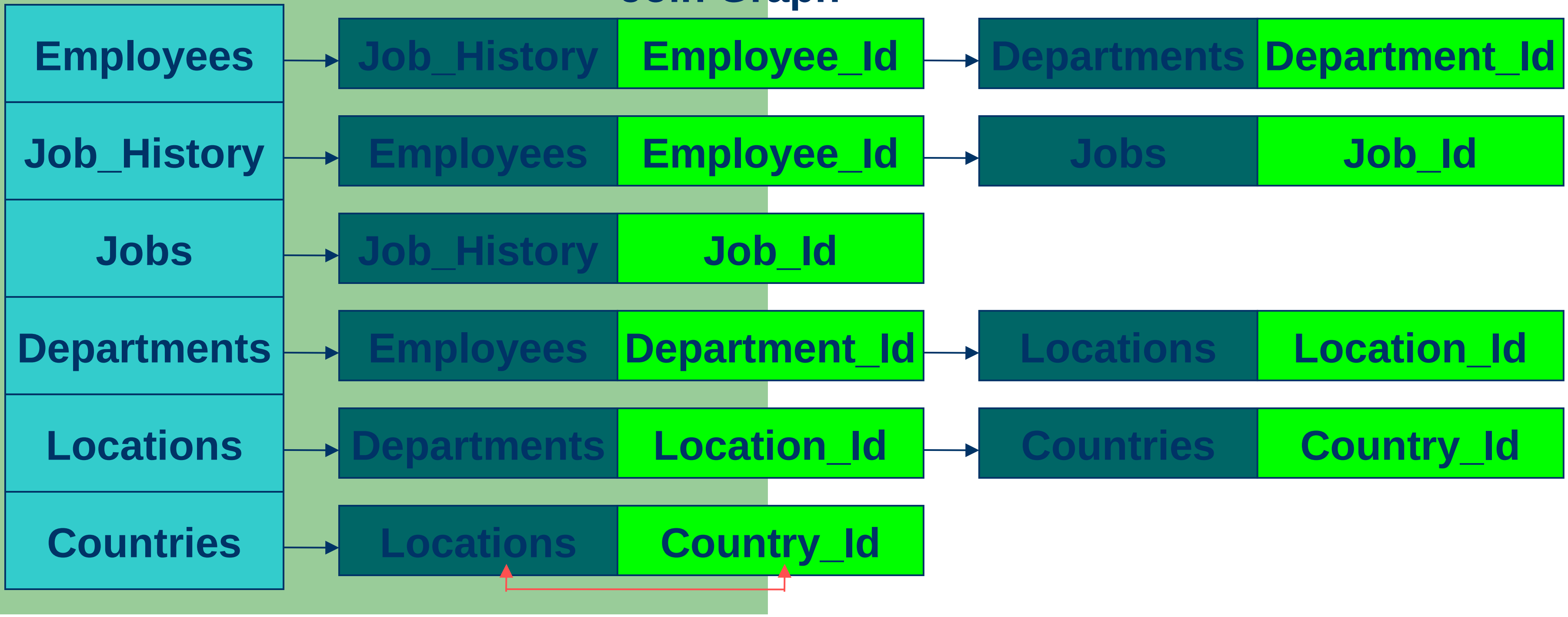
path

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 $T_0$…$T_m$ 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
Job_History
Departments
Jobs
Locations
Countries

```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0…T_m$ 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

path
Employees
Job_History
Departments
Jobs
Locations
Countries

vertexes

Employees
Job_History
Deparments
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 $NodesList[T_{[buf]} ] = T_{[buf]}$
```

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

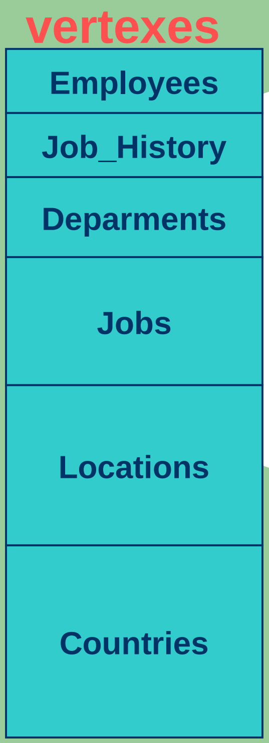
buf

- 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

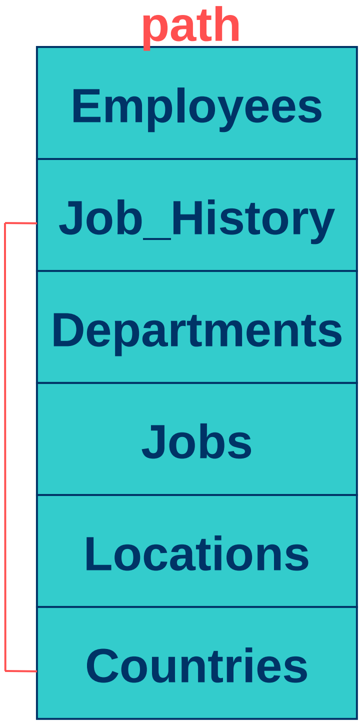
Jobs

Locations

Countries

buf

Employees



path

Employees

Job_History

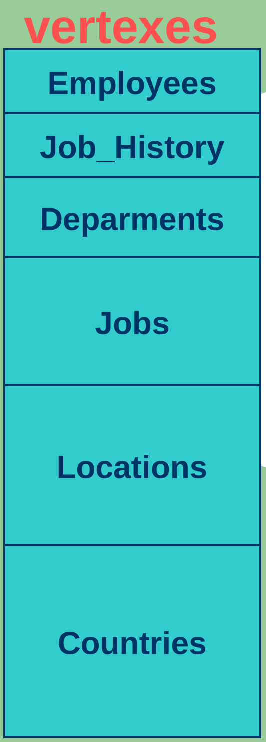
Departments

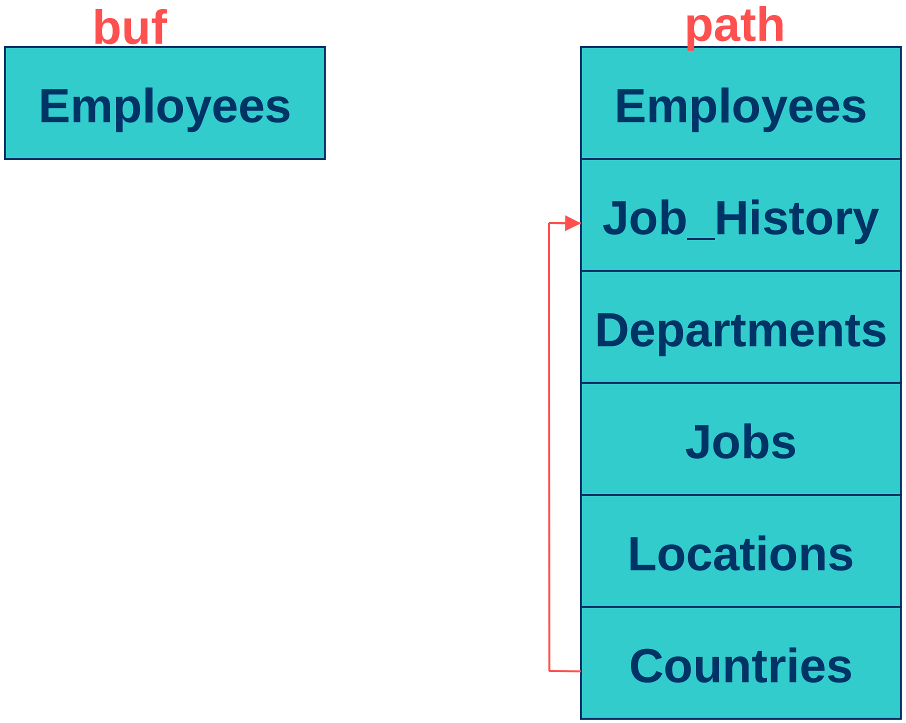
Jobs

Locations

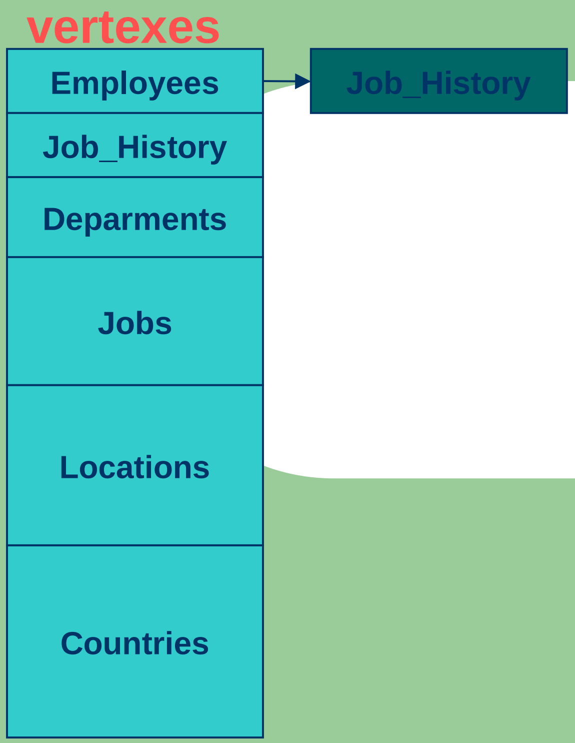
Countries

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





```
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

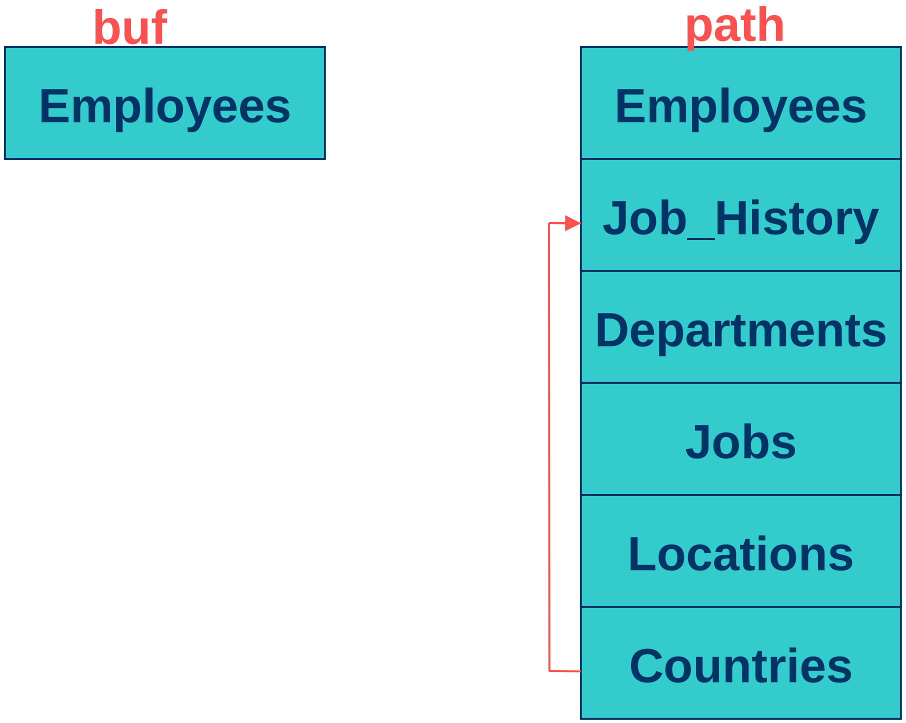
Deparments

Jobs

Locations

Countries

Job_History



buf

Employees

path

Employees

Job_History

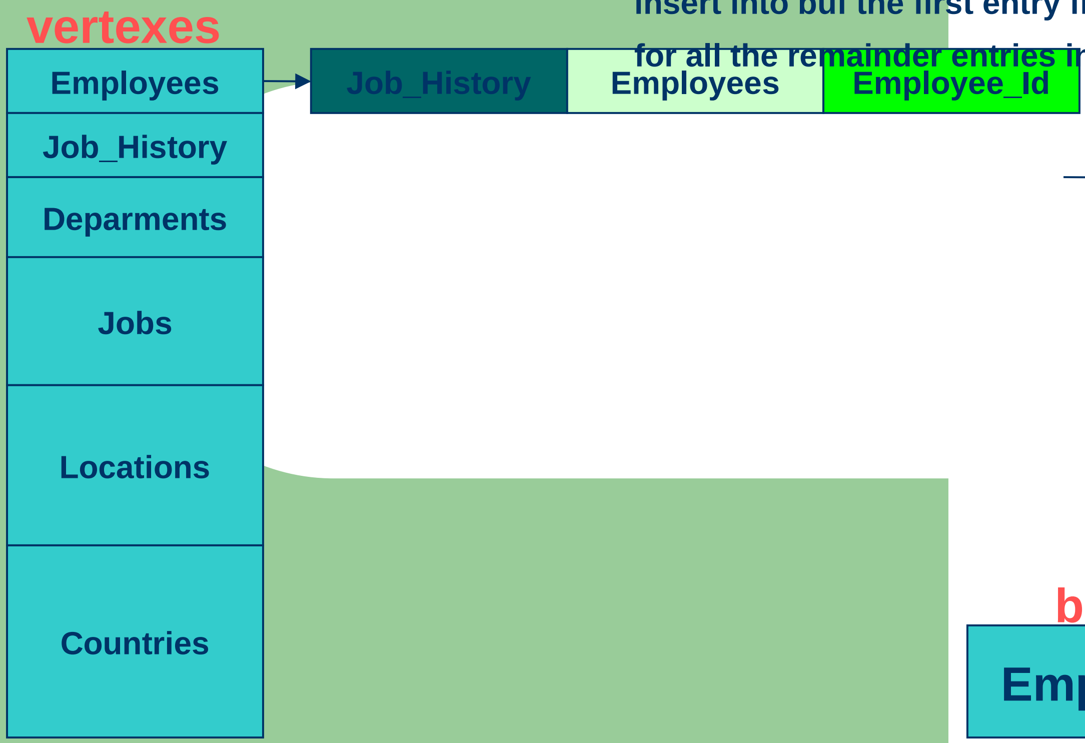
Departments

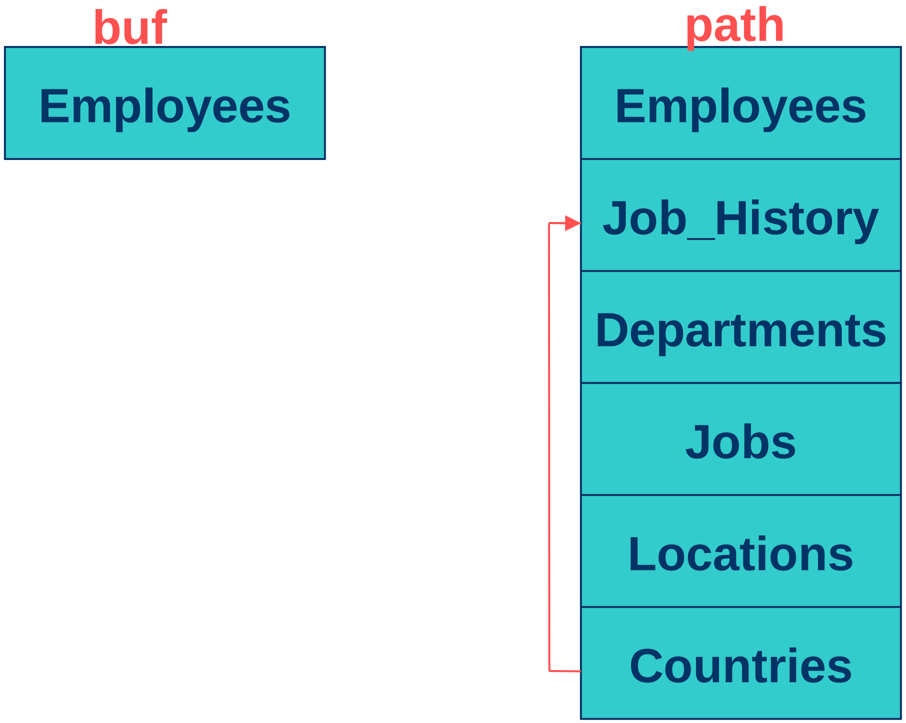
Jobs

Locations

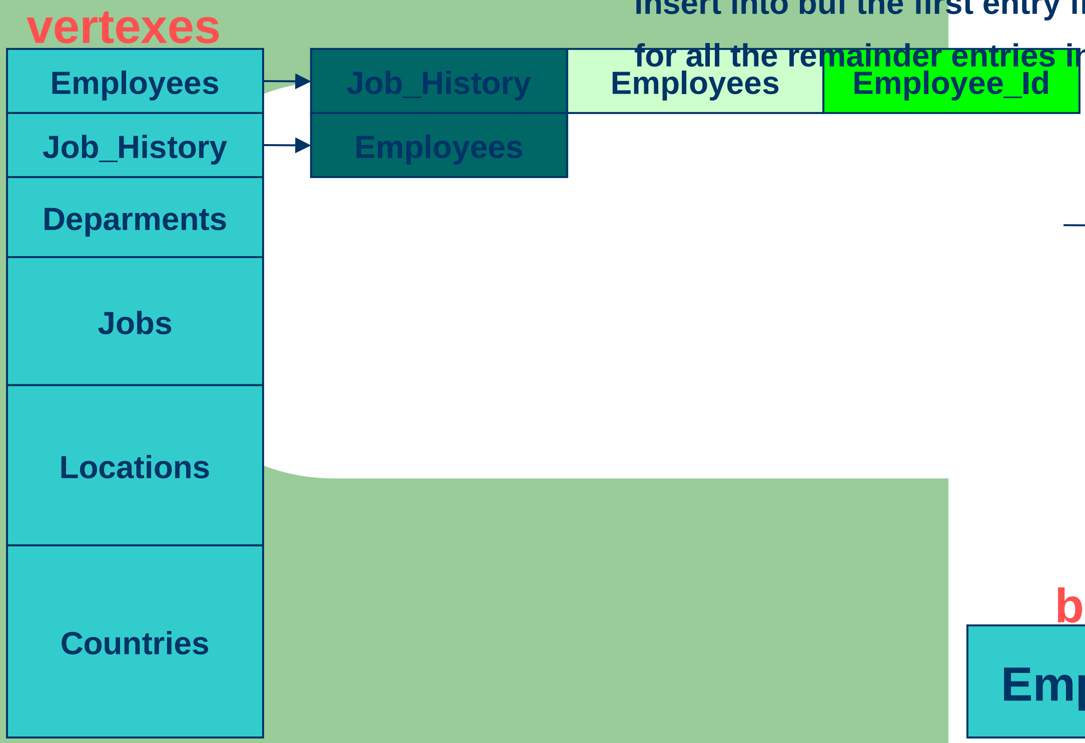
Countries

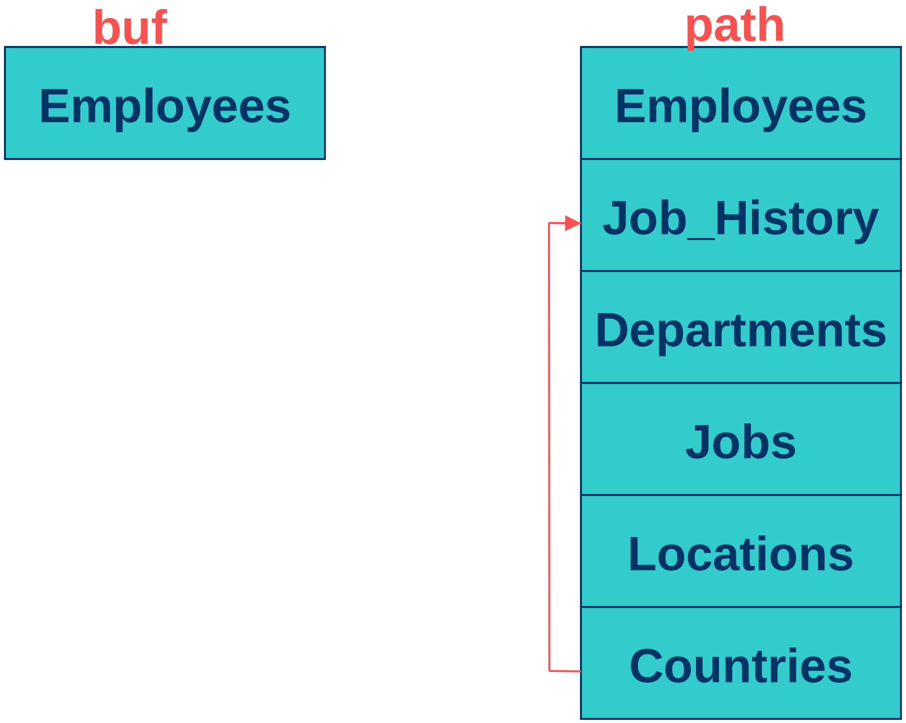
```
insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
        take one $T_i$ at a time
    →   JoinPathAdjacentList($T_i$) = $T_{[buf]}$
        Key($T_i$) = getFirstAdjacentListKey($T_i$,$T_{[buf]}$)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



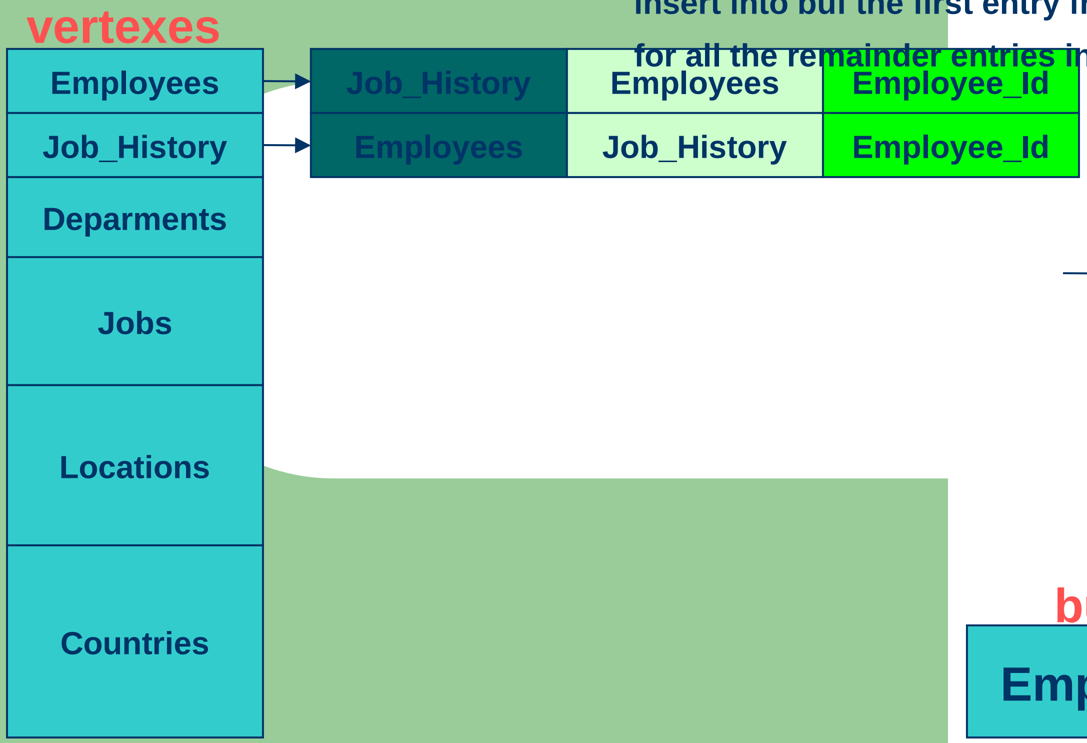


```
insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
        take one $T_i$ at a time
        JoinPathAdjacentList($T_i$) = $T_{[buf]}$
    →   Key($T_i$) = getFirstAdjacentListKey($T_i$,$T_{[buf]}$)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

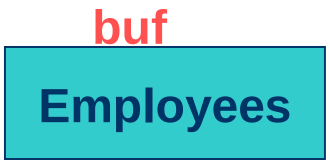


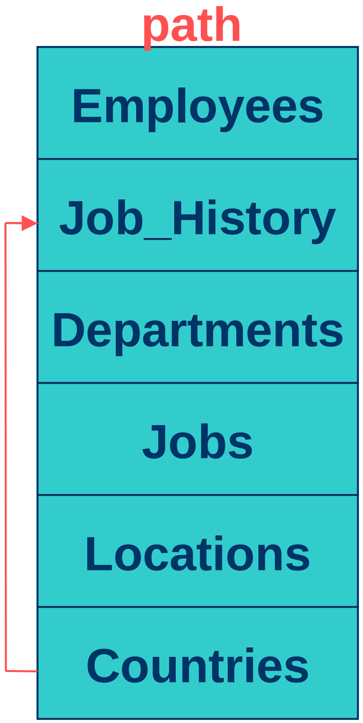


```
insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
        take one $T_i$ at a time
        JoinPathAdjacentList($T_i$) = $T_{[buf]}$
        Key($T_i$) = getFirstAdjacentListKey($T_i$,$T_{[buf]}$)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

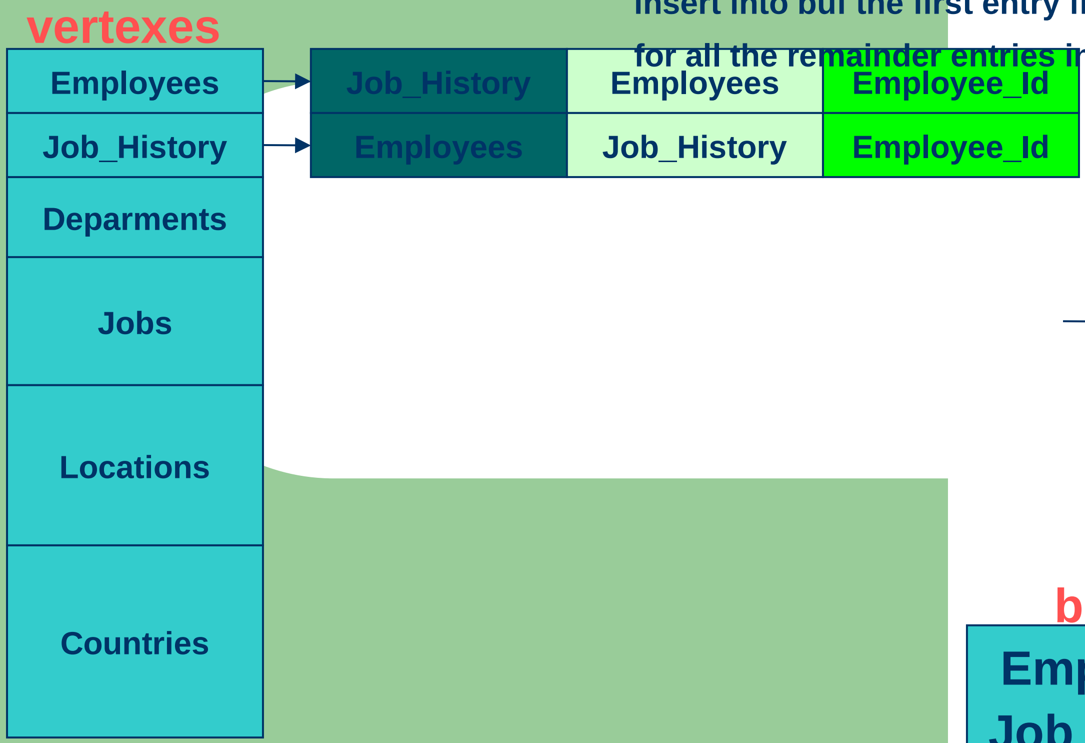


Job_History	Employees	Employee_Id
Employees	Job_History	Employee_Id

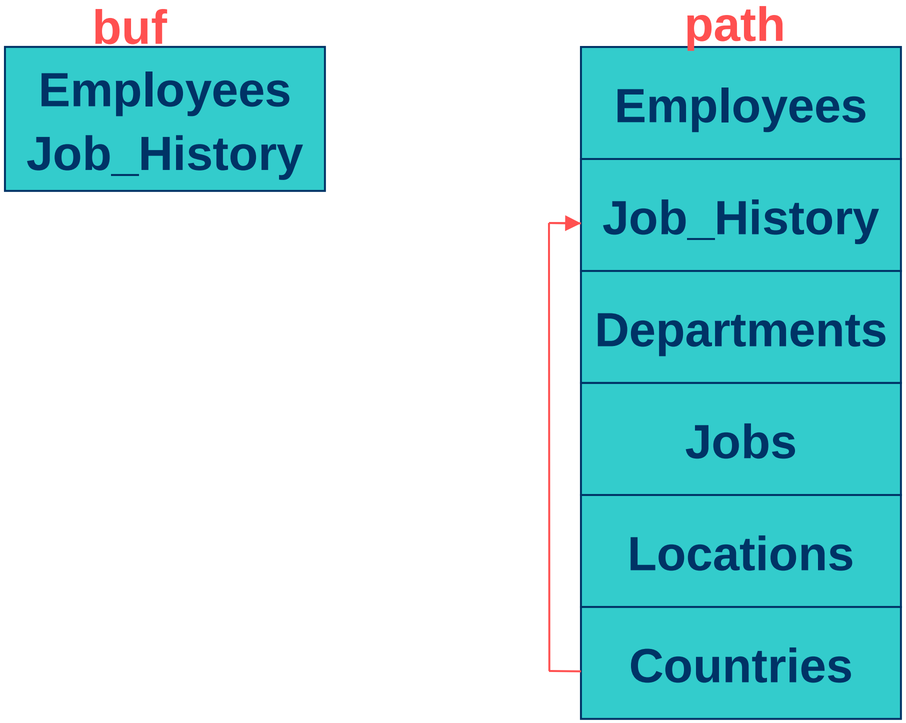




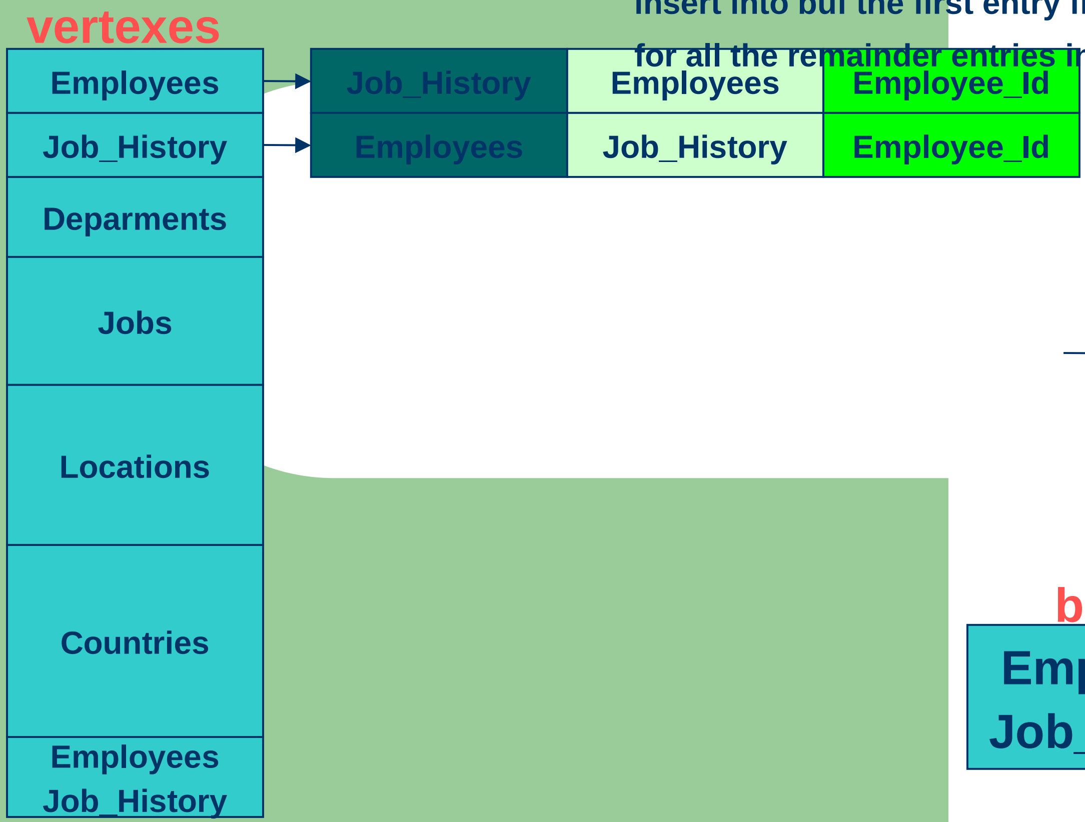
```
insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
    take one $T_i$ at a time
    $JoinPathAdjacentList(T_i) = T_{[buf]}$
    $Key(T_i) = getFirstAdjacentListKey(T_i,T_{[buf]})$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



Job_History	Employees	Employee_Id
Employees	Job_History	Employee_Id

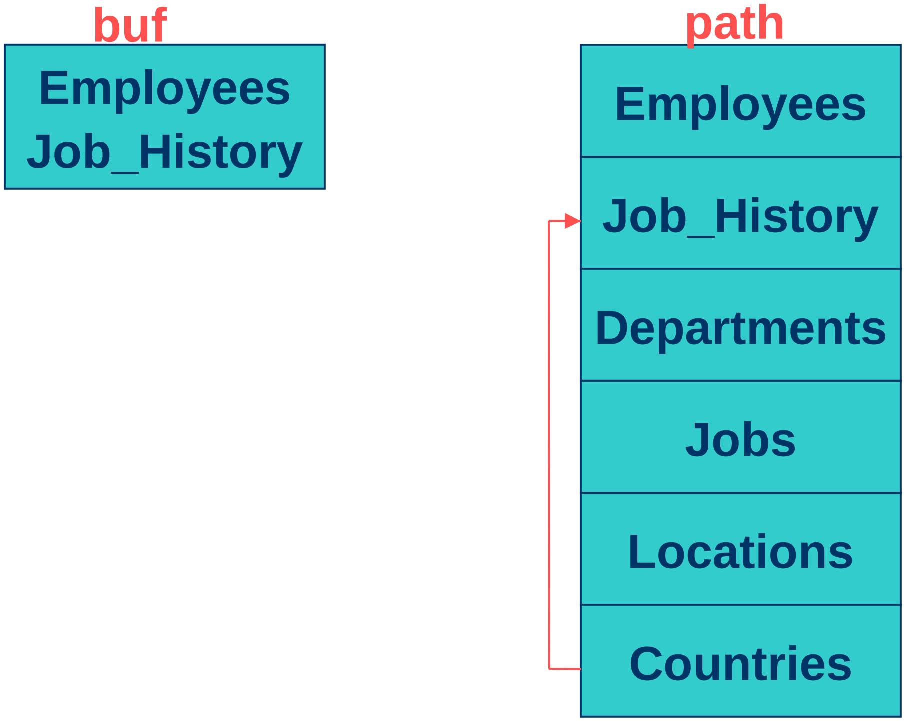


```
insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
        take one $T_i$ at a time
        JoinPathAdjacentList($T_i$) = $T_{[buf]}$
        Key($T_i$) = getFirstAdjacentListKey($T_i$,$T_{[buf]}$)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



vertexes

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



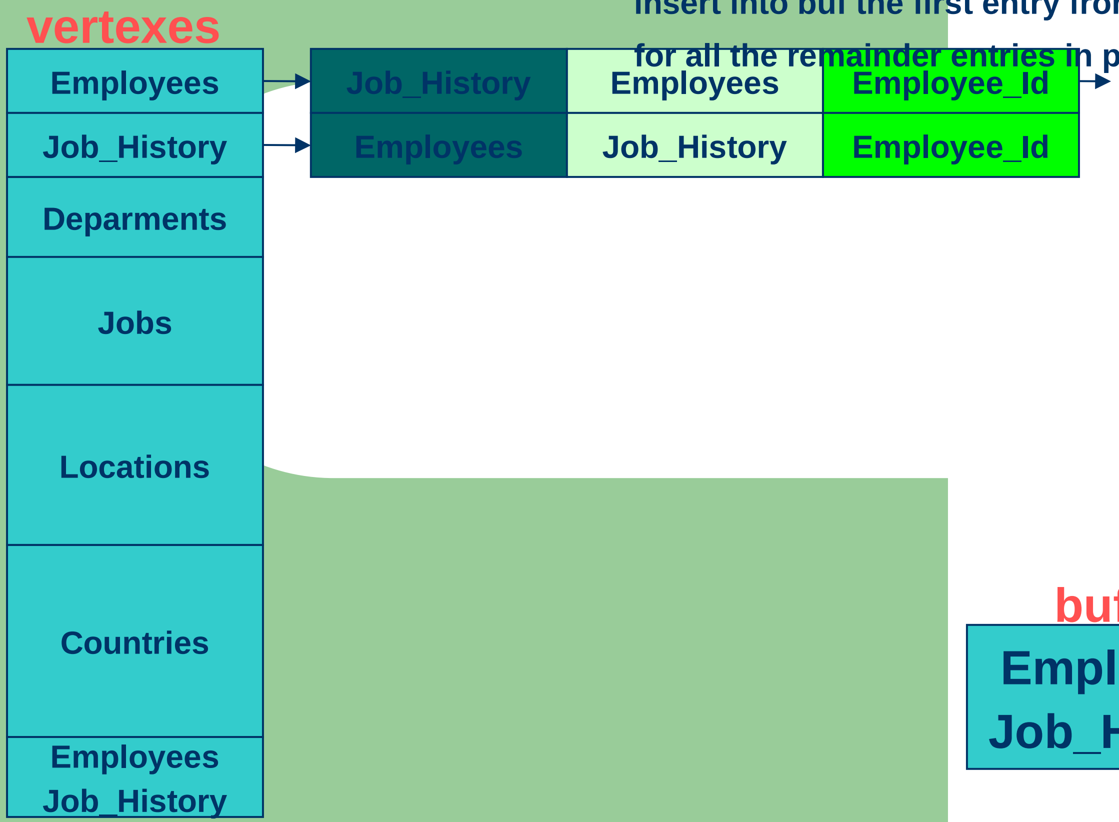
buf

buf
Employees Job_History

path

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

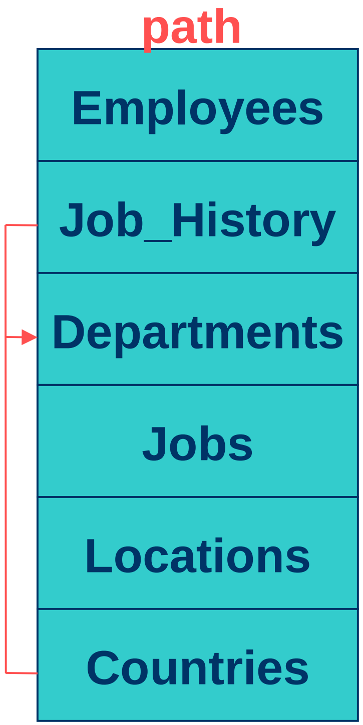
Countries

Employees
Job_History

Job_History	Employees	Employee_Id
Employees	Job_History	Employee_Id

buf

Employees
Job_History



path

Employees

Job_History

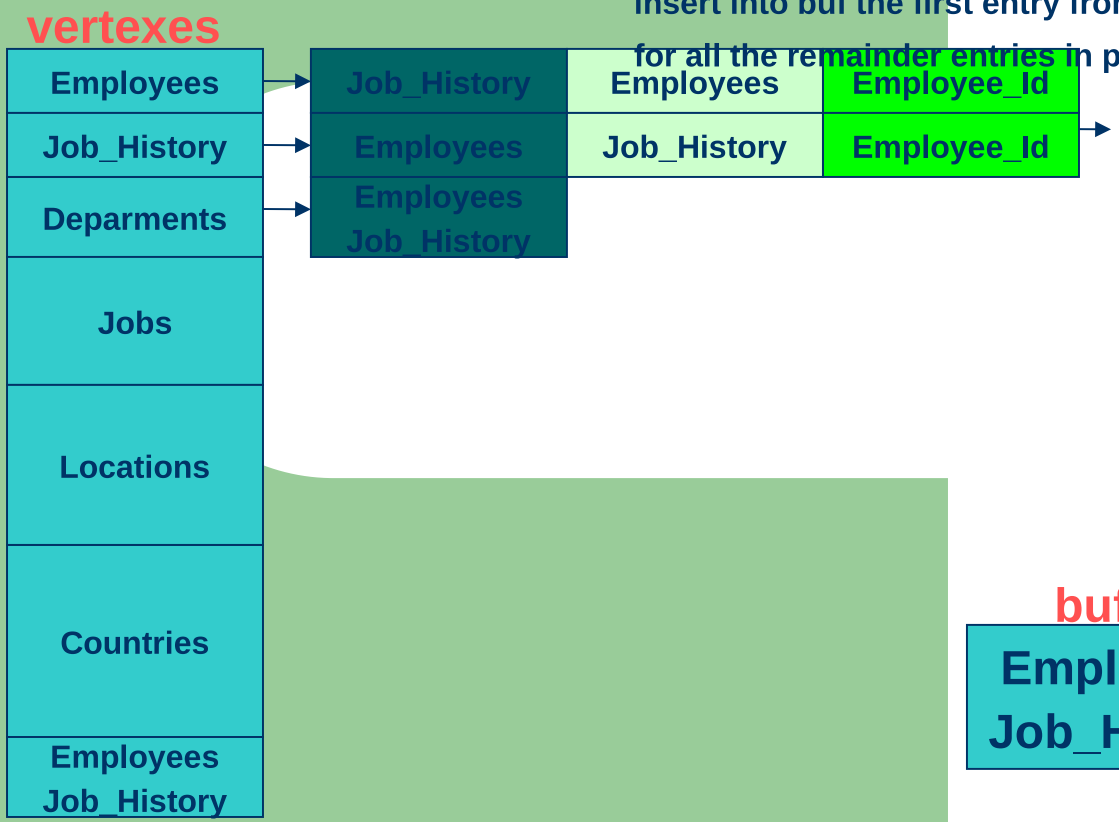
Departments

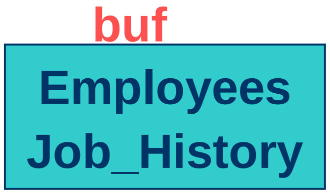
Jobs

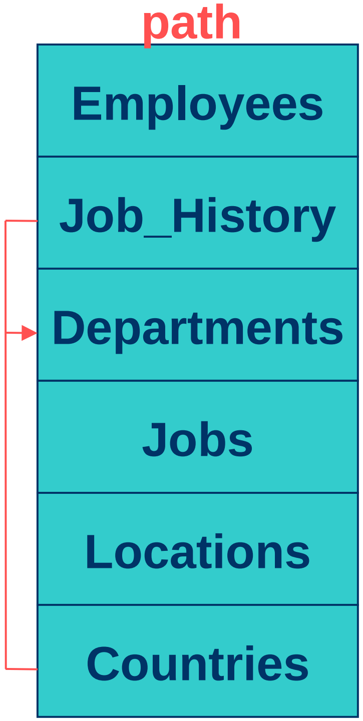
Locations

Countries

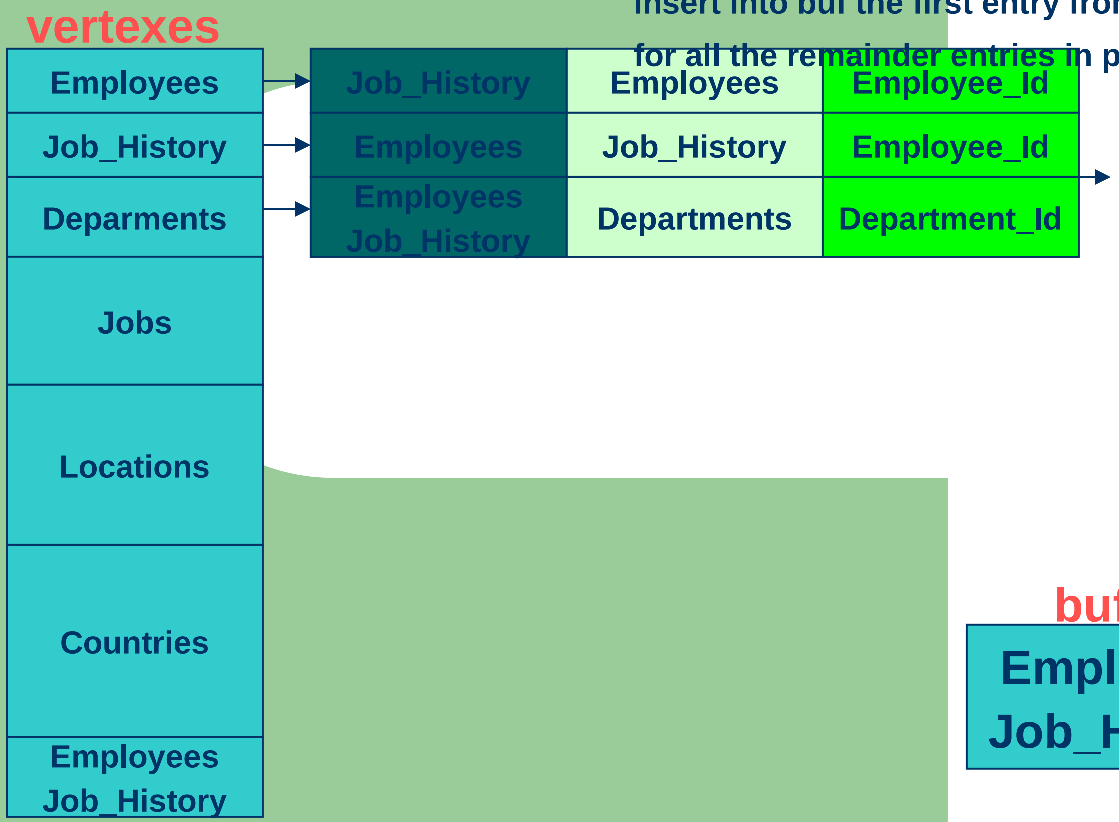
```
insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
    take one $T_i$ at a time
    JoinPathAdjacentList($T_i$) = $T_{[buf]}$
    Key($T_i$) = getFirstAdjacentListKey($T_i$,$T_{[buf]}$)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



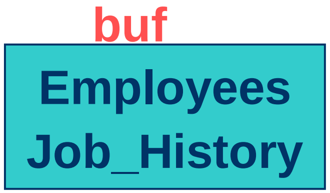


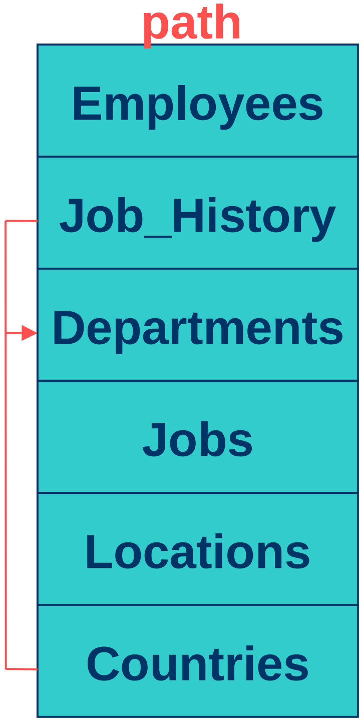


```
insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
        take one $T_i$ at a time
        JoinPathAdjacentList($T_i$) = $T_{[buf]}$
        Key($T_i$) = getFirstAdjacentListKey($T_i$,$T_{[buf]}$)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

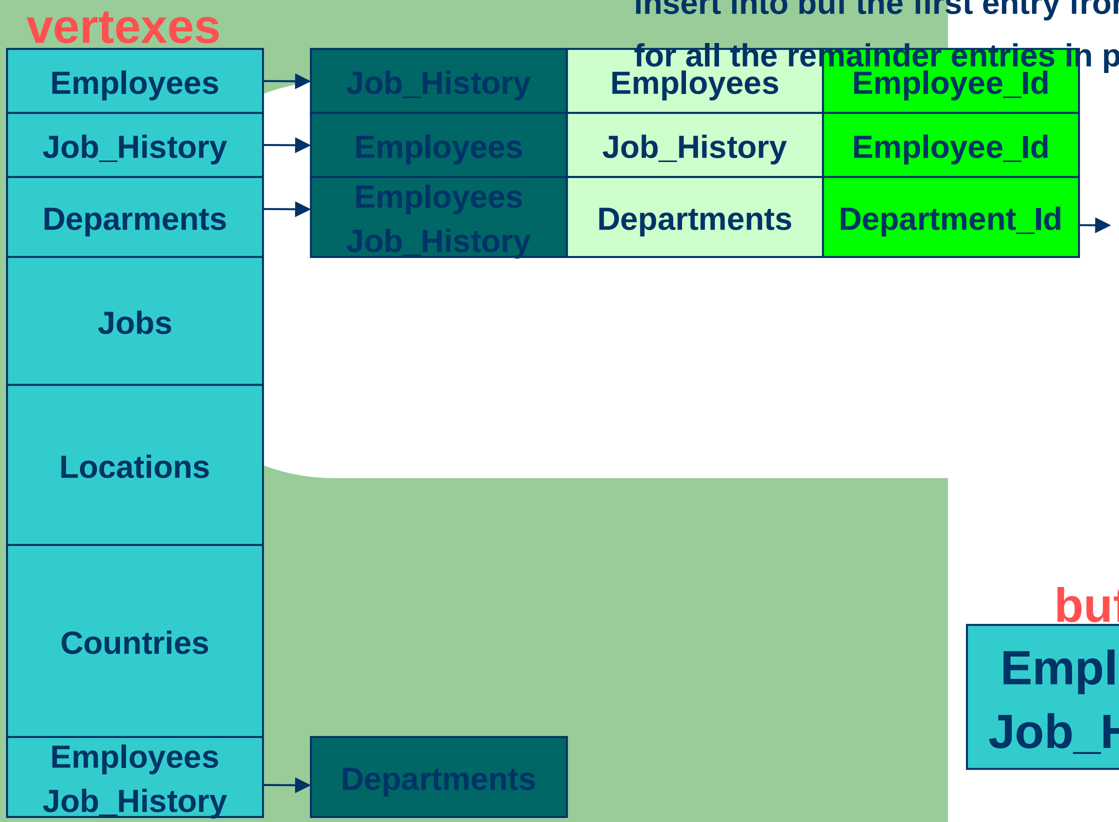


Job_History	Employees	Employee_Id
Employees	Job_History	Employee_Id
Employees Job_History	Departments	Department_Id



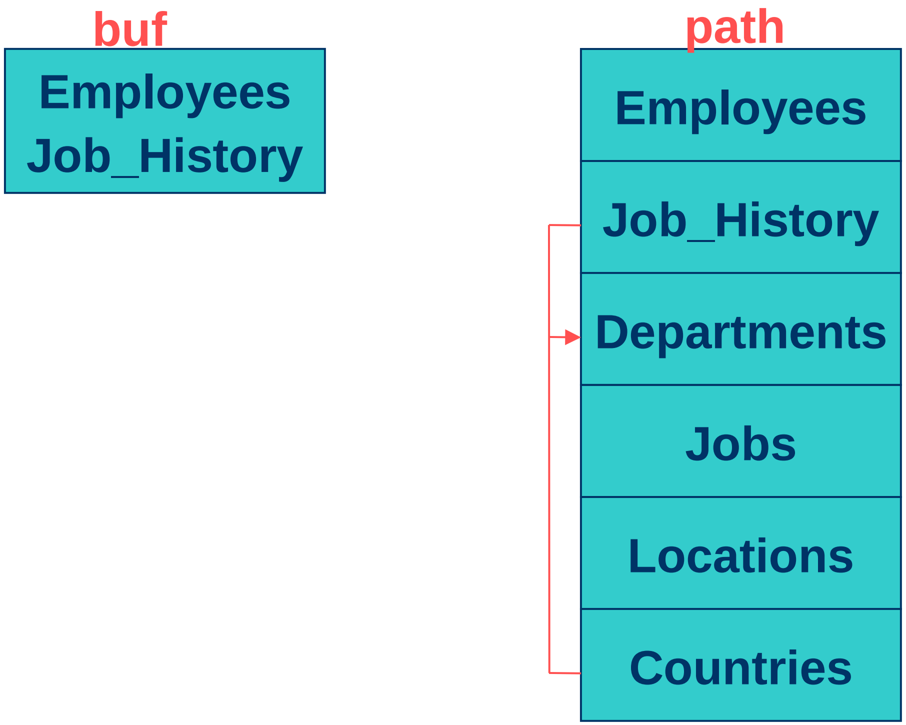


```
insert all the names of base tables from path as vertexes in JoinPathList
create a local buffer buf
insert into buf the first entry from path
for all the remainder entries in path do
        take one $T_i$ at a time
        JoinPathAdjacentList($T_i$) = $T_{[buf]}$
        Key($T_i$) = getFirstAdjacentListKey($T_i$,$T_{[buf]}$)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



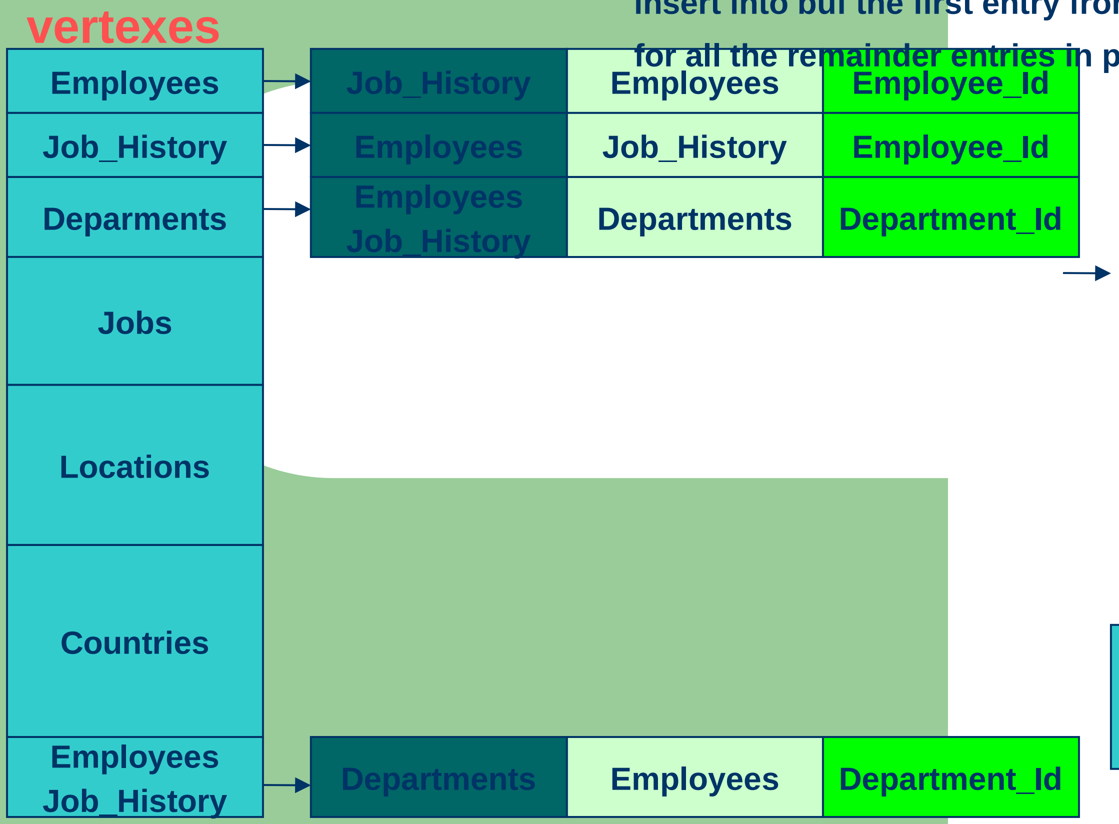
buf

Employees
Job_History

path

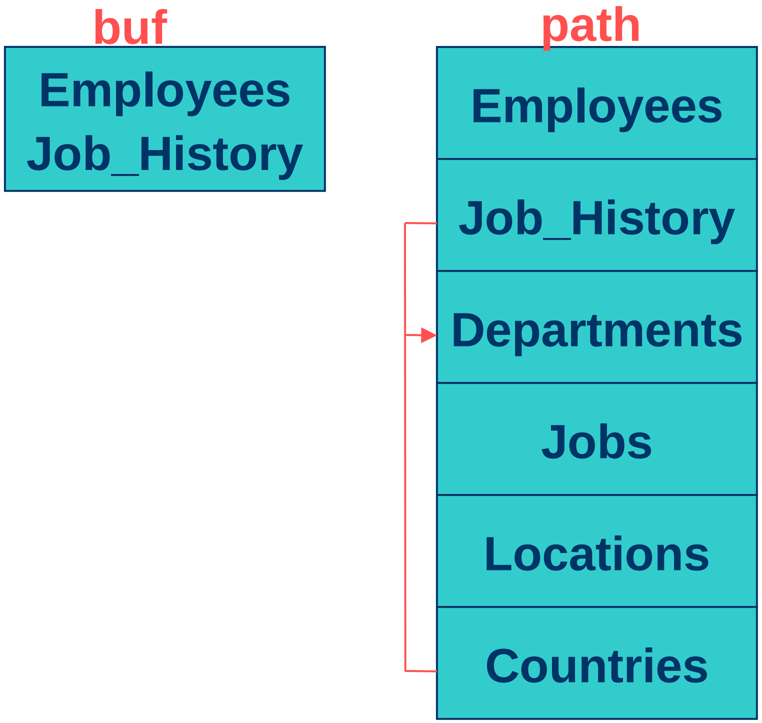
Employees
Job_History
Departments
Jobs
Locations
Countries

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



buf

Employees
Job_History

path

Employees

Job_History

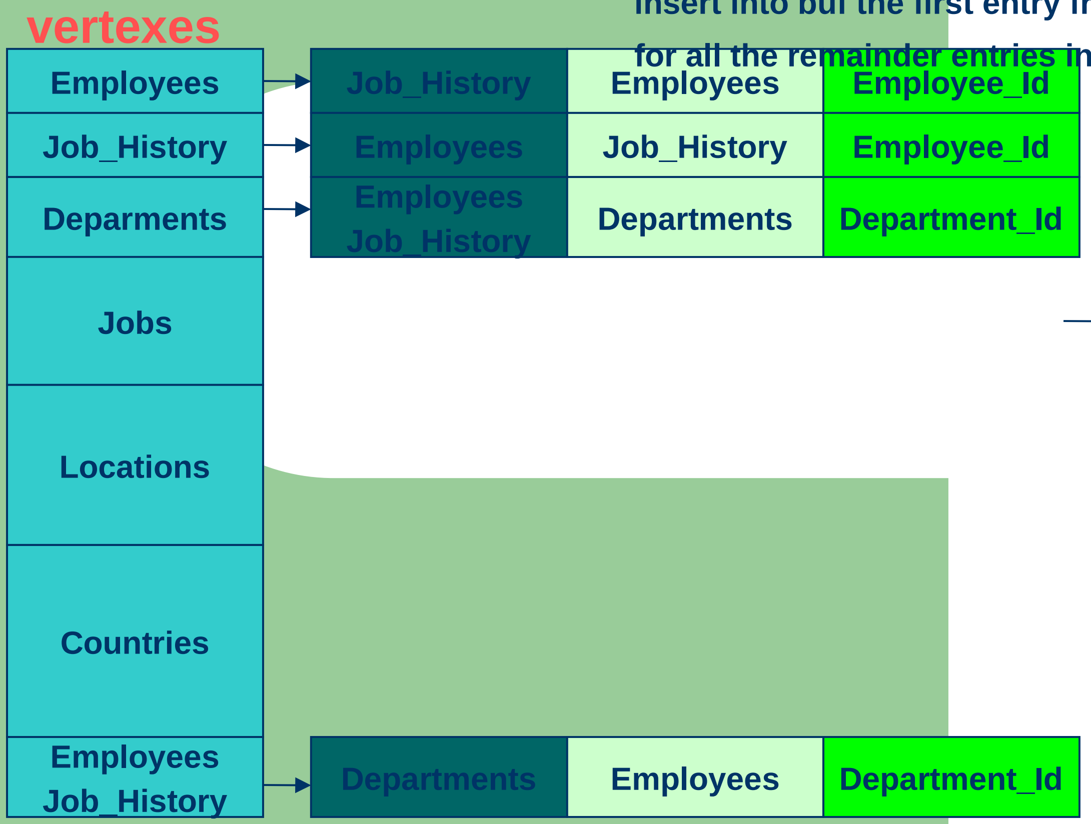
Departments

Jobs

Locations

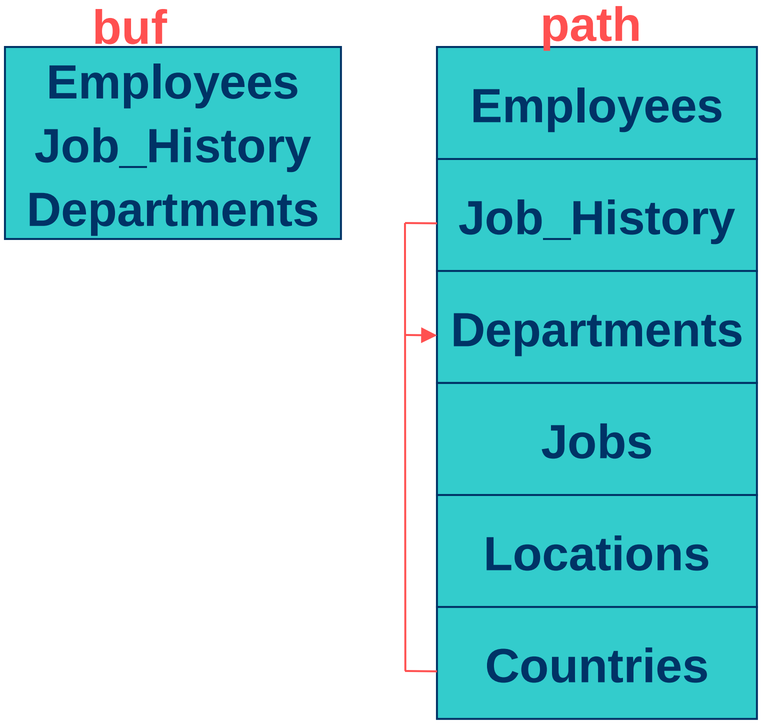
Countries

```
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[$T_{[buf]}$ ] = $T_{[buf]}$
```



vertexes

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



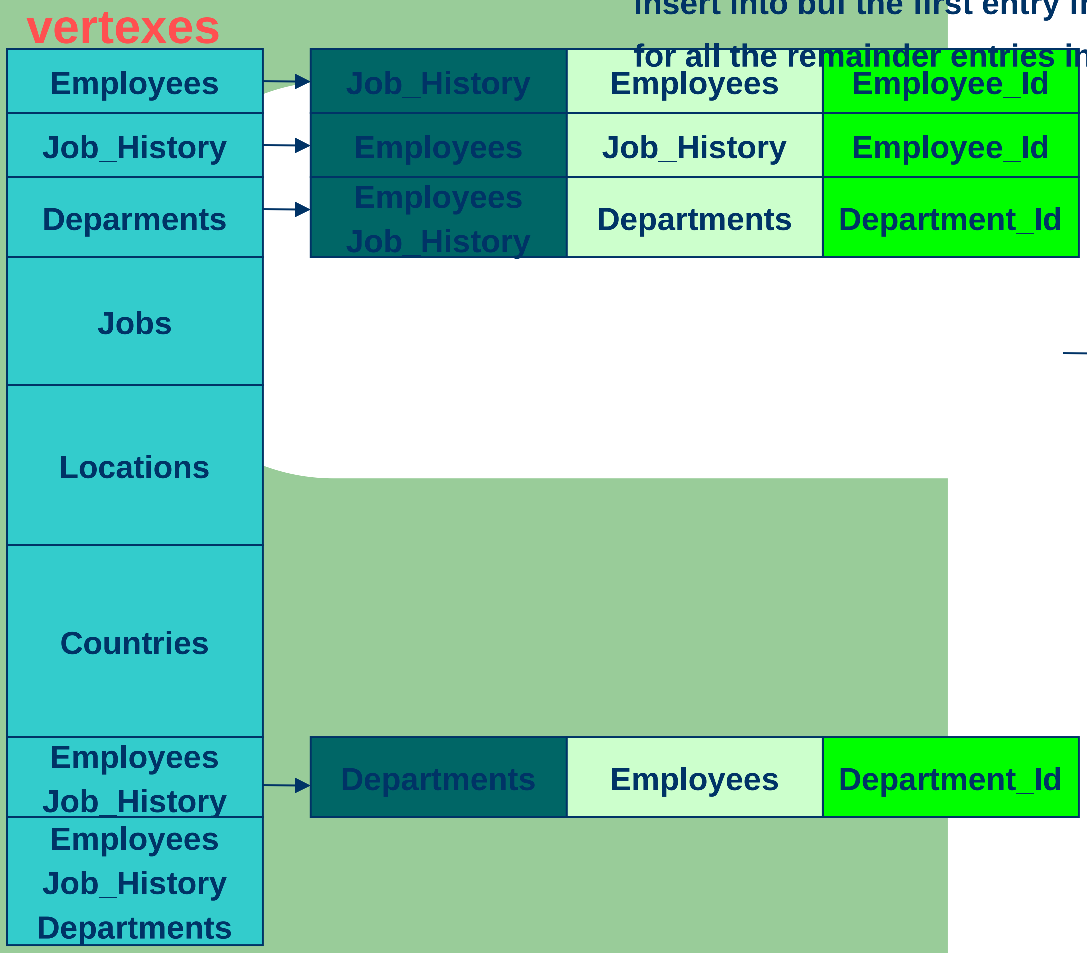
buf

Employees
Job_History
Departments

path

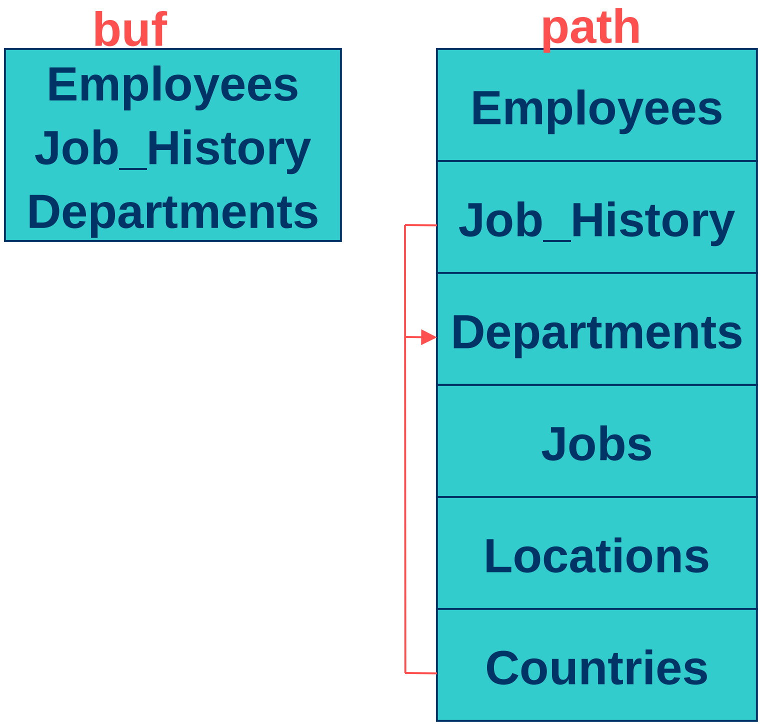
Employees
Job_History
Departments
Jobs
Locations
Countries

```
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				
Locations				
Countries				
Employees Job_History	→	Departments	Employees	Department_Id
Employees Job_History Departments				



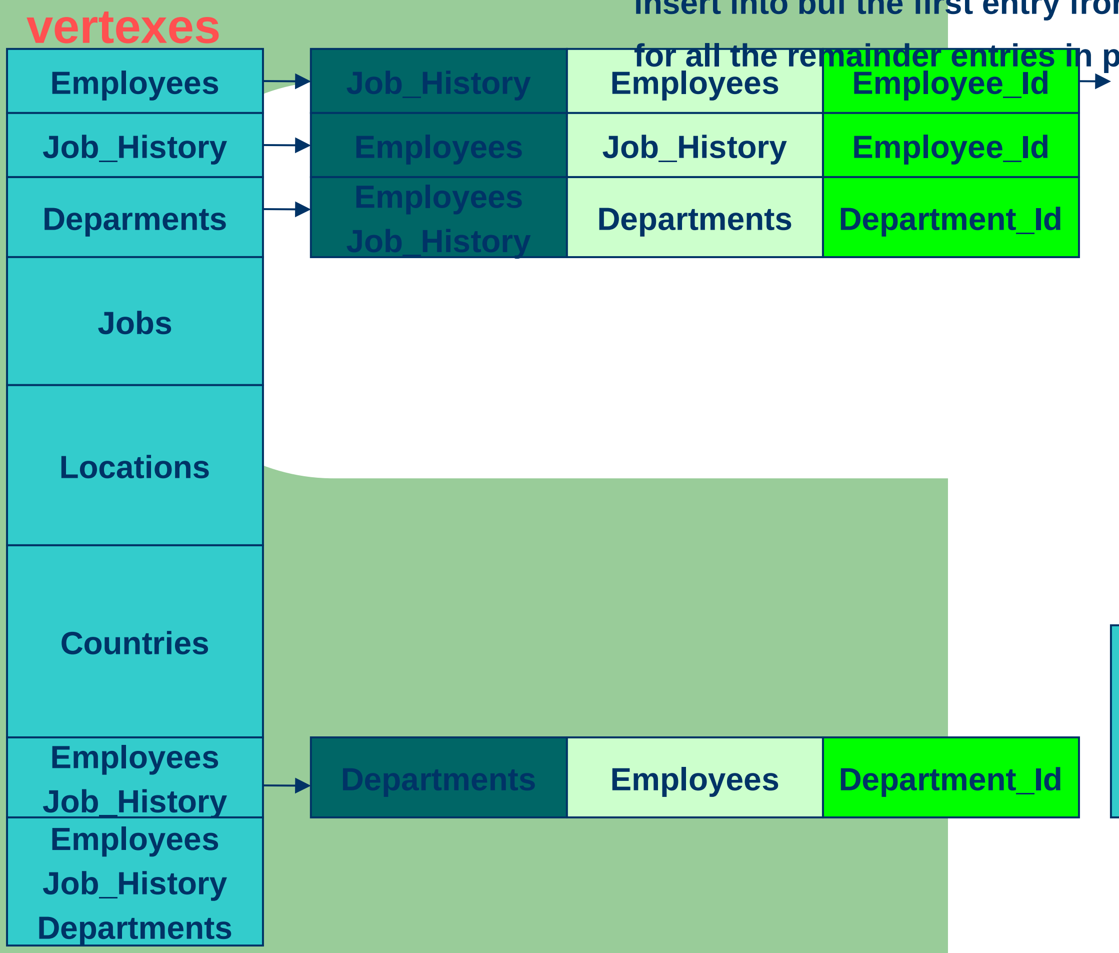
buf

Employees Job_History Departments

path

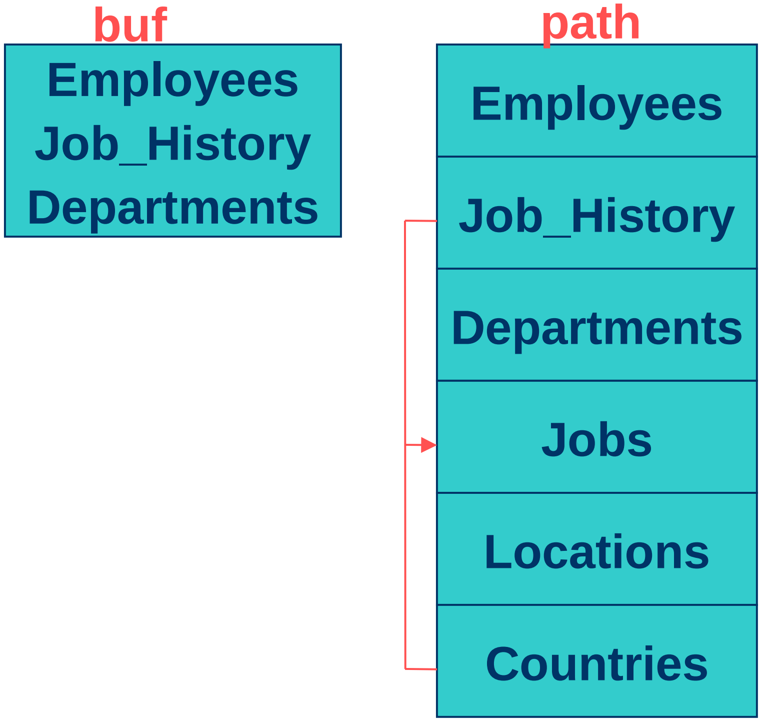
Employees
Job_History
Departments
Jobs
Locations
Countries

```
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			
Locations			
Countries			
Employees Job_History	Departments	Employees	Department_Id
Employees Job_History Departments			



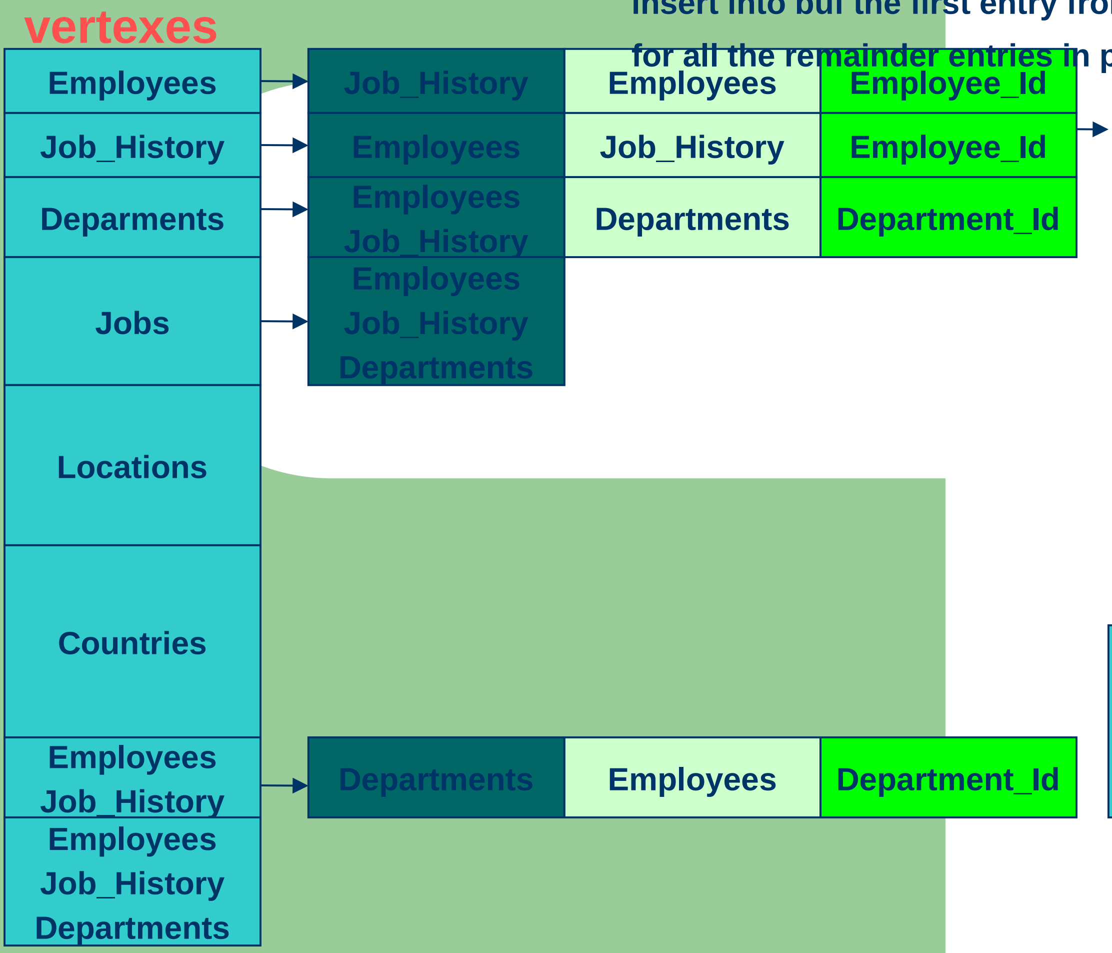
buf

Employees
Job_History
Departments

path

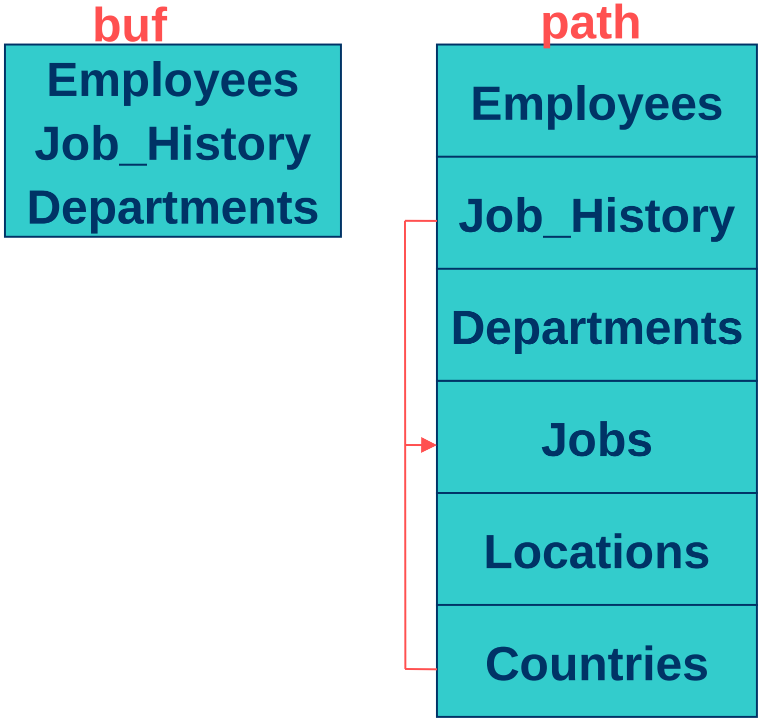
Employees
Job_History
Departments
Jobs
Locations
Countries

```
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		
Locations			
Countries			
Employees Job_History	Departments	Employees	Department_Id
Employees Job_History Departments			



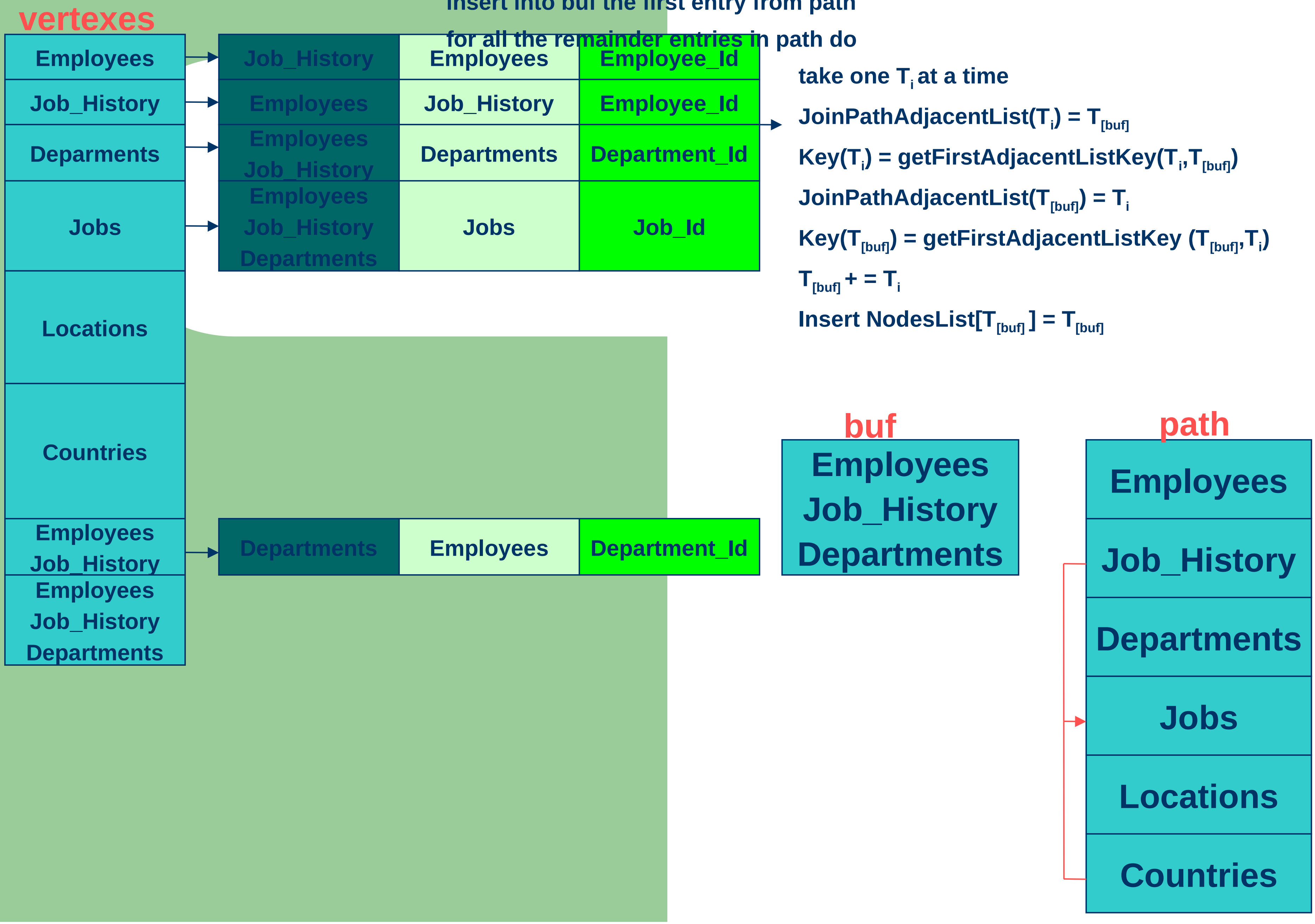
buf

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[$T_{[buf]}$ ] = $T_{[buf]}$
```



Employees
Job_History
Deparments
Jobs
Locations
Countries
Employees Job_History
Employees Job_History Departments

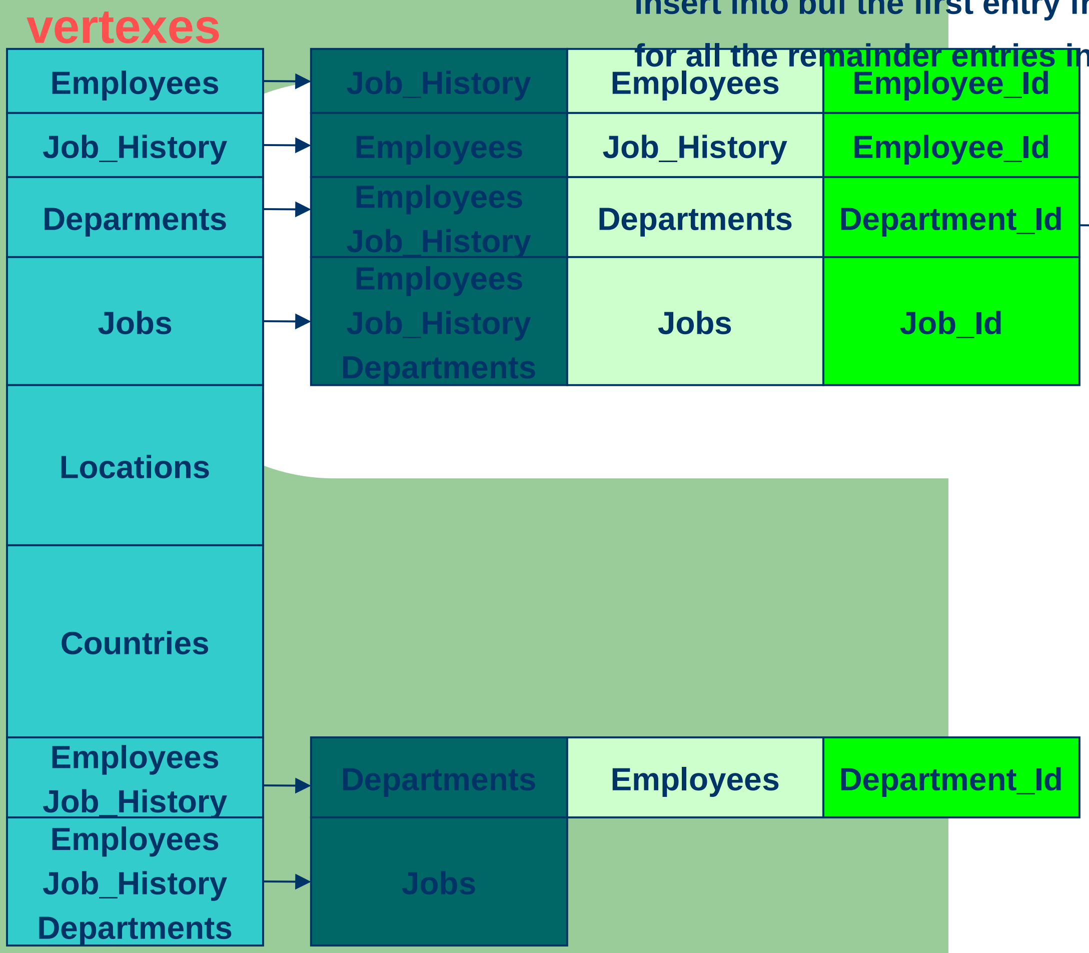
Job_History	Employees	Employee_Id
Employees	Job_History	Employee_Id
Employees Job_History	Departments	Department_Id
Employees Job_History Departments	Jobs	Job_Id

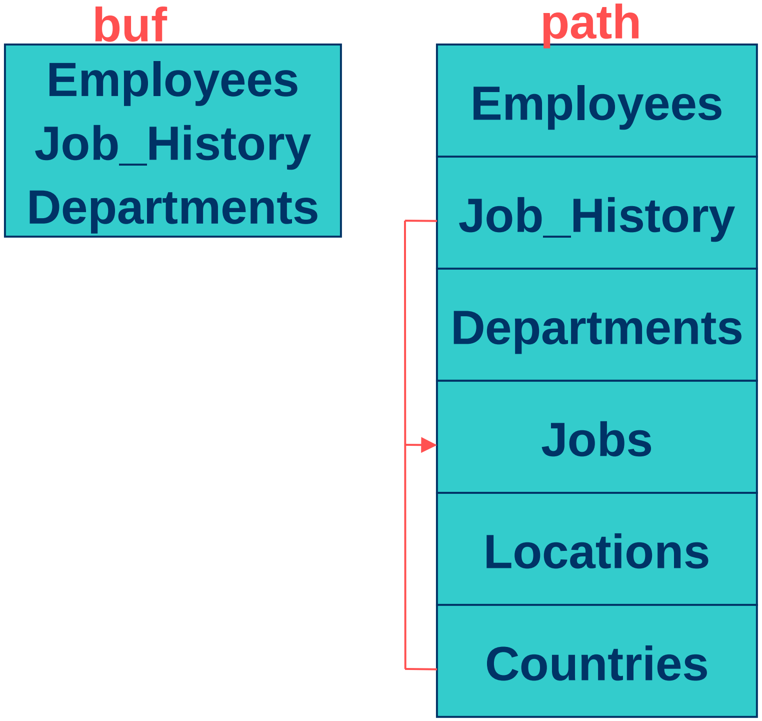
Departments	Employees	Department_Id

Employees Job_History Departments

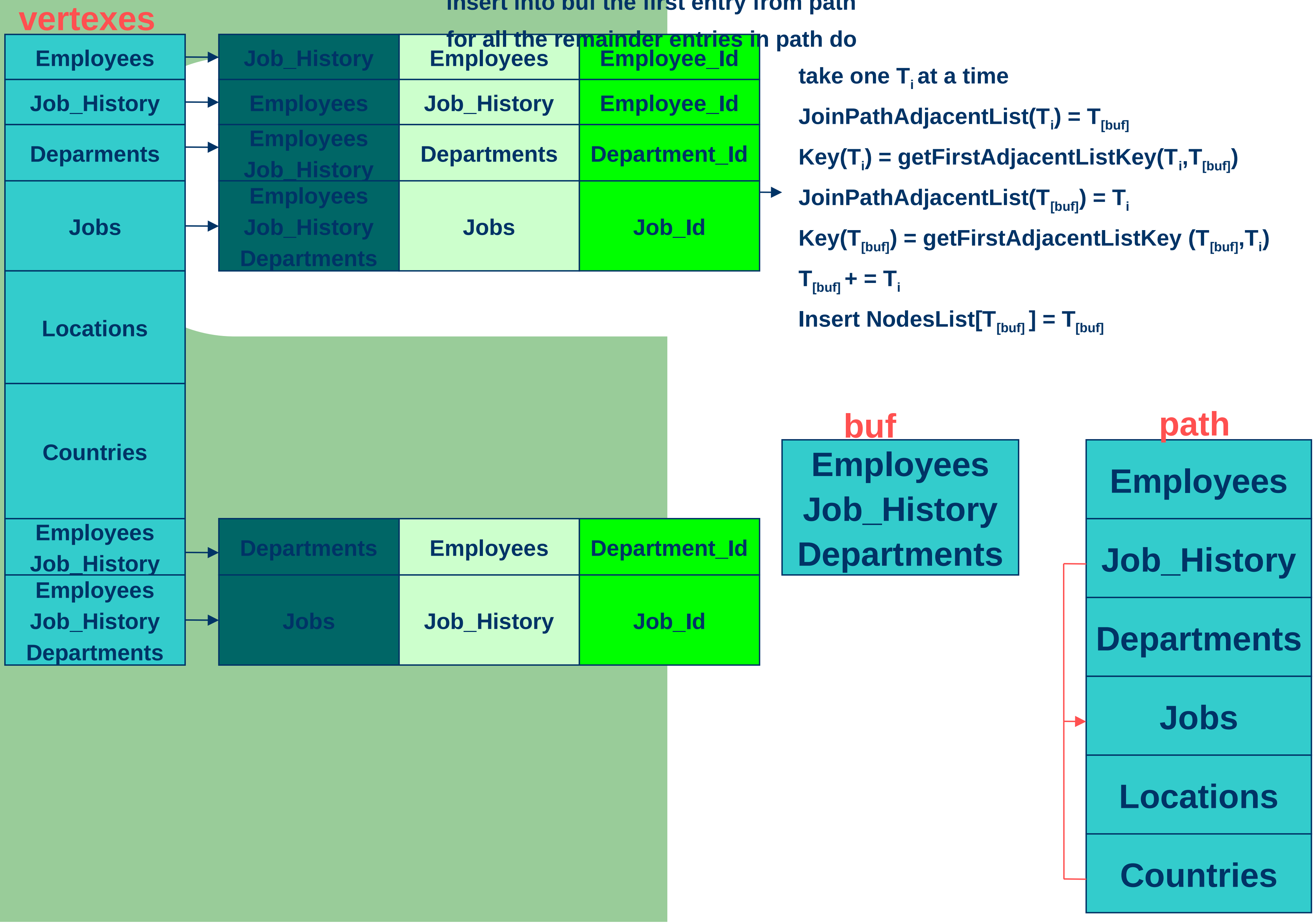
Employees
Job_History
Departments
Jobs
Locations
Countries

```
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[$T_{[buf]}$ ] = $T_{[buf]}$
```





```
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[$T_{[buf]}$ ] = $T_{[buf]}$
```

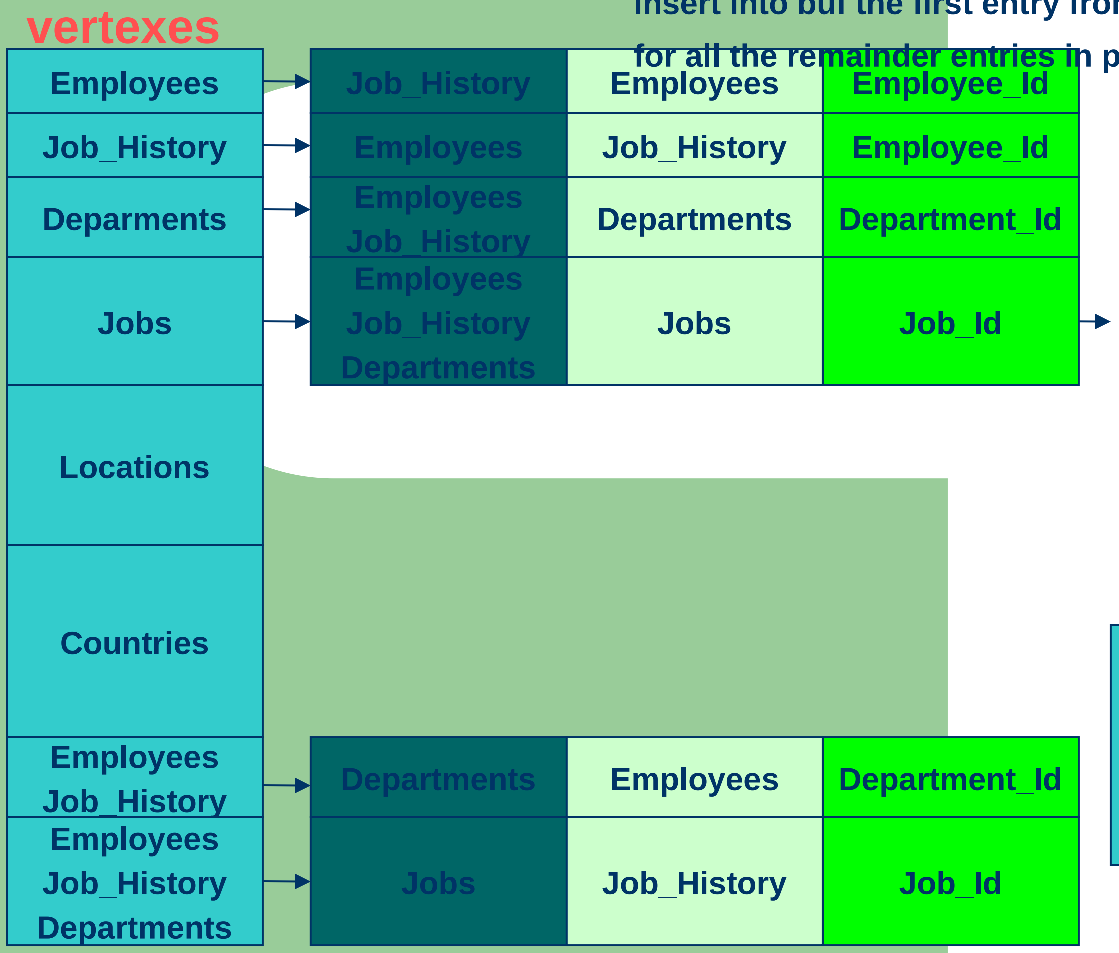


Employees	Job_History	Employees	Employee_Id
Job_History	Employees	Job_History	Employee_Id
Deparments	Employees Job_History	Departments	Department_Id
Jobs	Employees Job_History Departments	Jobs	Job_Id
Locations			
Countries			
Employees Job_History	Departments	Employees	Department_Id
Employees Job_History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

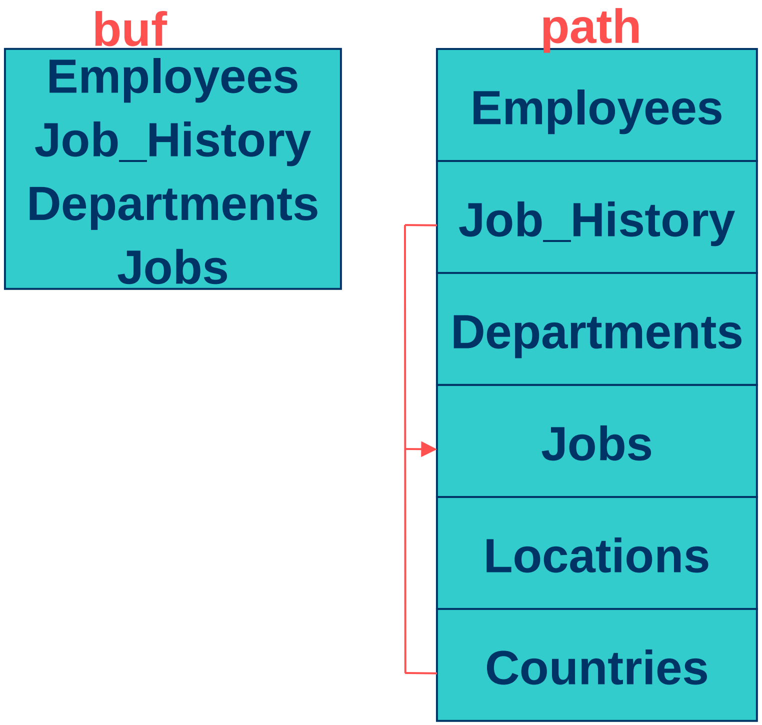
Employees
Job_History
Departments

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	Job_History	Job_Id

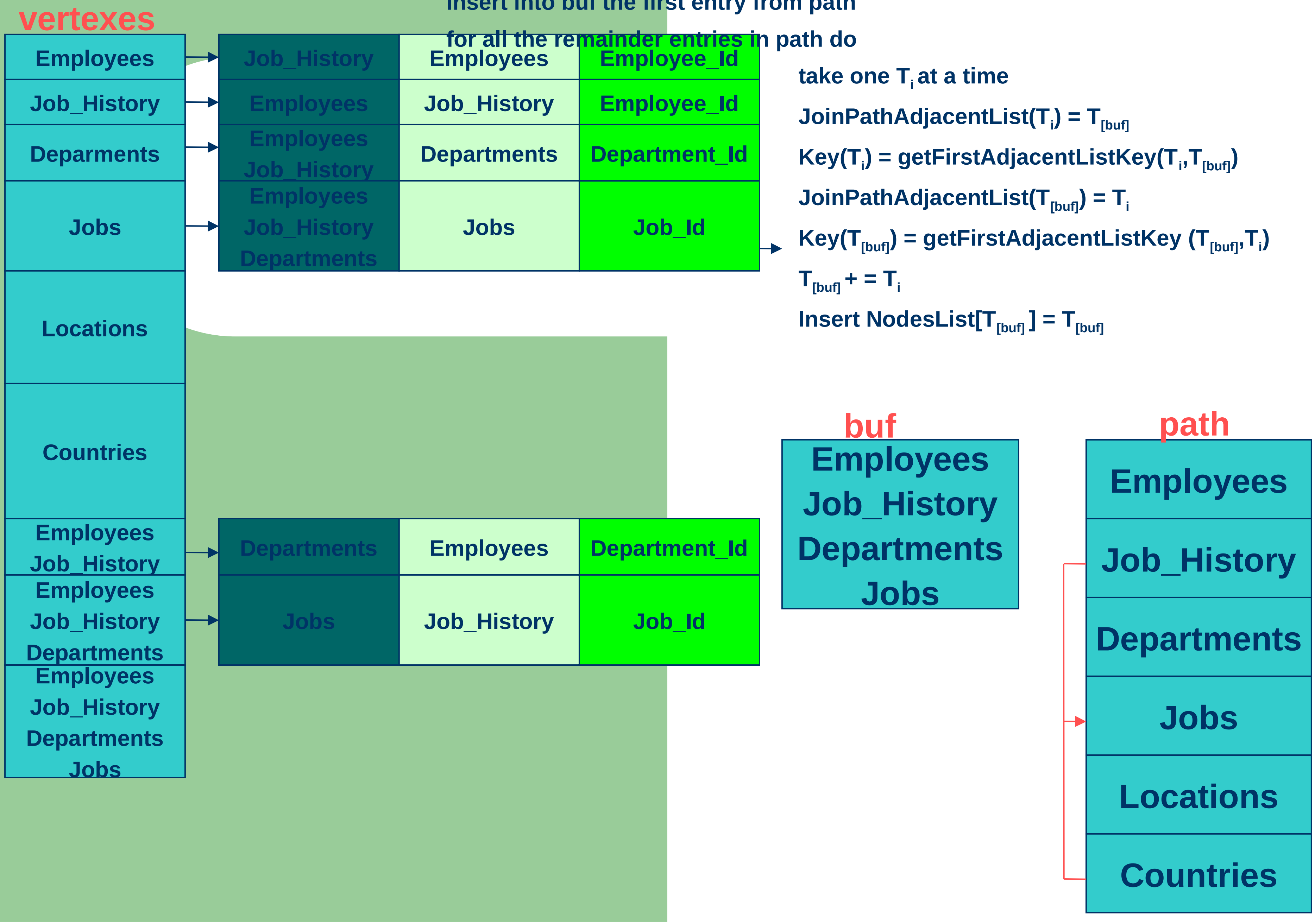

buf

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	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
Employees Job_History Departments Jobs			

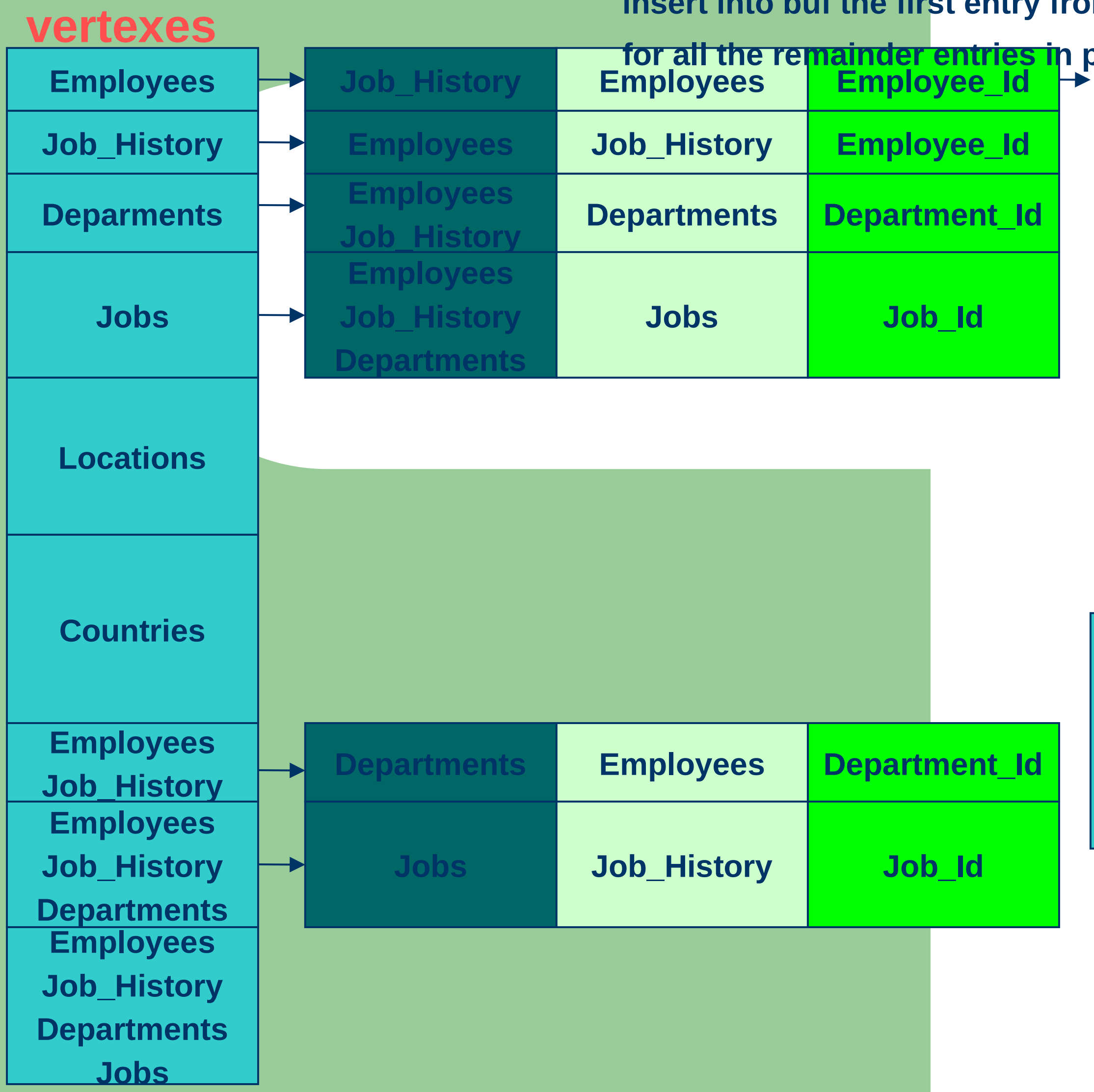
buf

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[$T_{[buf]}$ ] = $T_{[buf]}$
```

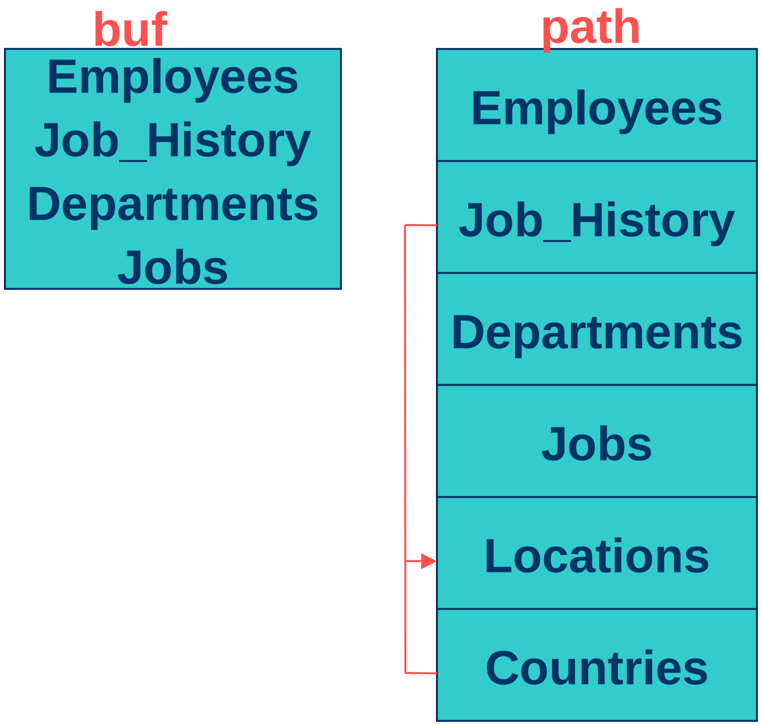


vertexes

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



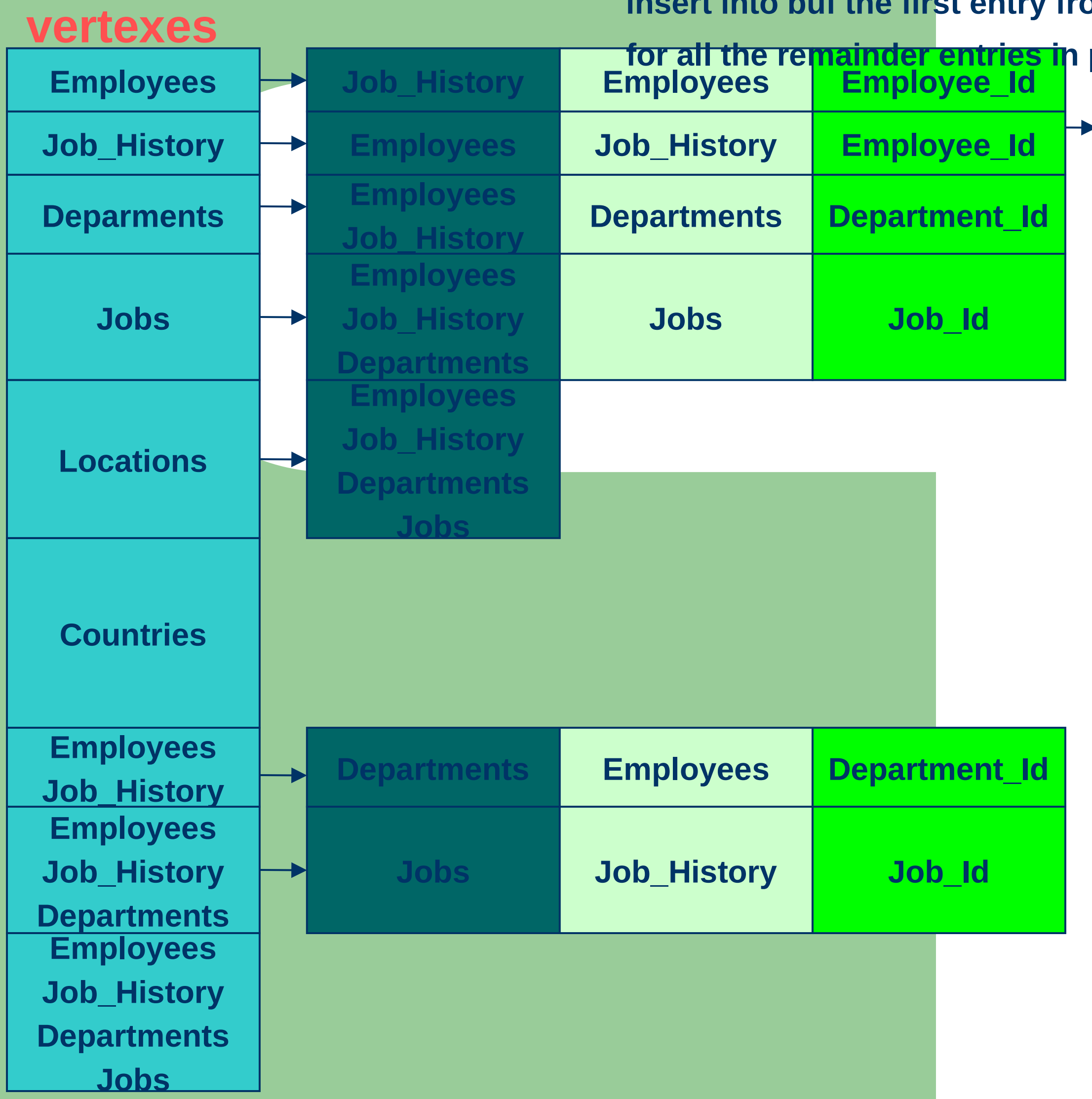
buf

Employees Job_History Departments Jobs

path

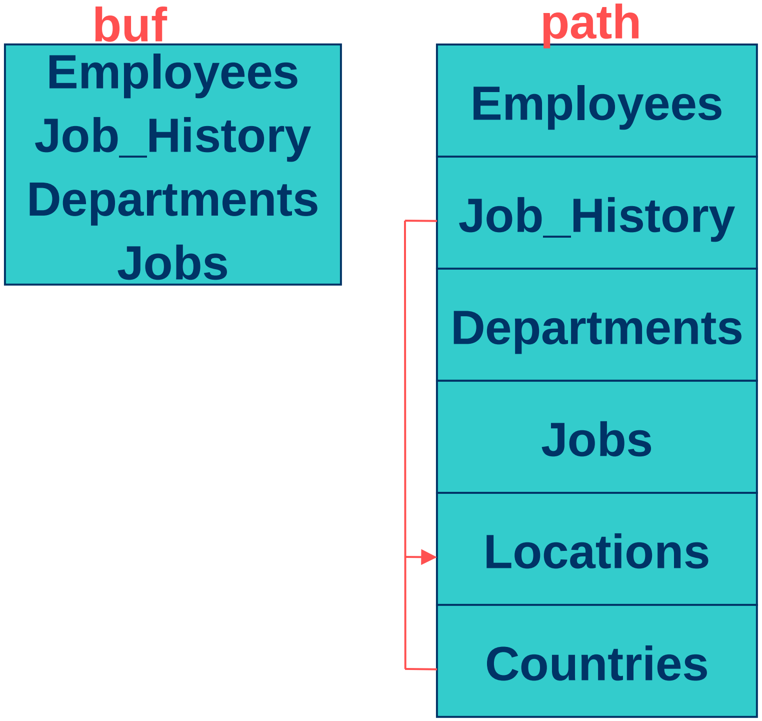
Employees
Job_History
Departments
Jobs
Locations
Countries

```
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	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		
Countries			
Employees Job_History	Departments	Employees	Department_Id
Employees Job_History Departments	Jobs	Job_History	Job_Id
Employees Job_History Departments Jobs			



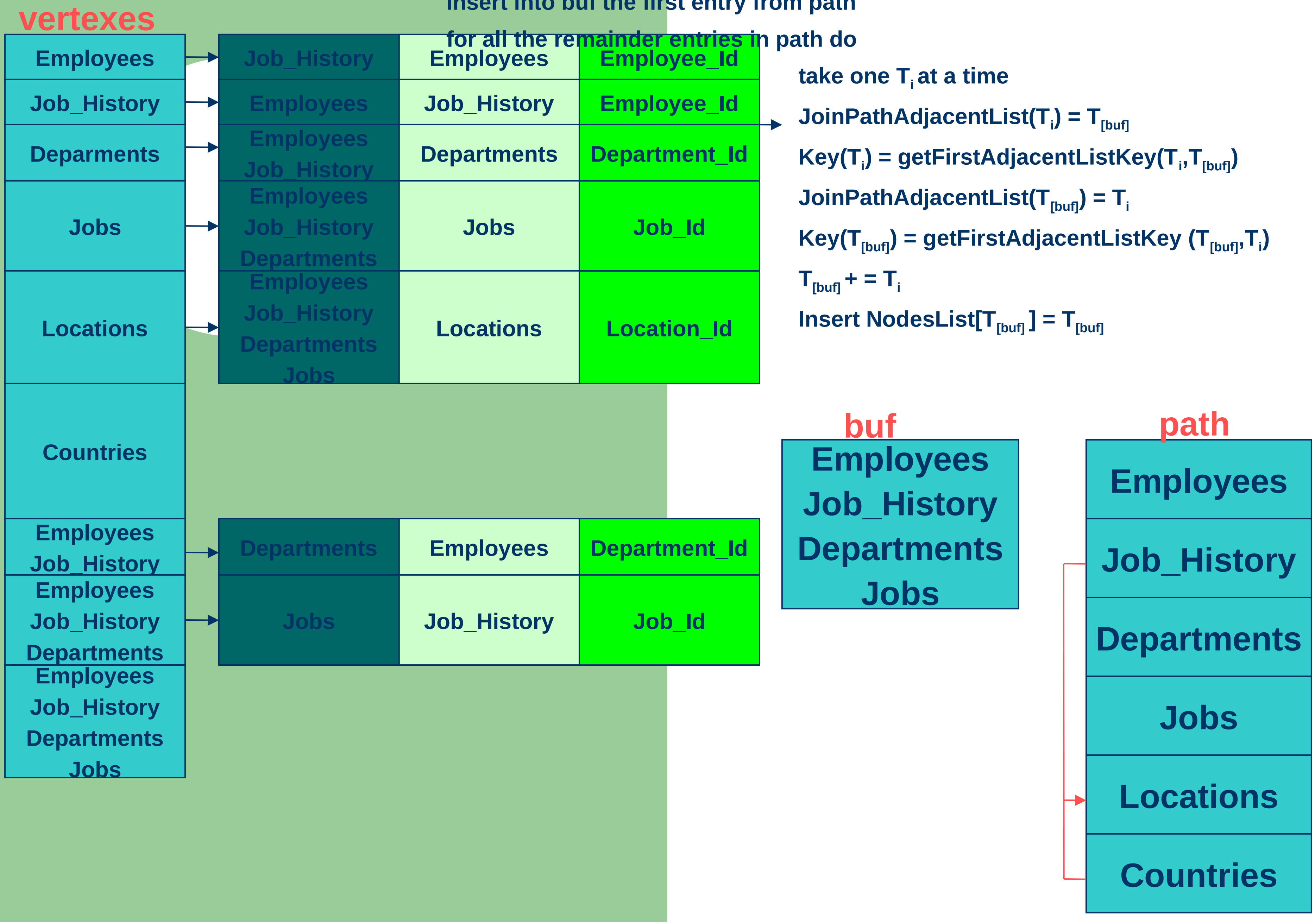
buf

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

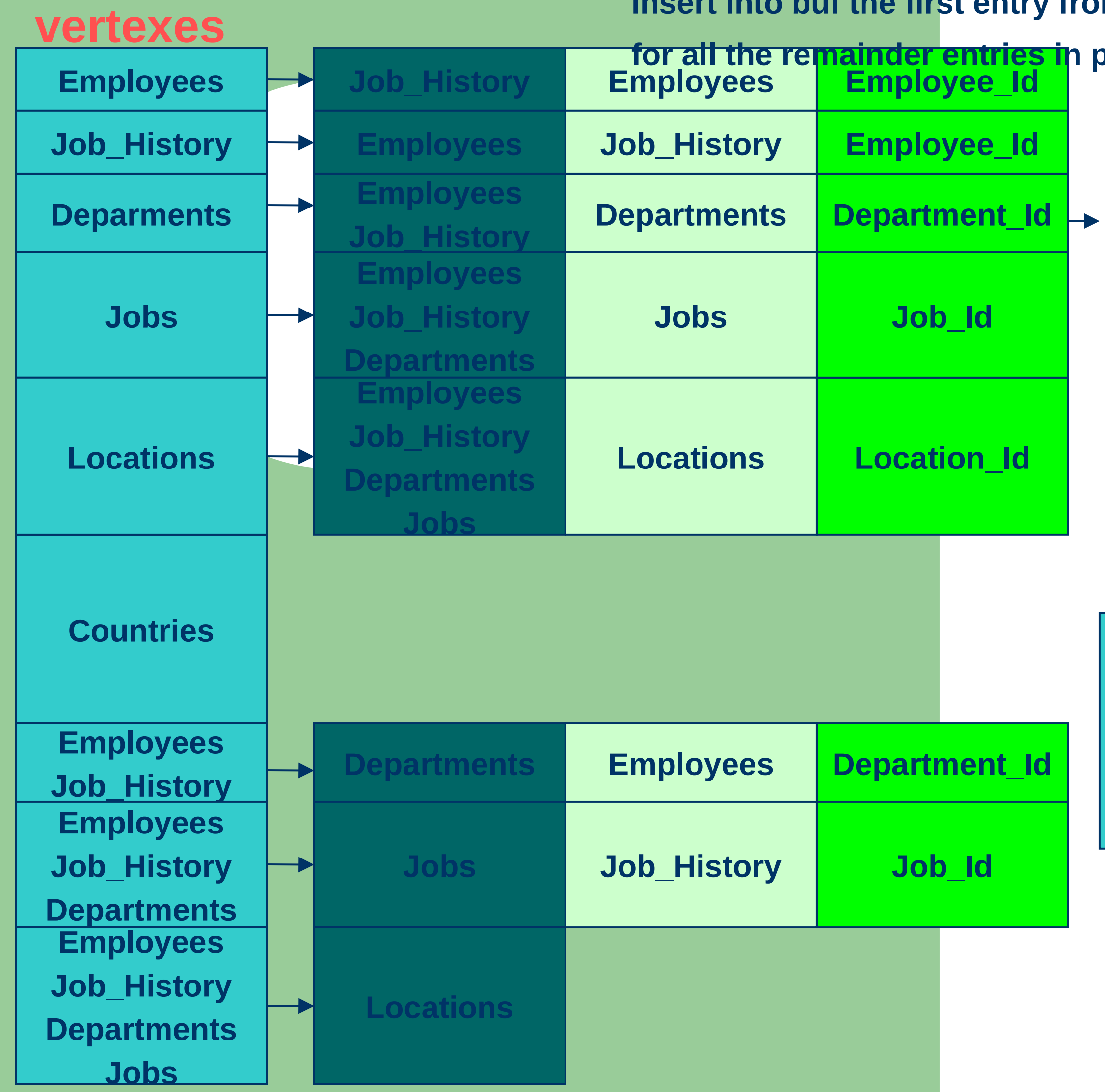
buf

Employees Job_History Departments Jobs

path

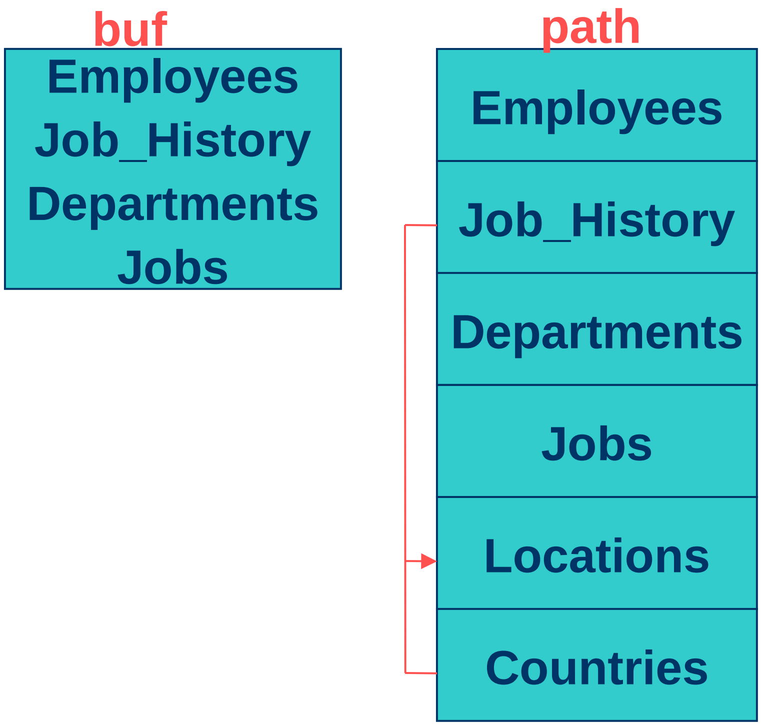
Employees
Job_History
Departments
Jobs
Locations
Countries

```
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	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		



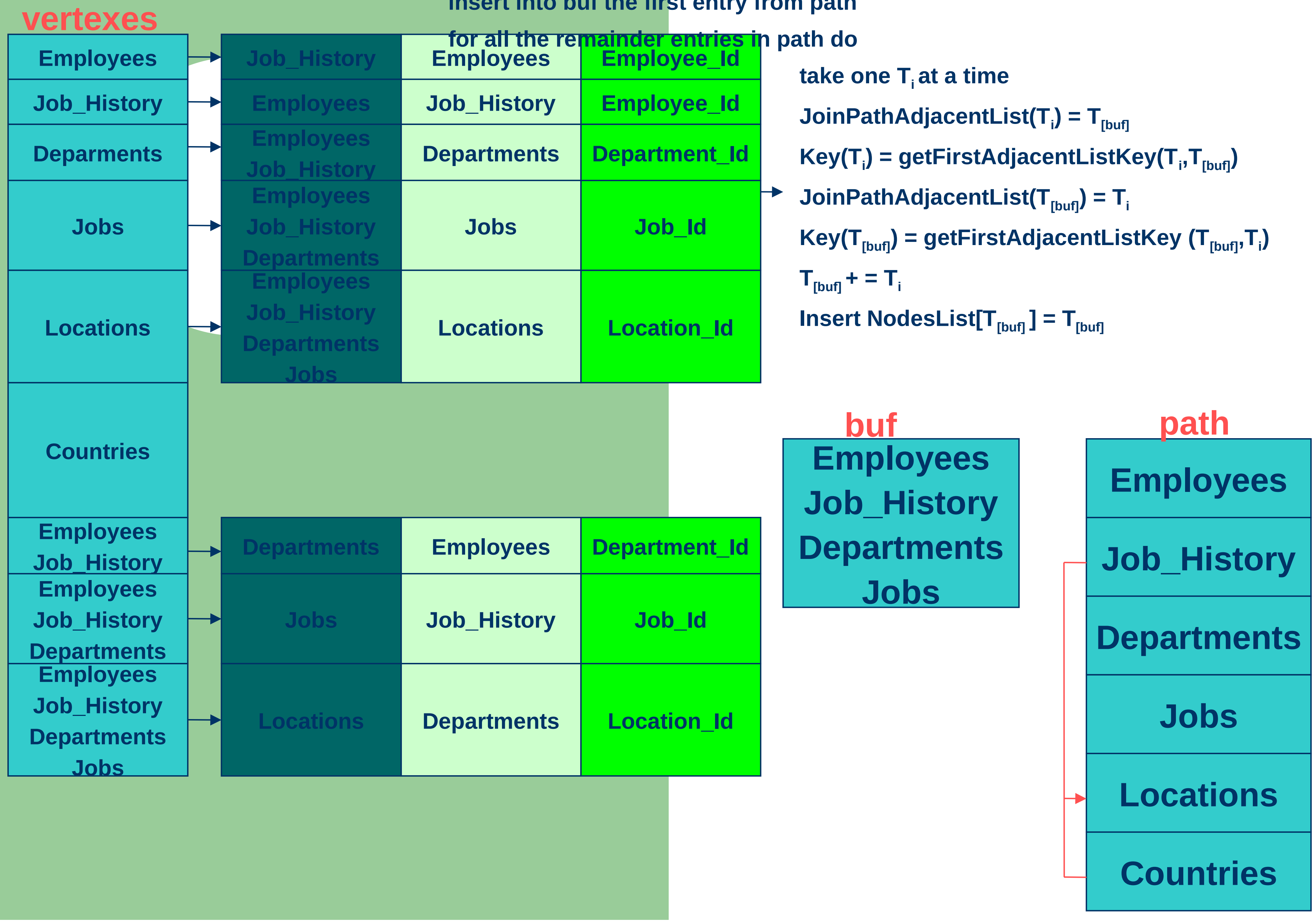
buf

- 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

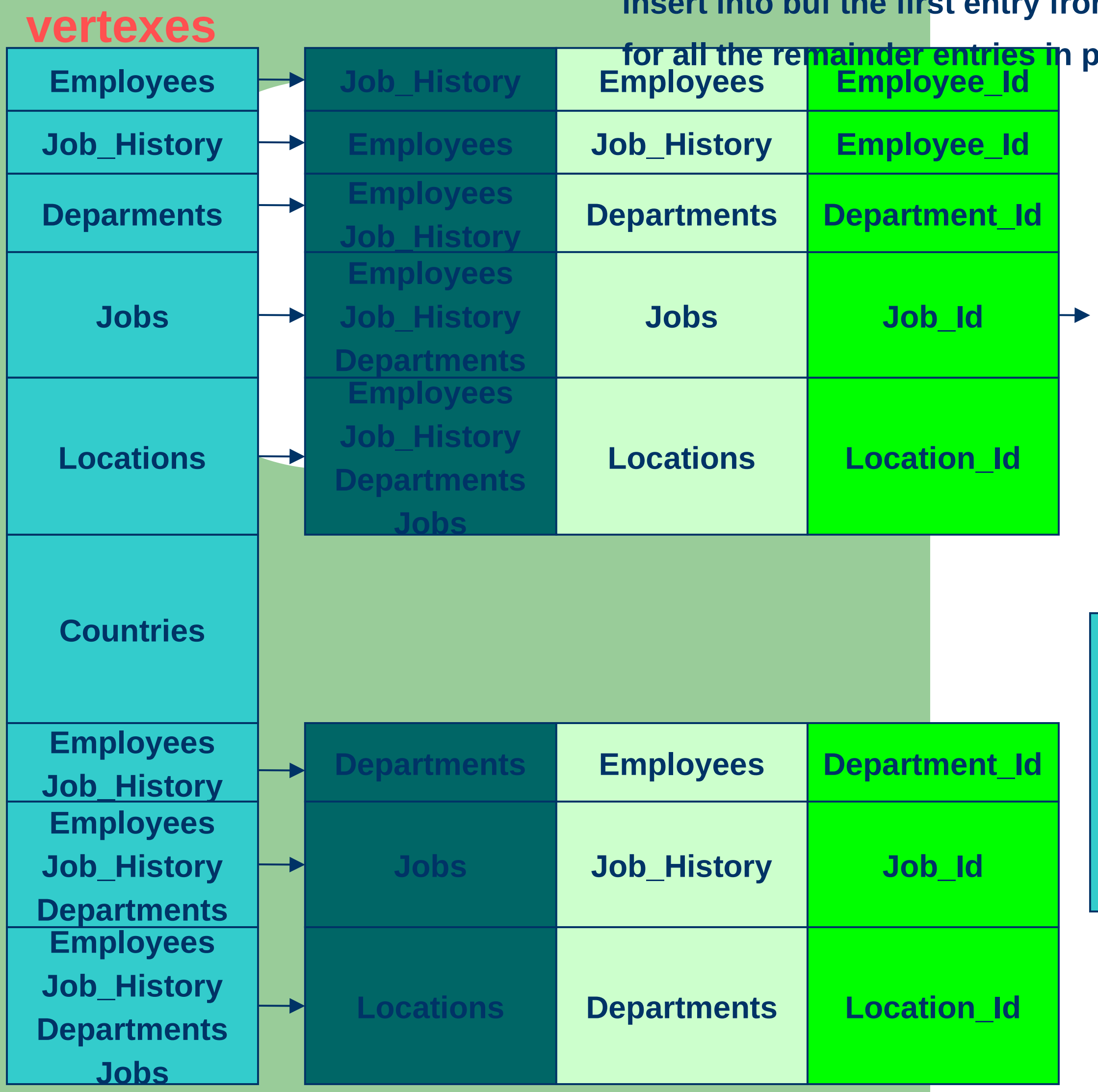
buf

Employees Job_History Departments Jobs

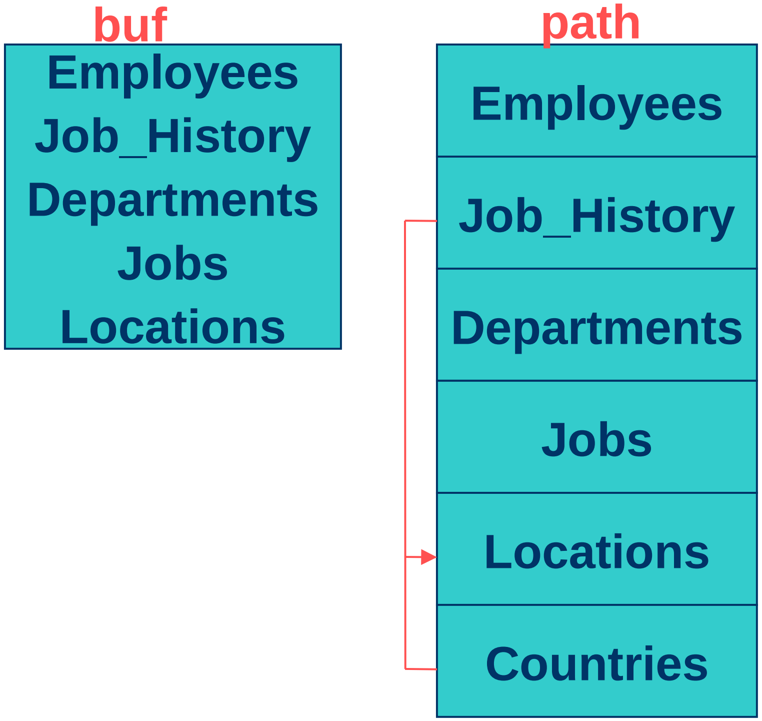
path

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



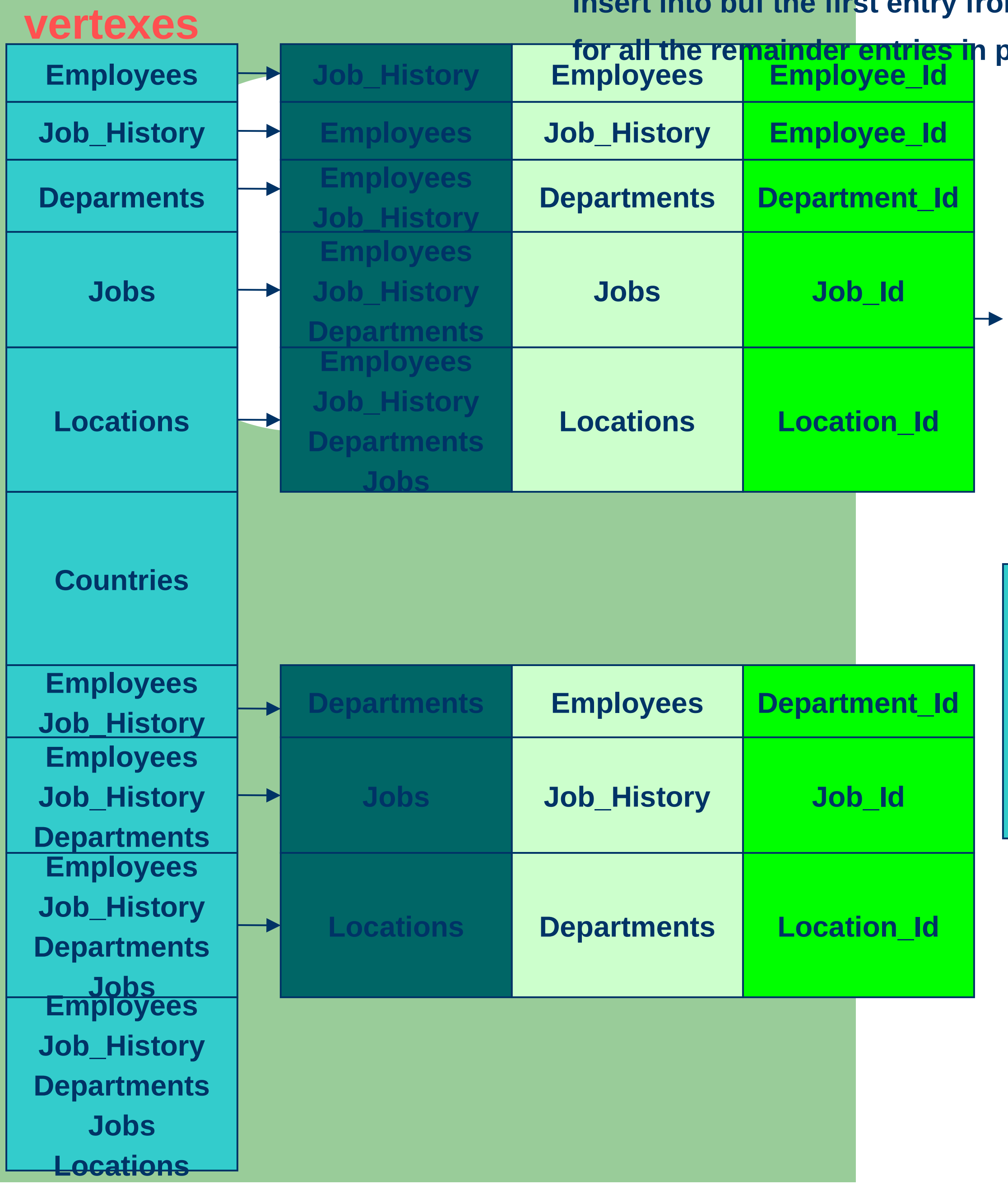
buf

Employees
Job_History
Departments
Jobs
Locations

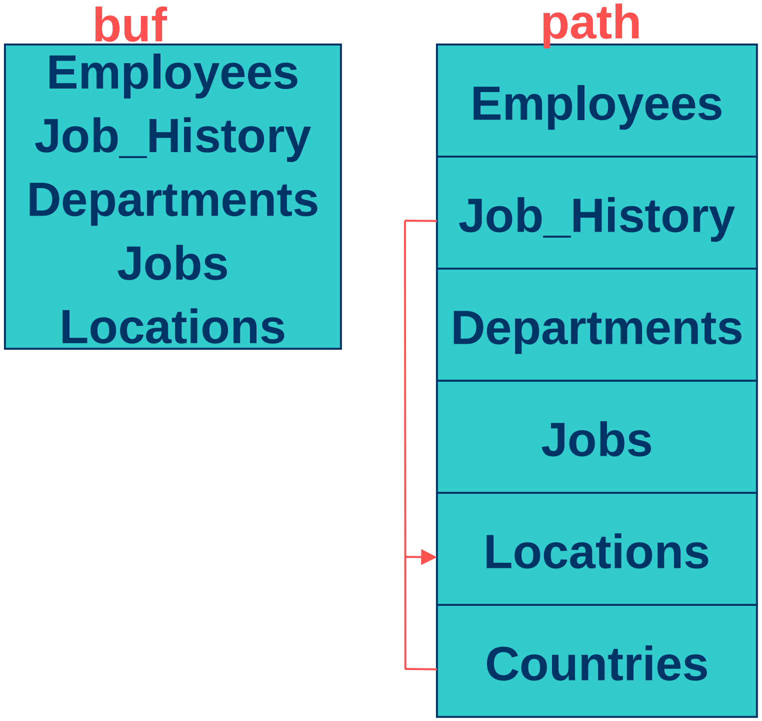
path

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	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
Employees Job_History Departments Jobs Locations			



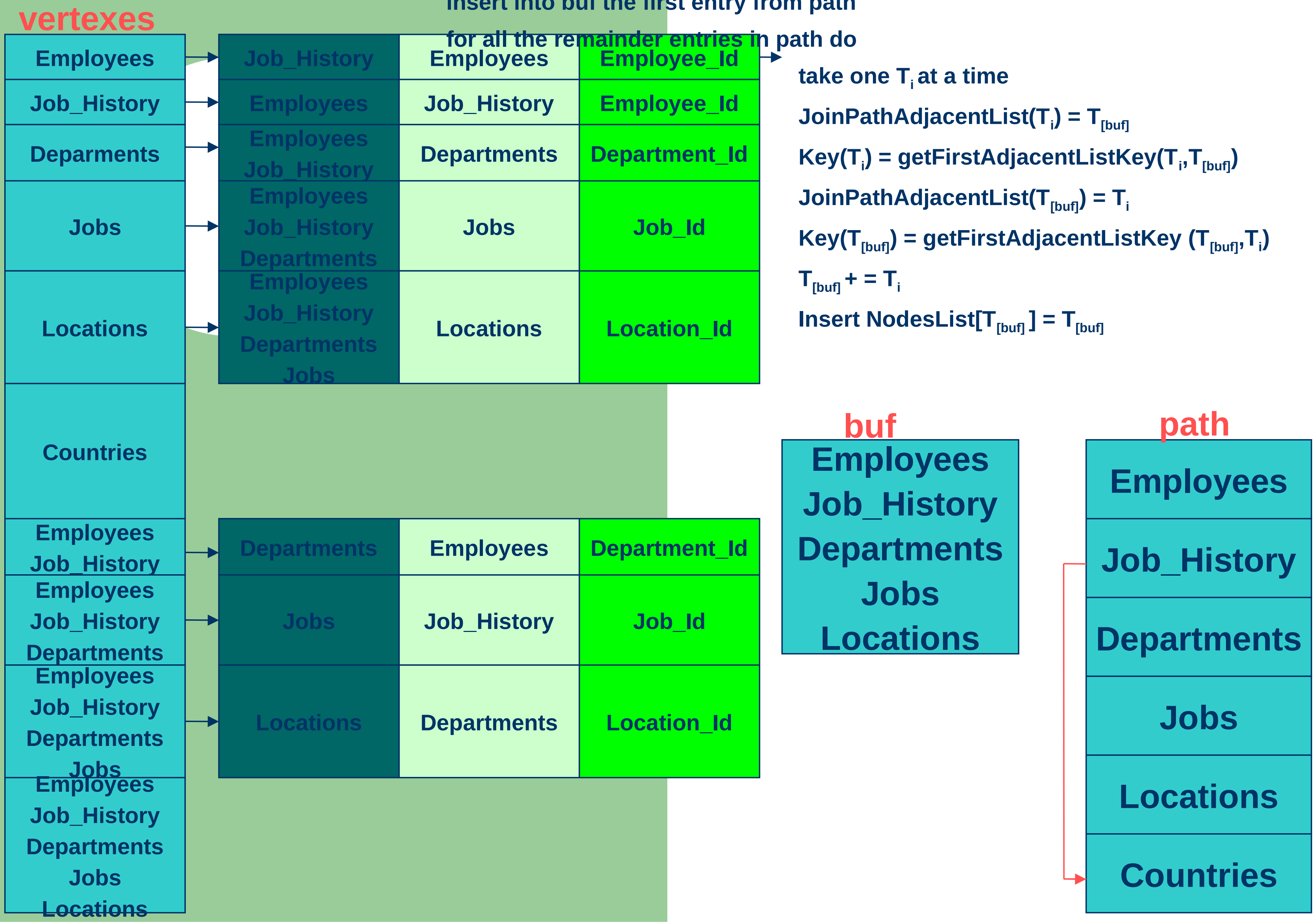
buf

Employees
Job_History
Departments
Jobs
Locations

path

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
Employees Job_History Departments Jobs Locations			

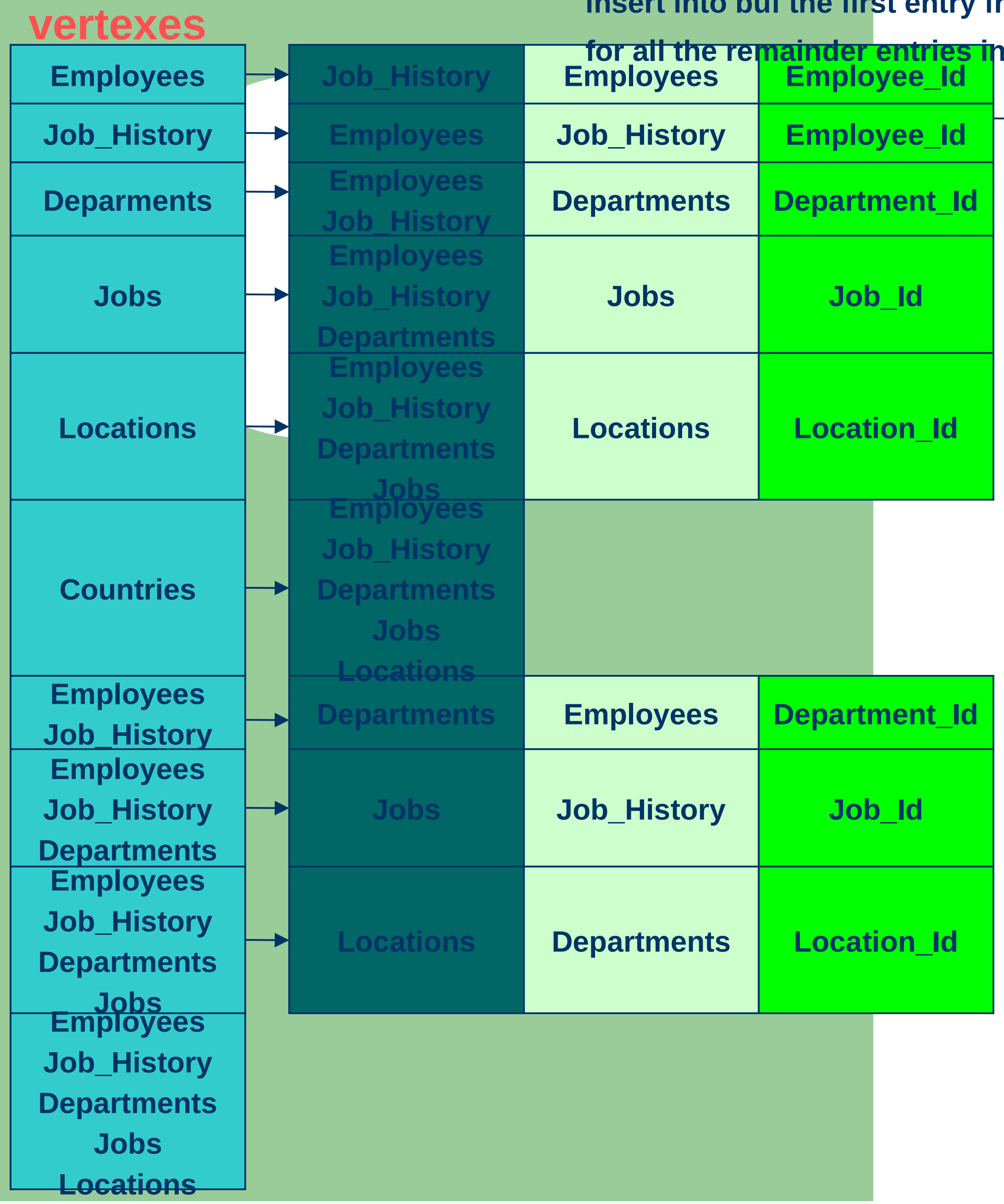
buf

Employees
Job_History
Departments
Jobs
Locations

path

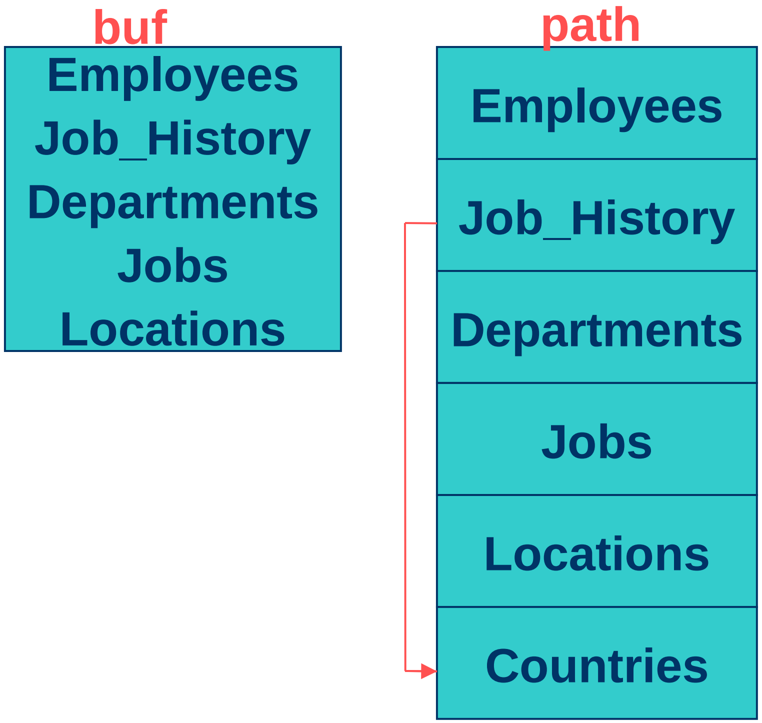
Employees
Job_History
Departments
Jobs
Locations
Countries

```
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	Employees	Employee_Id
Job_History	Employees	Job_History	Employee_Id
Departments	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 Jobs Locations		
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
Employees Job_History Departments Jobs Locations			



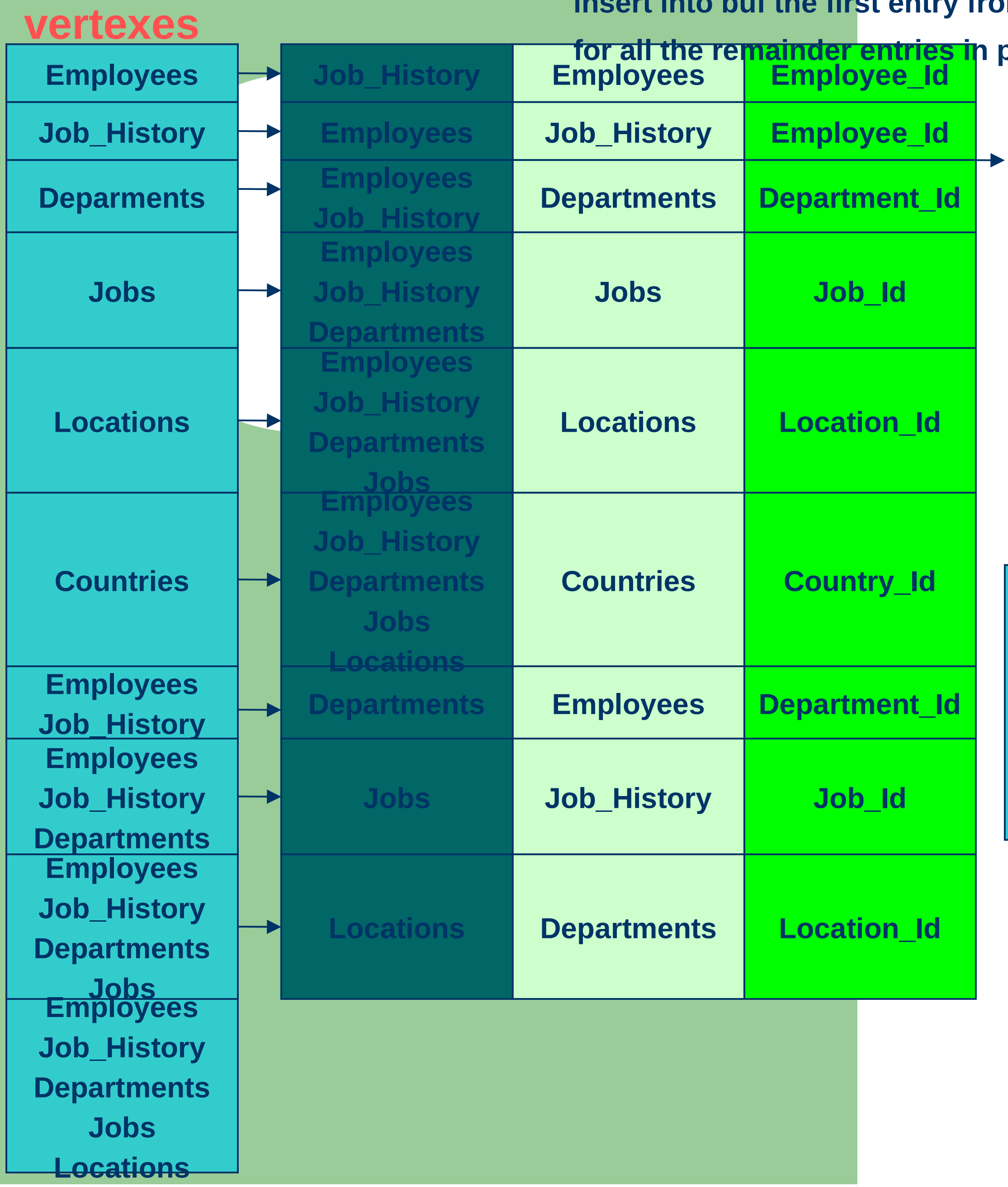
buf

Employees
Job_History
Departments
Jobs
Locations

path

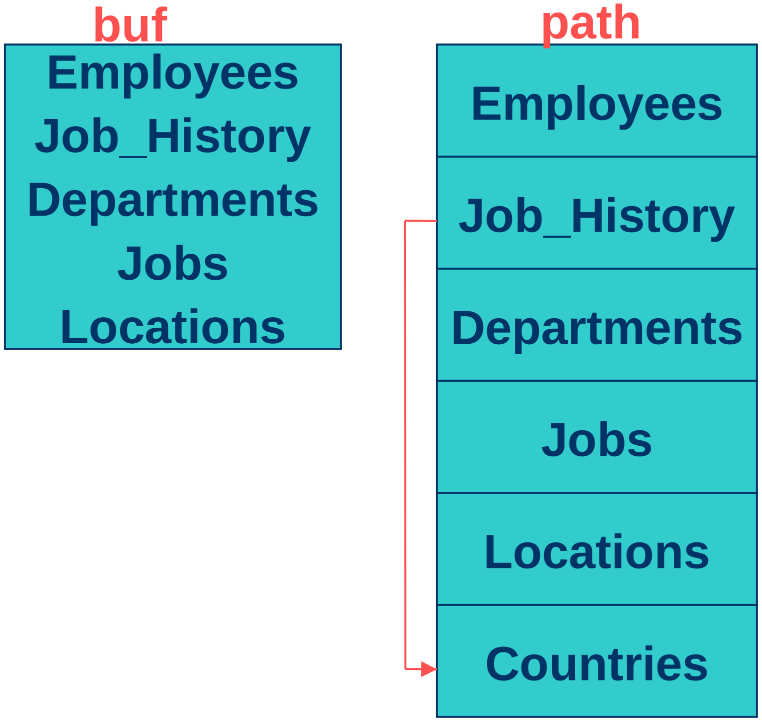
Employees
Job_History
Departments
Jobs
Locations
Countries

```
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	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 Jobs Locations	Countries	Country_Id
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
Employees Job_History Departments Jobs Locations			



buf

Employees
Job_History
Departments
Jobs
Locations

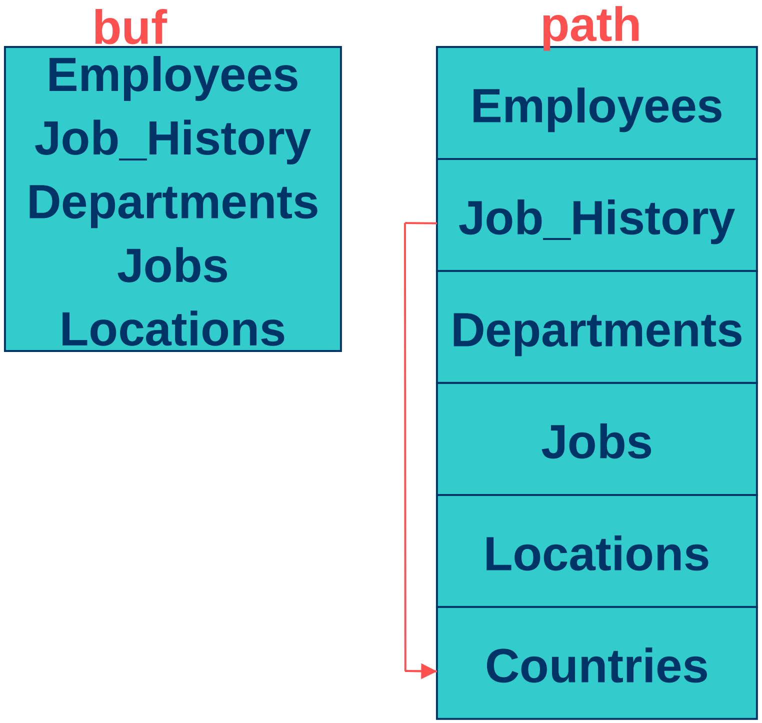
path

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	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 Jobs Locations	Countries	Country_Id
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
Employees Job_History Departments Jobs Locations	Countries		



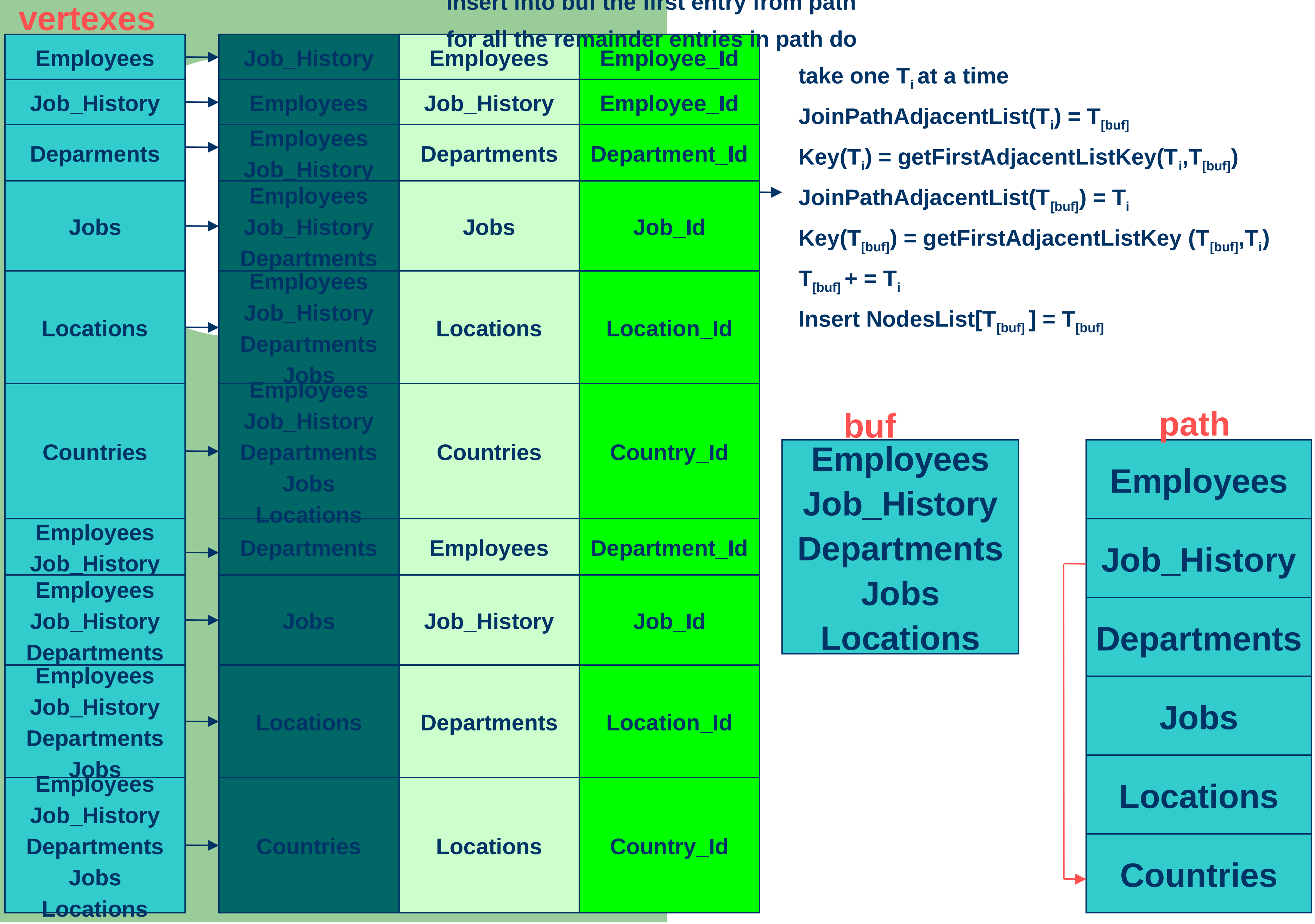
buf

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	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 Jobs Locations	Countries	Country_Id
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
Employees Job_History Departments Jobs Locations	Countries	Locations	Country_Id

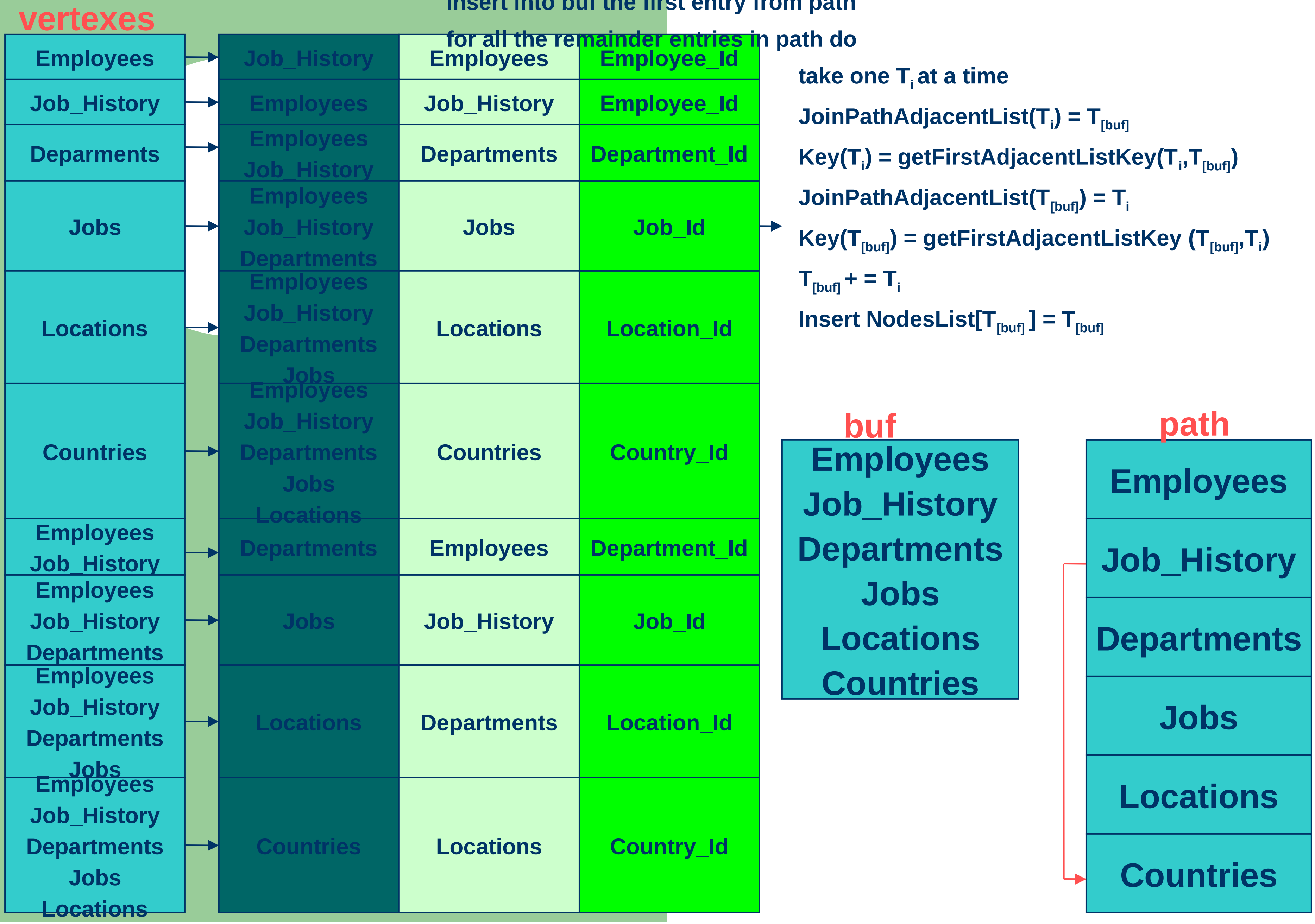
buf

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	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 Jobs Locations	Countries	Country_Id
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
Employees Job_History Departments Jobs Locations	Countries	Locations	Country_Id

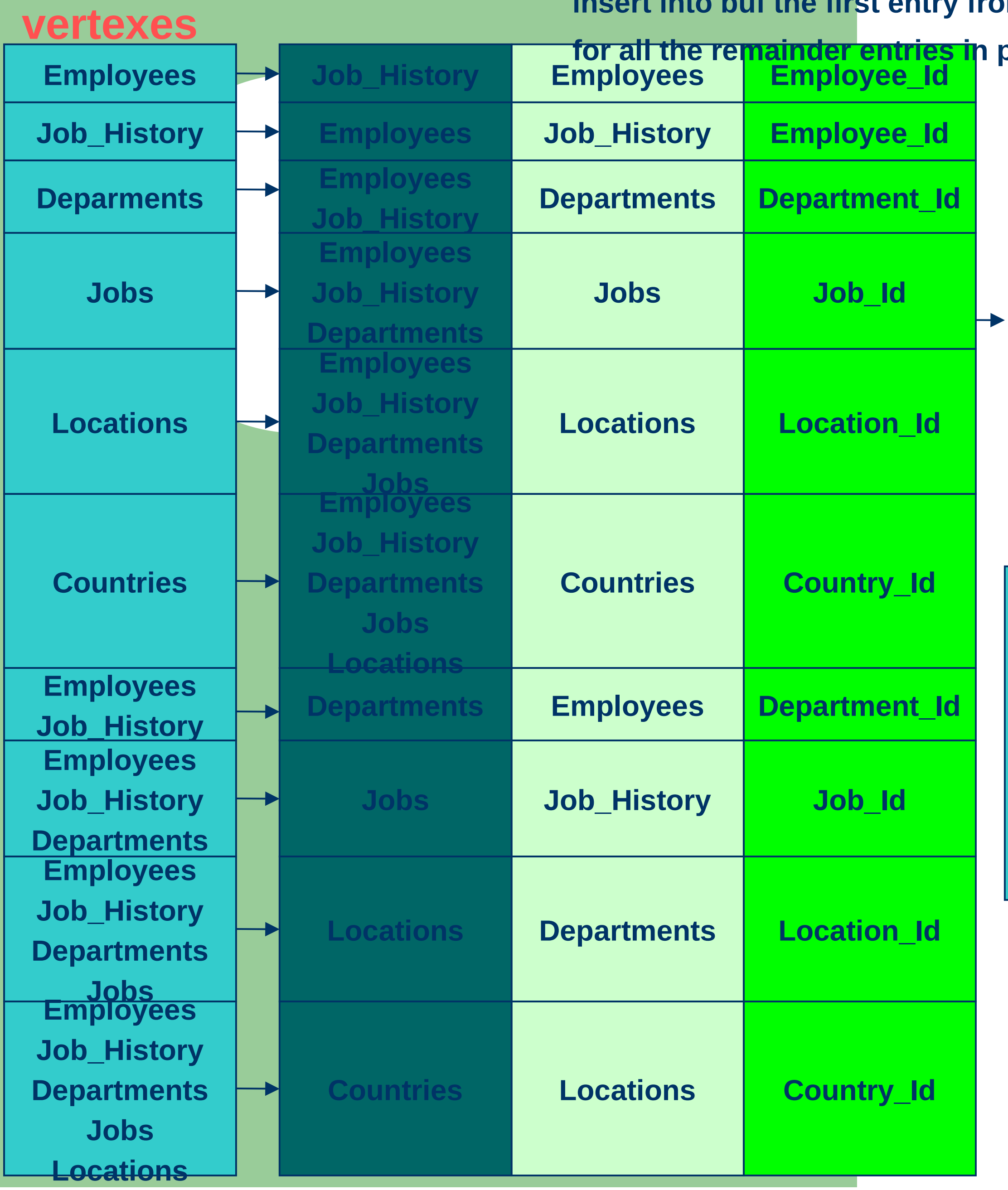
buf

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

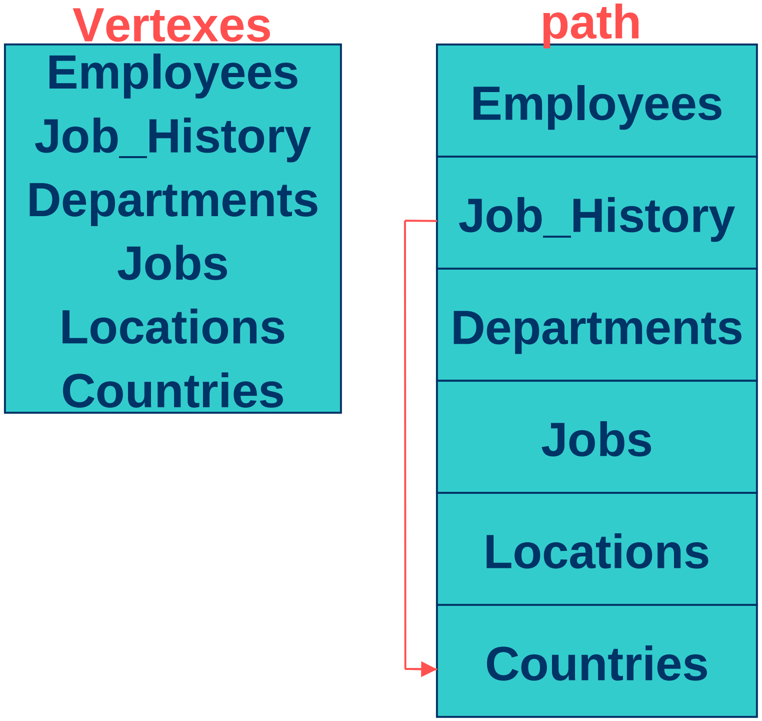
path

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



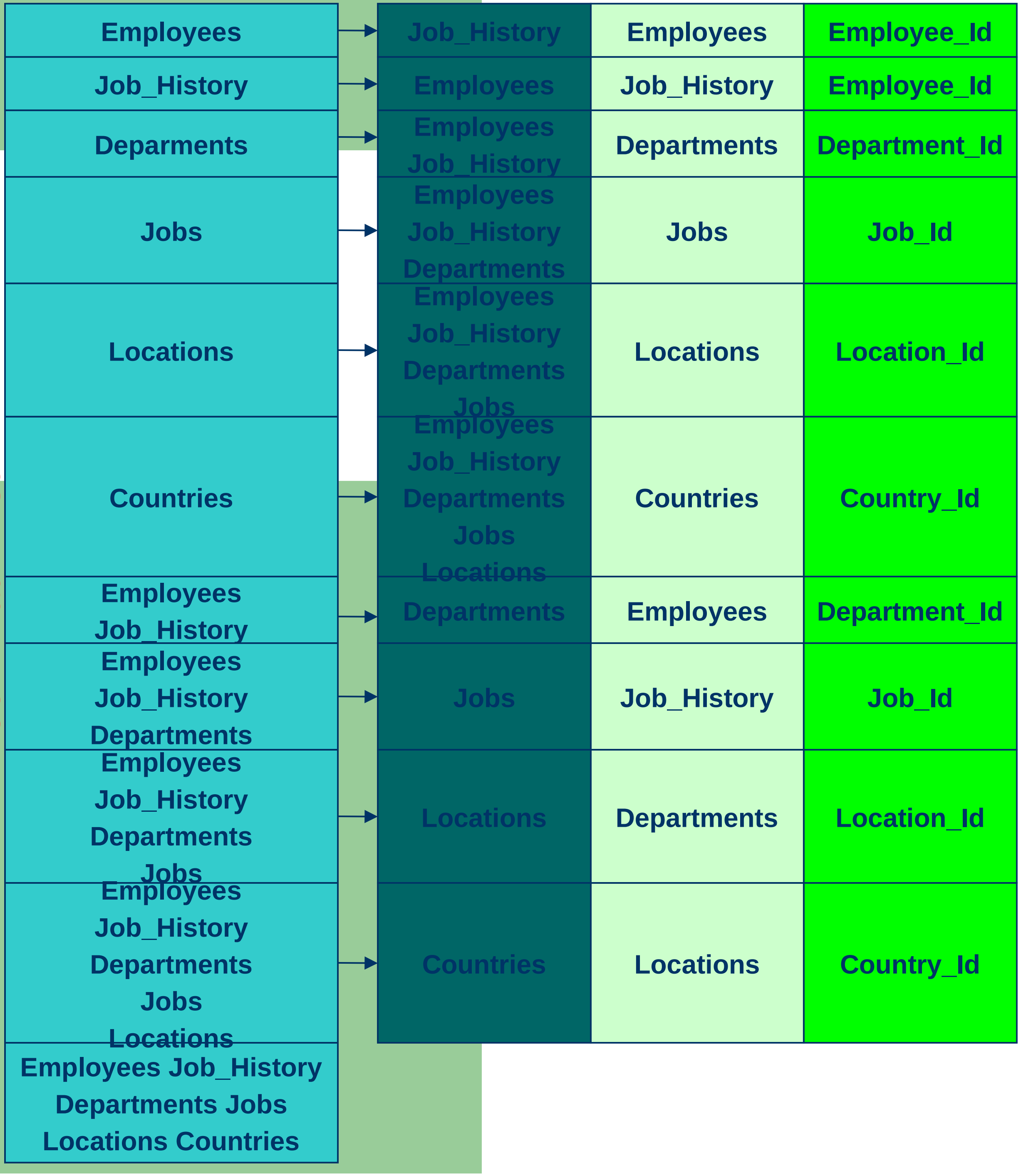
Vertexes

Employees
Job_History
Departments
Jobs
Locations
Countries

path

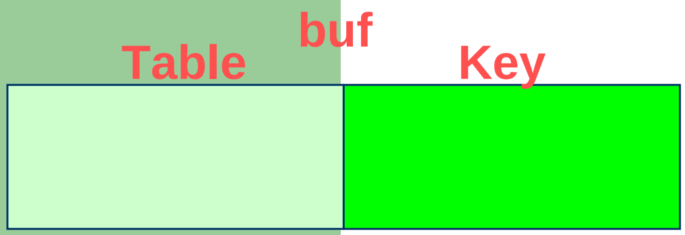
Employees
Job_History
Departments
Jobs
Locations
Countries

Join Path List

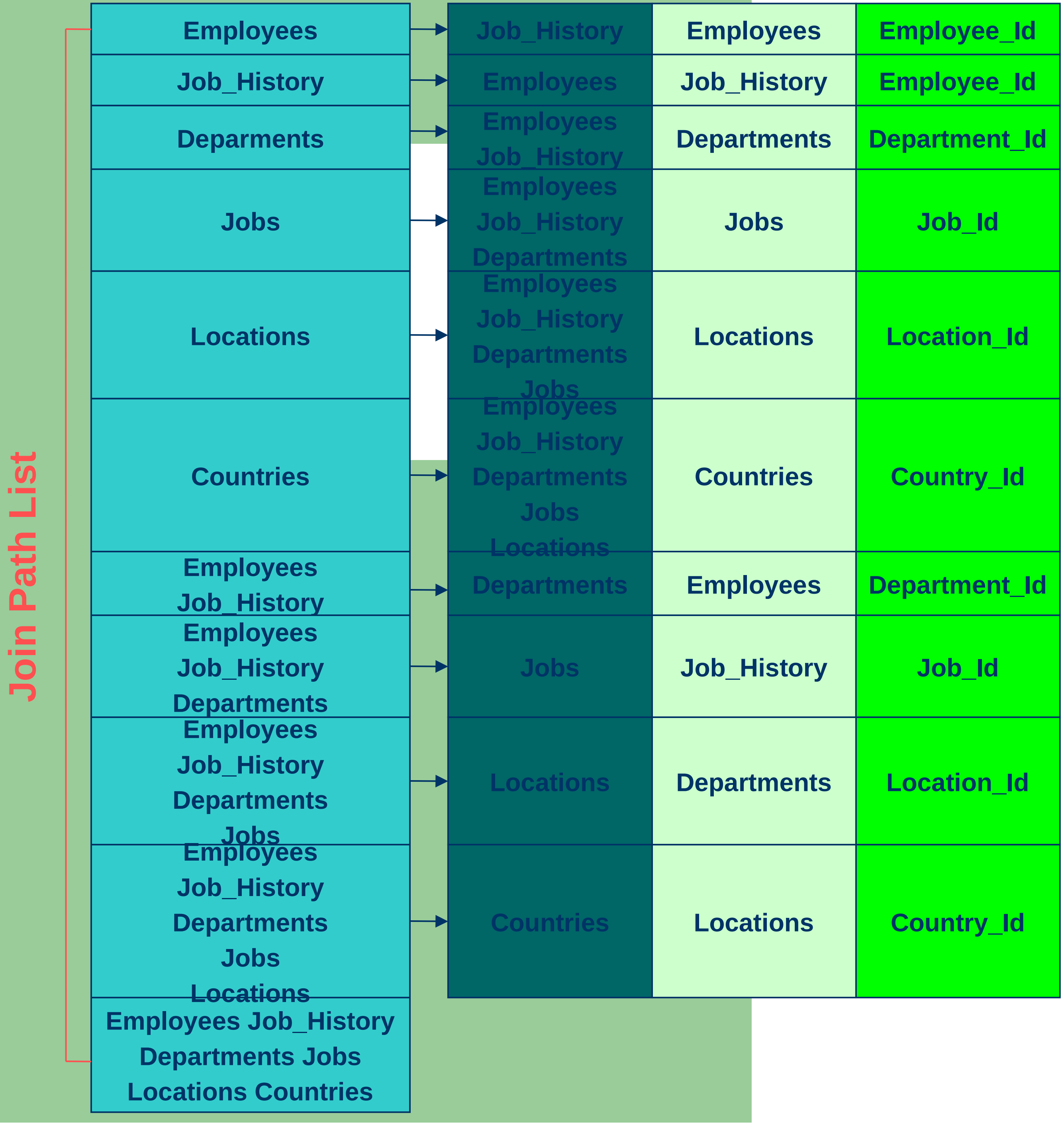


Employees	Job_History	Employees	Employee_Id
Job_History	Employees	Job_History	Employee_Id
Departments	Employees Job_History	Departments	Department_Id
Jobs	Employees Job_History Departments	Jobs	Job_Id
Locations	Employees Job_History Departments Jobs	Locations	Location_Id
Countries	Employees Job_History Departments Jobs Locations	Countries	Country_Id
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
Employees Job_History Departments Jobs Locations	Countries	Locations	Country_Id
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($T_{[i]}$)
```



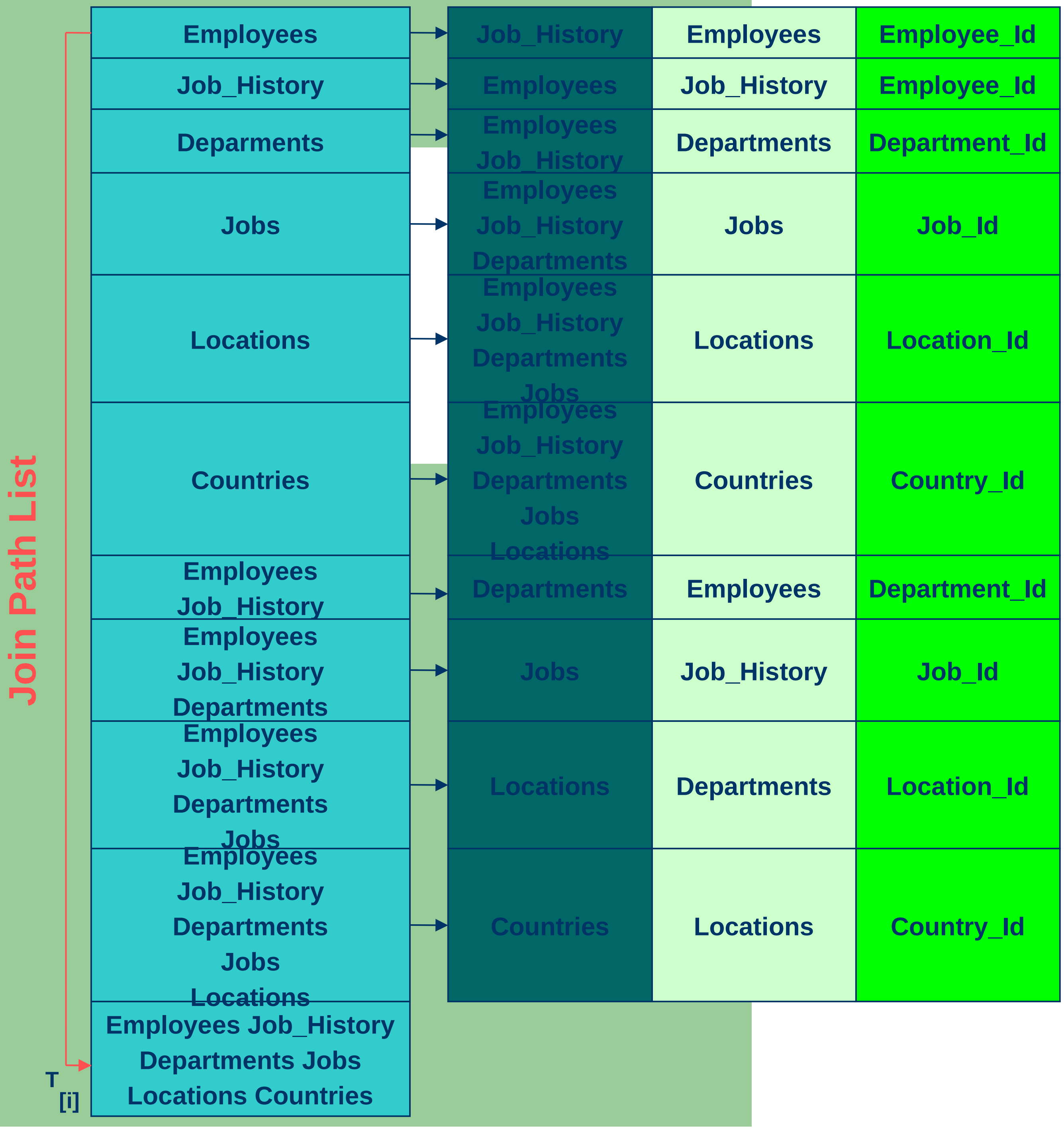
```
   create a structure buf with 2 fields: Table and Key
→  for all the tables in JoinPathList going downward do
       take one $T_{[i]}$ at a time
       for all Base Tables in$T_{[i]}$ do
           take one $T_k$ at a time
           for every buf.Table = $T_k$ do
               if (buf.key != 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

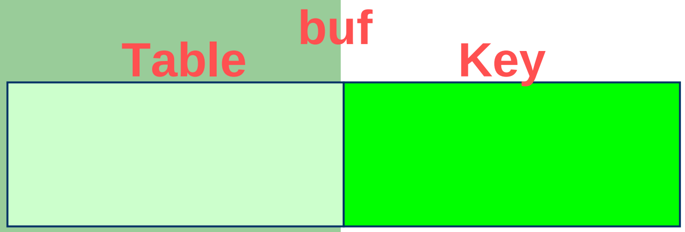
Employees	Job_History	Employees	Employee_Id
Job_History	Employees	Job_History	Employee_Id
Deparments	Employees Job_History	Departments	Department_Id
Jobs	Employees Job_History Departments	Jobs	Job_Id
Locations	Employees Job_History Departments Jobs	Locations	Location_Id
Countries	Employees Job_History Departments Jobs Locations	Countries	Country_Id
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
Employees Job_History Departments Jobs Locations	Countries	Locations	Country_Id
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($T_{[i]}$)
```



Employees	Job_History	Employees	Employee_Id
Job_History	Employees	Job_History	Employee_Id
Departments	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 Jobs Locations	Countries	Country_Id
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
Employees Job_History Departments Jobs Locations	Countries	Locations	Country_Id
Employees Job_History Departments Jobs Locations Countries			

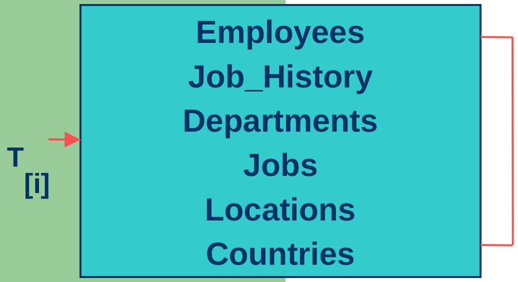
```
create a structure buf with 2 fields: Table and Key
for all the tables in JoinPathList going downward do
    take one $T_{[i]}$ at a time
→   for all Base Tables in$T_{[i]}$ do
        take one $T_k$ at a time
        for every buf.Table = $T_k$ do
            if (buf.key != Key($T_{[i]}$) ) and (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($T_{[i]}$)
```



buf

Table

Key



Employees

Job_History

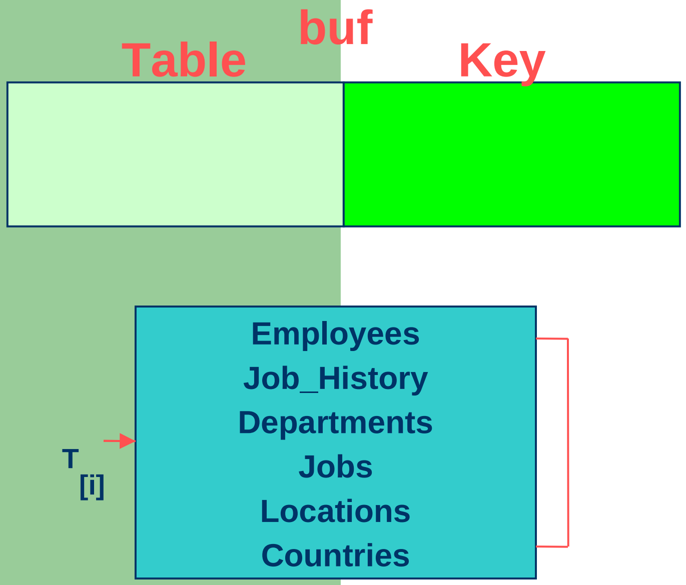
Departments

Jobs

Locations

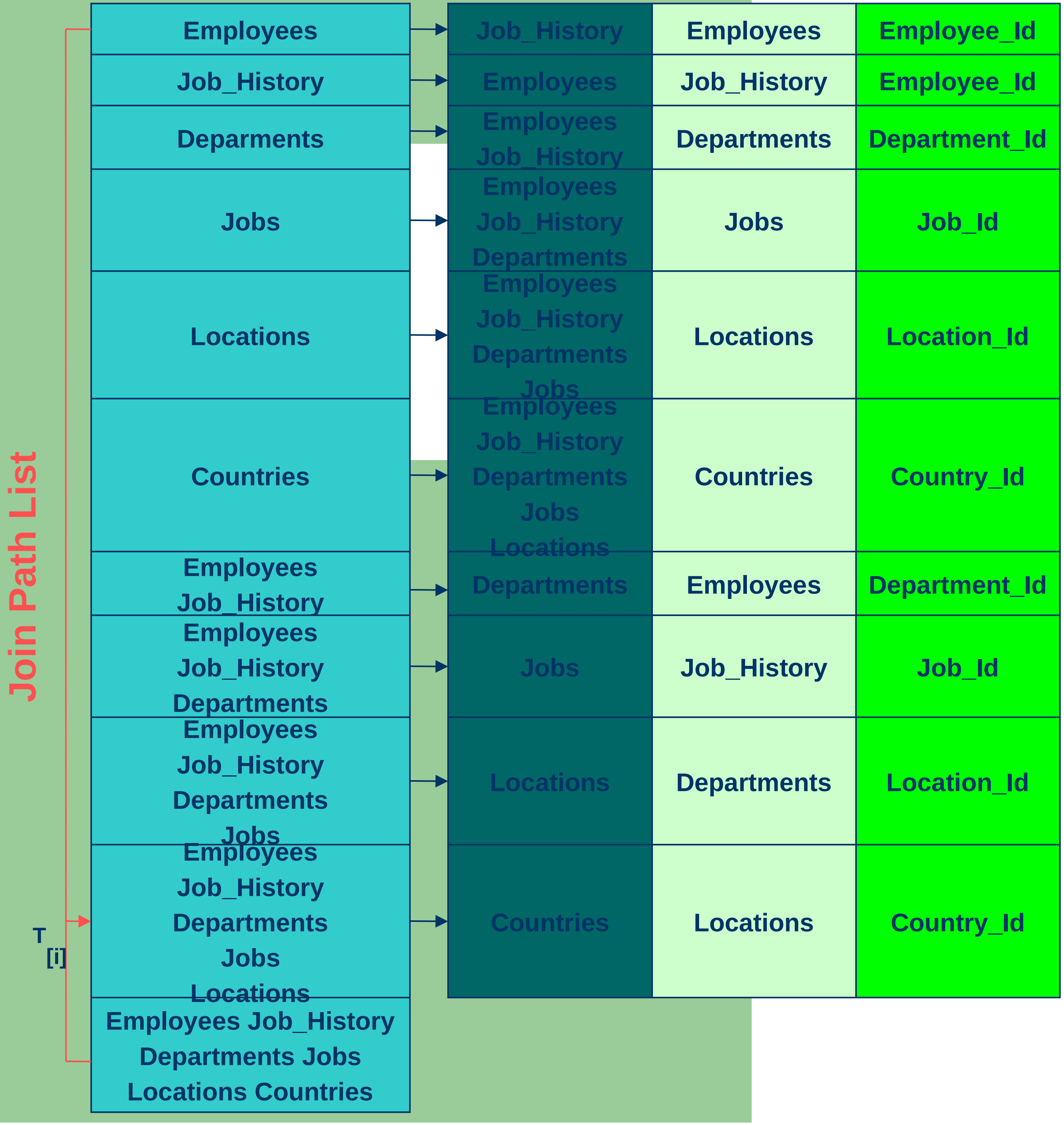
Countries

```
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($T_{[i]}$)
```



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($T_{[i]}$)
```

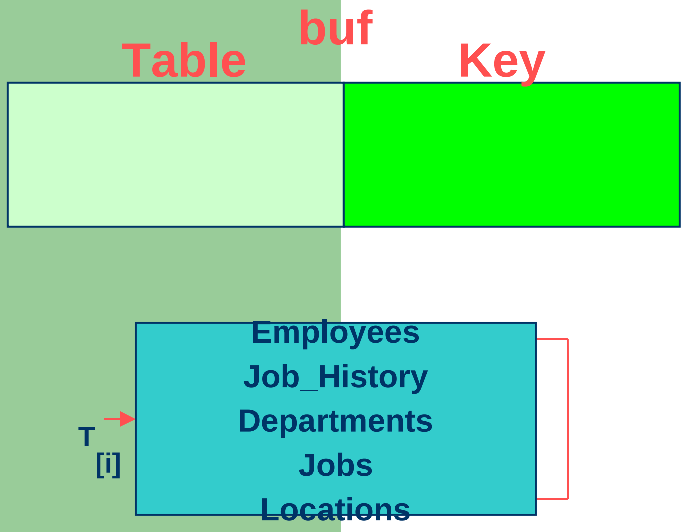


Join Path List

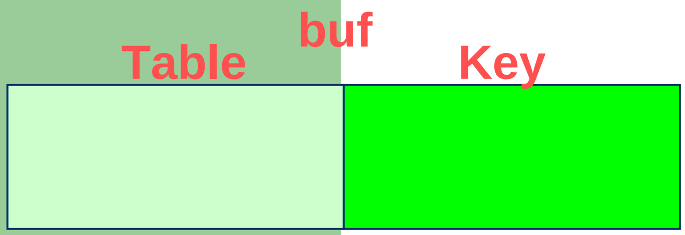
T [i]

Employees	Job_History	Employees	Employee_Id
Job_History	Employees	Job_History	Employee_Id
Departments	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 Jobs Locations	Countries	Country_Id
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
Employees Job_History Departments Jobs Locations	Countries	Locations	Country_Id
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($T_{[i]}$)
```



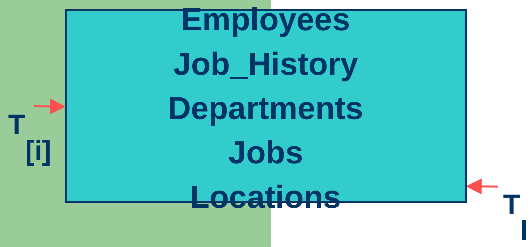
```
create a structure buf with 2 fields: Table and Key
for all the tables in JoinPathList going downward do
    take one $T_{[i]}$ at a time
    for all Base Tables in$T_{[i]}$ do
        take one $T_k$ at a time
        for every buf.Table = $T_k$ do
            if (buf.key != Key($T_{[i]}$) ) and (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



Employees

Job_History

Departments

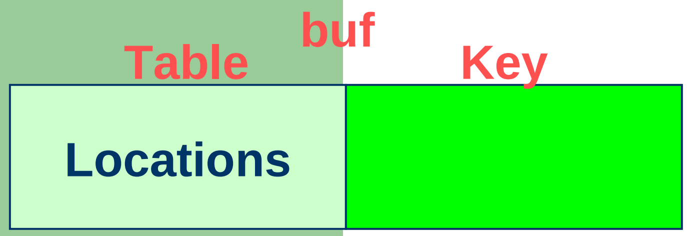
Jobs

Locations

$T_{[i]}$

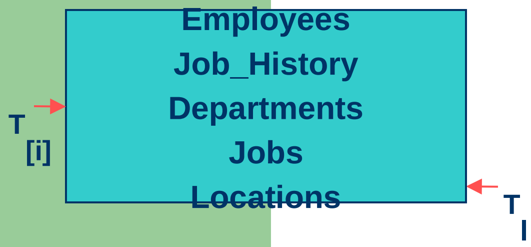
T_l

```
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	

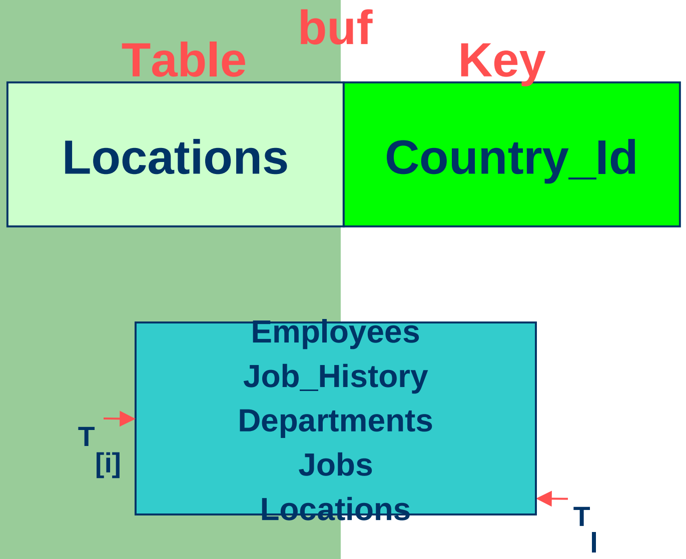


Employees
Job_History
Departments
Jobs
Locations

$T_{[i]}$

T_l

```
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($T_{[i]}$)
```

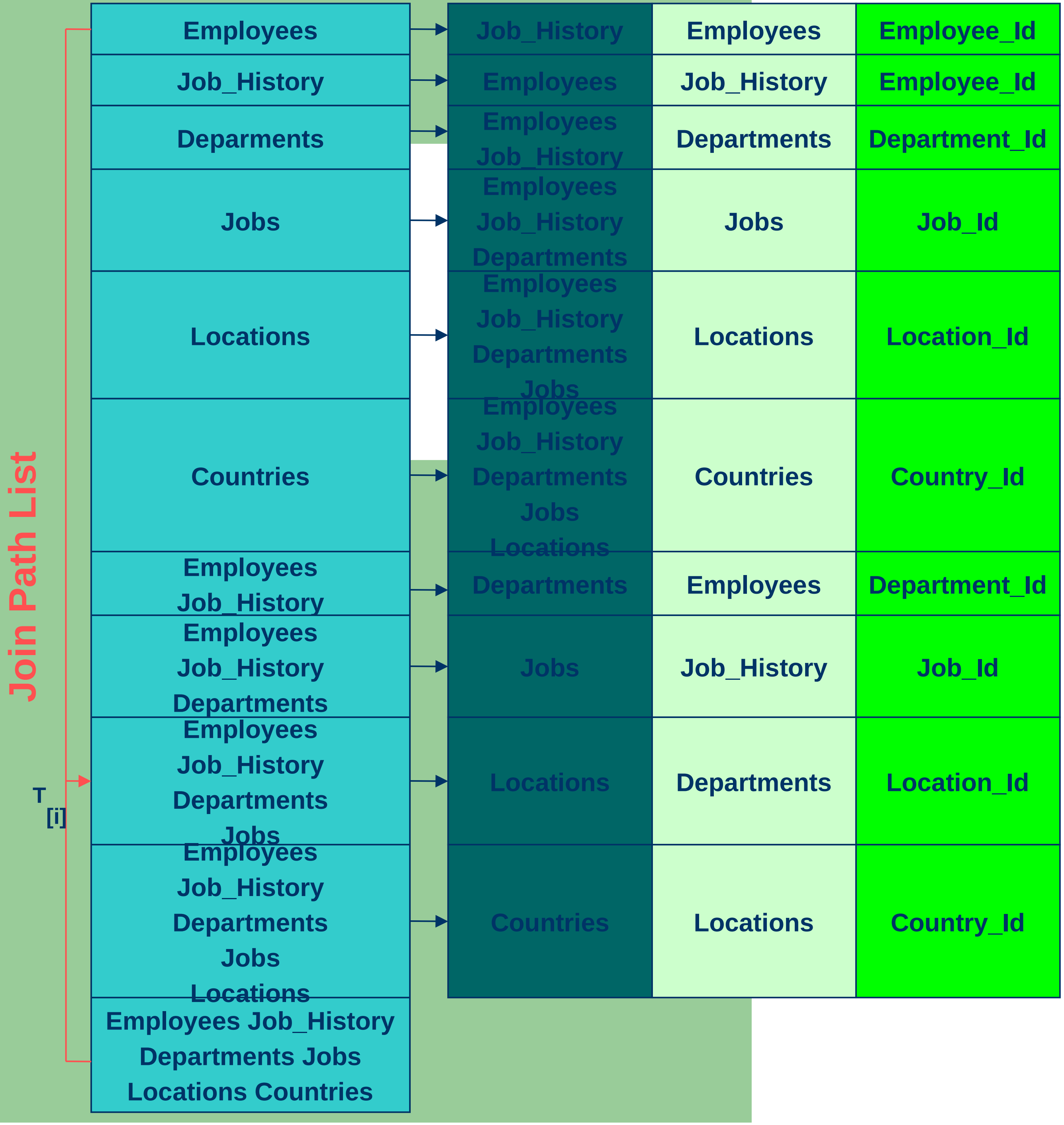


buf

Table	Key
Locations	Country_Id

Employees
Job_History
Departments
Jobs
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($T_{[i]}$)
```

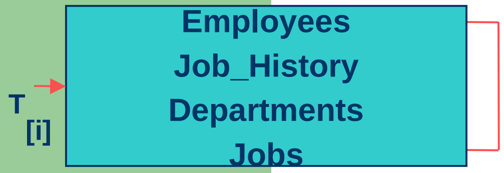


Employees	Job_History	Employees	Employee_Id
Job_History	Employees	Job_History	Employee_Id
Departments	Employees Job_History	Departments	Department_Id
Jobs	Employees Job_History Departments	Jobs	Job_Id
Locations	Employees Job_History Departments Jobs	Locations	Location_Id
Countries	Employees Job_History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

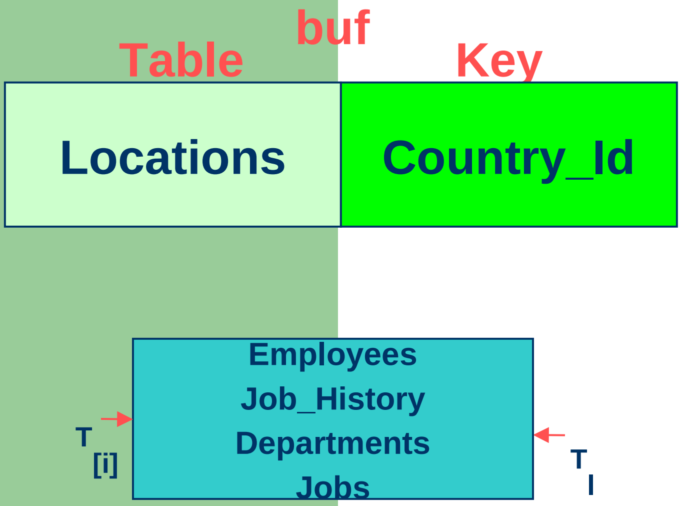
```
create a structure buf with 2 fields: Table and Key
for all the tables in JoinPathList going downward do
    take one $T_{[i]}$ at a time
→   for all Base Tables in$T_{[i]}$ do
        take one $T_k$ at a time
        for every buf.Table = $T_k$ do
            if (buf.key != Key($T_{[i]}$) ) and (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($T_{[i]}$)
```

buf

Table	Key
Locations	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($T_{[i]}$)
```



buf

Table	Key
Locations	Country_Id

$T_{[i]}$

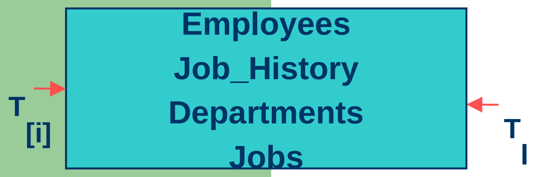
Employees
Job_History
Departments
Jobs

T_l

```
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

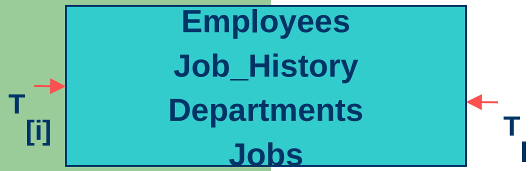
Table	Key
Locations	Country_Id
Departments	



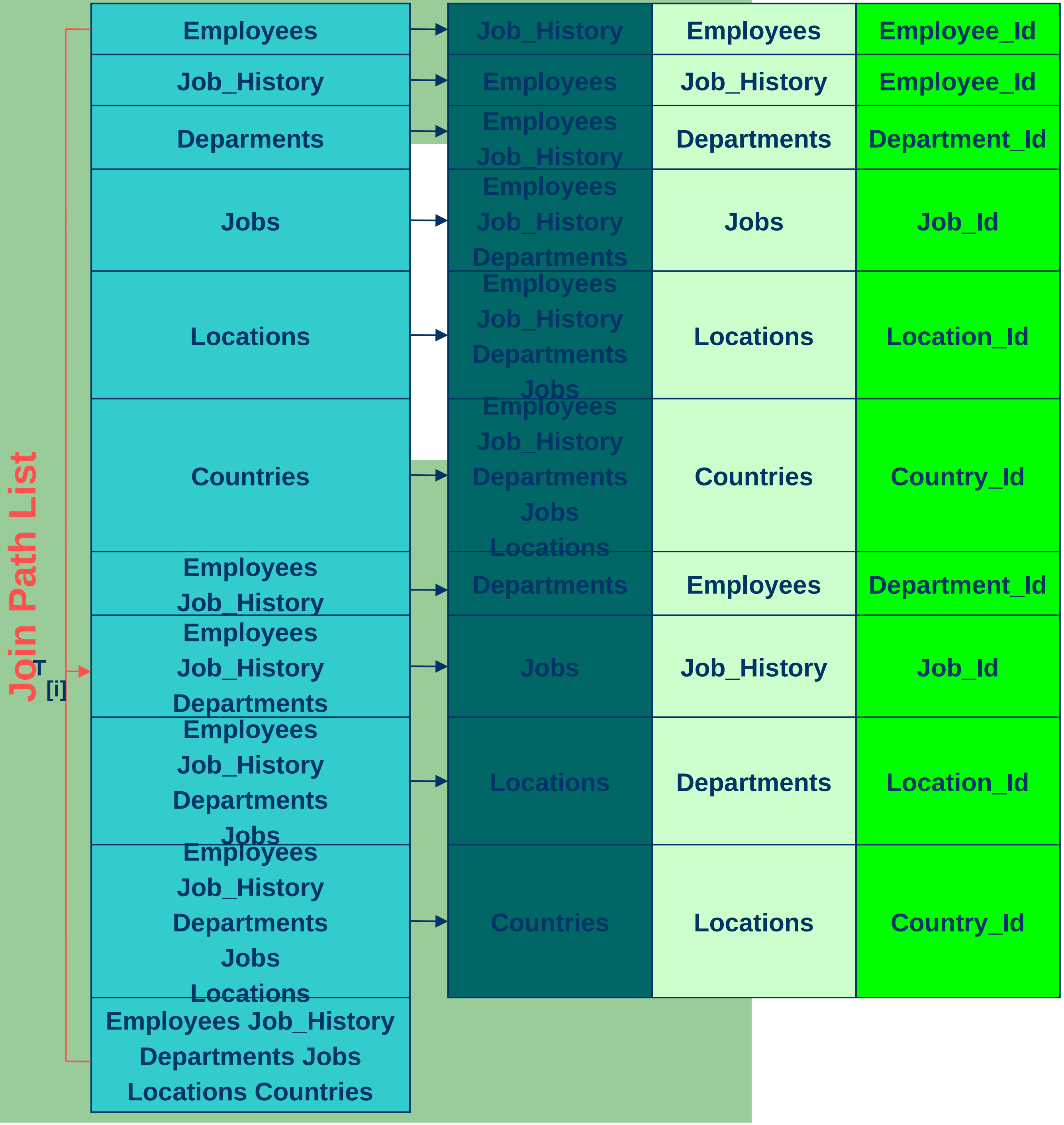
```
create a structure buf with 2 fields: Table and Key
for all the tables in JoinPathList going downward do
    take one $T_{[i]}$ at a time
    for all Base Tables in$T_{[i]}$ do
        take one $T_k$ at a time
        for every buf.Table = $T_k$ do
            if (buf.key != Key($T_{[i]}$) ) and (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



```
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

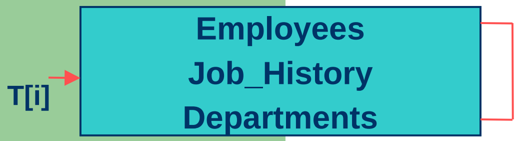
$T_{[i]}$

Employees	Job_History	Employees	Employee_Id
Job_History	Employees	Job_History	Employee_Id
Departments	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 Jobs Locations	Countries	Country_Id
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
Employees Job_History Departments Jobs Locations	Countries	Locations	Country_Id
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($T_{[i]}$)
```

buf

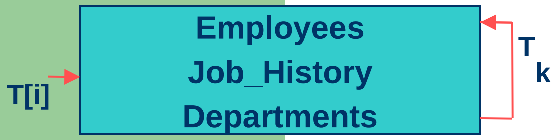
Table	Key
Locations	Country_Id
Departments	Location_Id



```
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

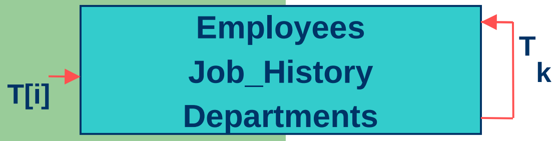
Table	Key
Locations	Country_Id
Departments	Location_Id



```
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

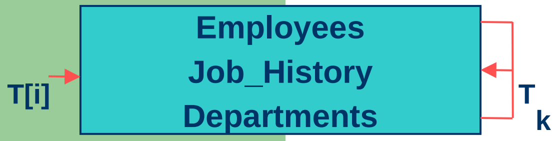
Table	Key
Locations	Country_Id
Departments	Location_Id



```
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

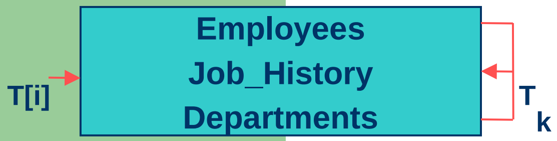
Table	Key
Locations	Country_Id
Departments	Location_Id



```
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

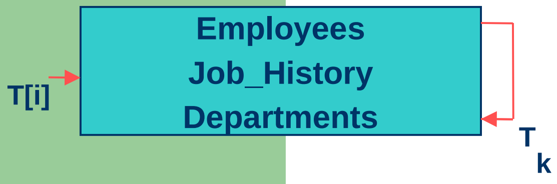
Table	Key
Locations	Country_Id
Departments	Location_Id



```
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

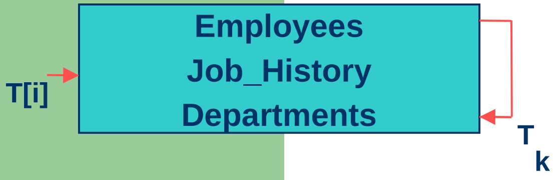
Table	Key
Locations	Country_Id
Departments	Location_Id



```
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



T[i]

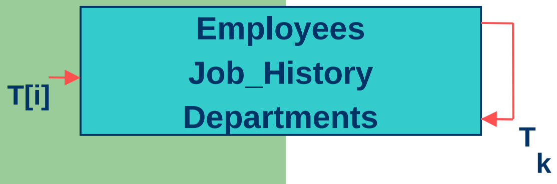
Employees
Job_History
Departments

T_k

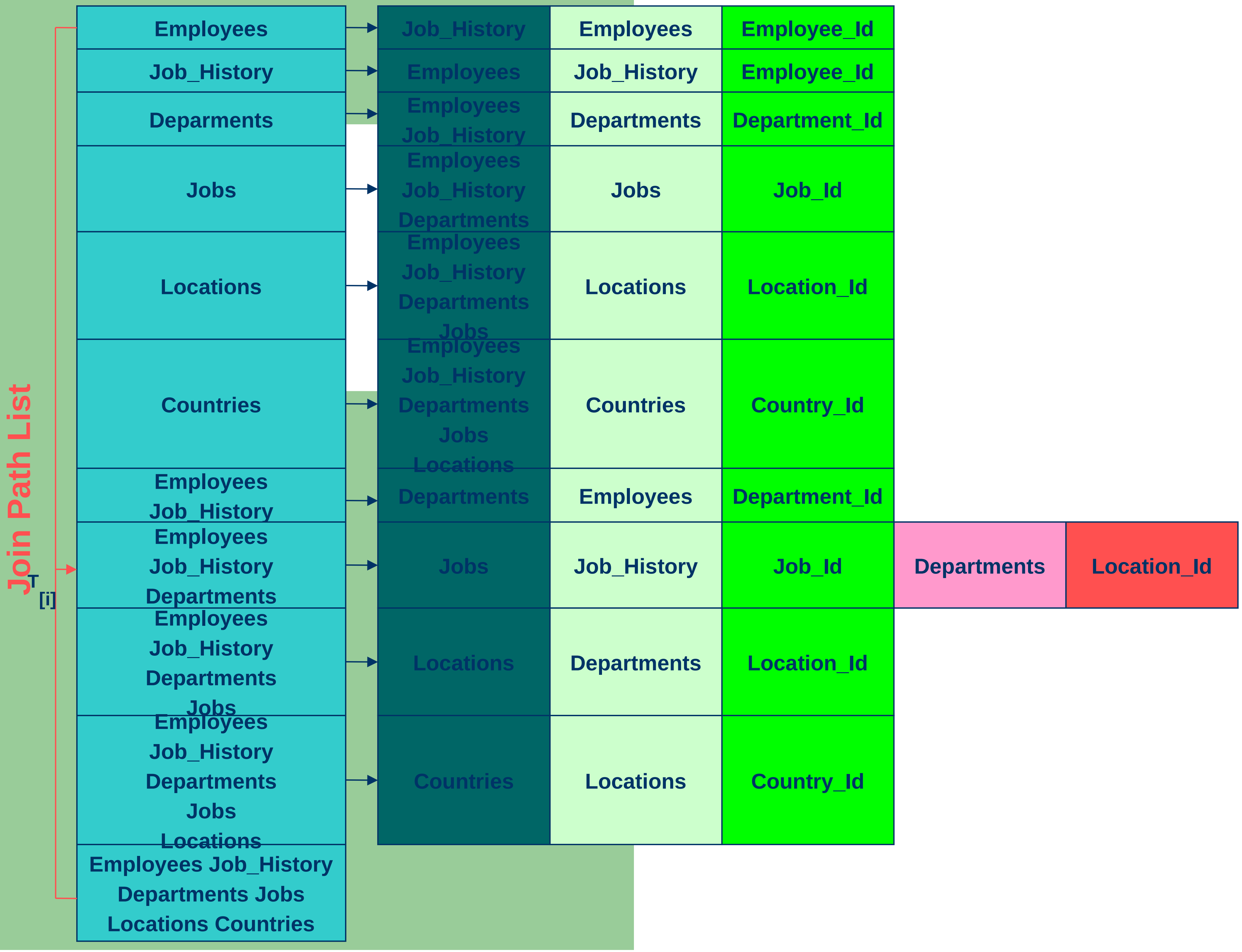
```
create a structure buf with 2 fields: Table and Key
for all the tables in JoinPathList going downward do
    take one $T_{[i]}$ at a time
    for all Base Tables in$T_{[i]}$ do
        take one $T_k$ at a time
        for every buf.Table = $T_k$ do
            if (buf.key != Key($T_{[i]}$) ) and (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



```
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($T_{[i]}$)
```

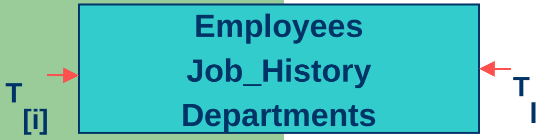


Employees	Job_History	Employees	Employee_Id		
Job_History	Employees	Job_History	Employee_Id		
Departments	Employees Job_History	Departments	Department_Id		
Jobs	Employees Job_History Departments	Jobs	Job_Id		
Locations	Employees Job_History Departments Jobs	Locations	Location_Id		
Countries	Employees Job_History Departments Jobs Locations	Countries	Country_Id		
Employees Job_History	Departments	Employees	Department_Id		
Employees Job_History Departments	Jobs	Job_History	Job_Id	Departments	Location_Id
Employees Job_History Departments Jobs	Locations	Departments	Location_Id		
Employees Job_History Departments Jobs Locations	Countries	Locations	Country_Id		
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($T_{[i]}$)
```

buf

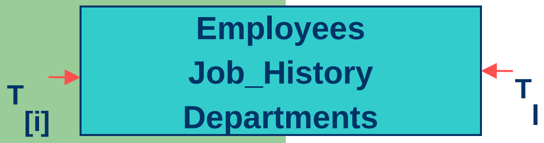
Table	Key
Locations	Country_Id
Departments	Location_Id



```
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

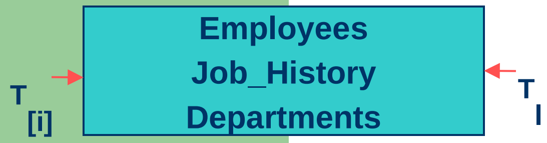
Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	



```
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

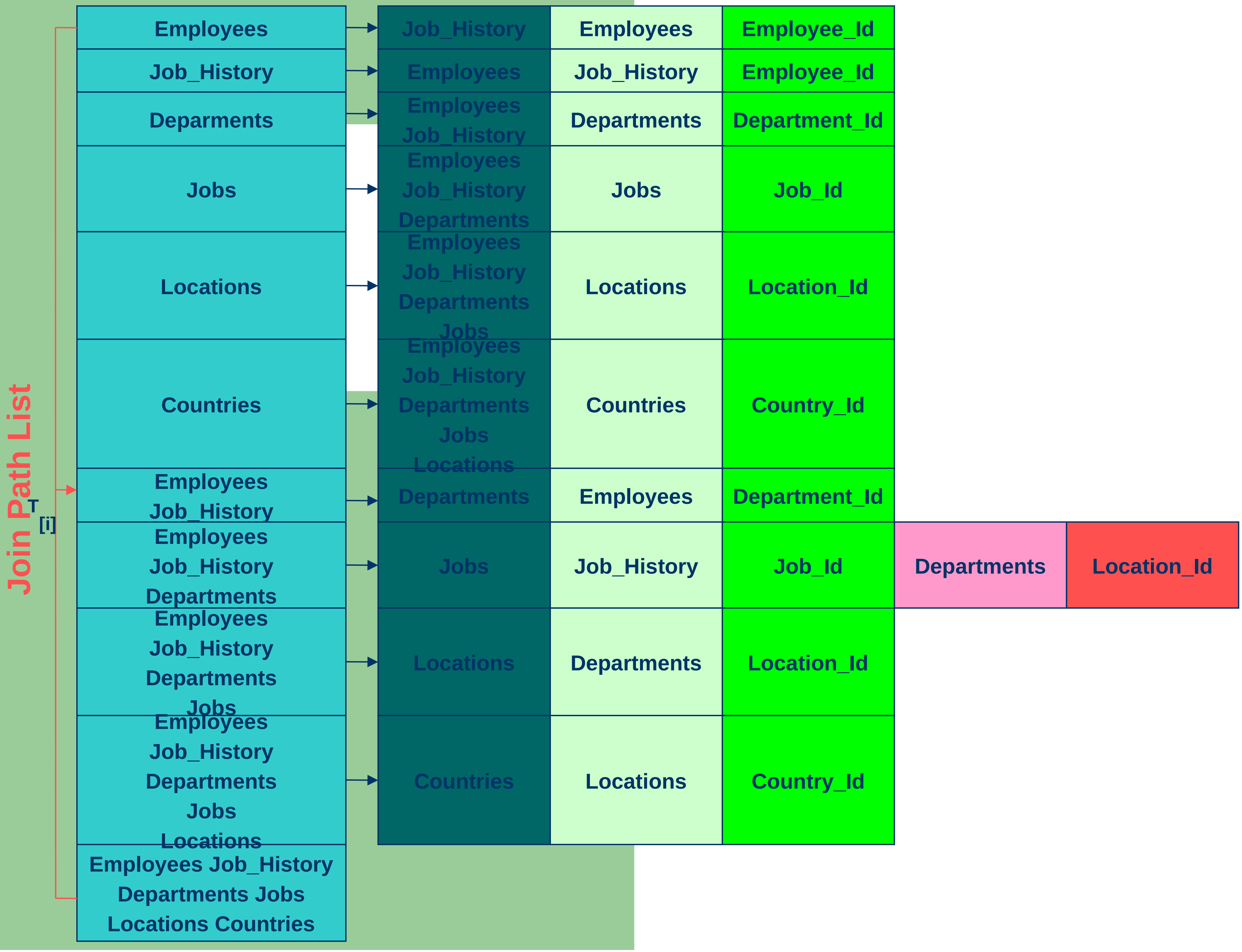


$T_{[i]}$

Employees
Job_History
Departments

T_l

```
  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	Employees	Employee_Id		
Job_History	Employees	Job_History	Employee_Id		
Departments	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 Jobs Locations	Countries	Country_Id		
Employees Job_History	Departments	Employees	Department_Id		
Employees Job_History Departments	Jobs	Job_History	Job_Id	Departments	Location_Id
Employees Job_History Departments Jobs	Locations	Departments	Location_Id		
Employees Job_History Departments Jobs Locations	Countries	Locations	Country_Id		
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($T_{[i]}$)
```

buf

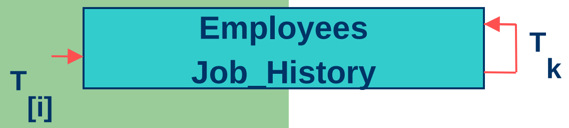
Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id

Employees
Job_History
$T_{[i]}$

```
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

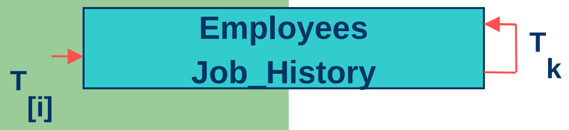
Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id



```
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

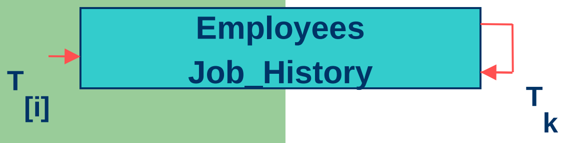
Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id



```
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



Employees

Job_History

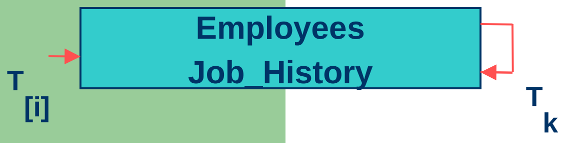
$T_{[i]}$

T_k

```
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

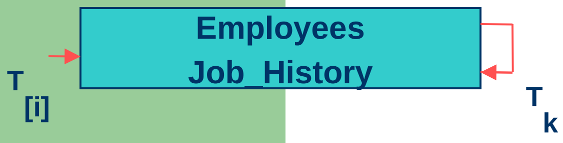
Table	Key
Locations	Country_Id
Departments	Location_Id
→ Job_History	Job_Id



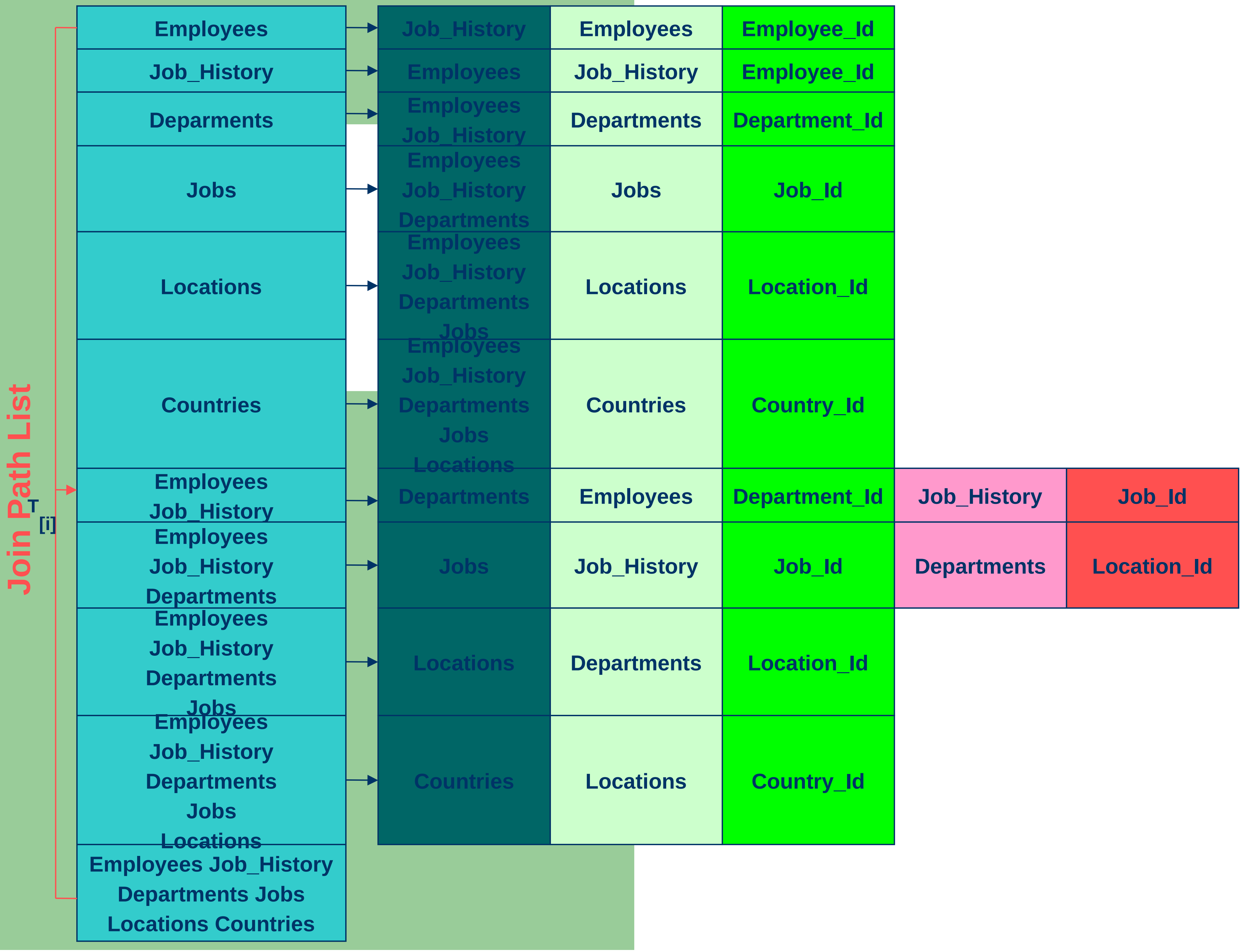
```
create a structure buf with 2 fields: Table and Key
for all the tables in JoinPathList going downward do
          take one $T_{[i]}$ at a time
          for all Base Tables in$T_{[i]}$ do
                    take one $T_k$ at a time
                    for every buf.Table = $T_k$ do
                              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($T_{[i]}$)
```



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 Jobs Locations	Countries	Country_Id		
Employees Job_History	Departments	Employees	Department_Id	Job_History	Job_Id
Employees Job_History Departments	Jobs	Job_History	Job_Id	Departments	Location_Id
Employees Job_History Departments Jobs	Locations	Departments	Location_Id		
Employees Job_History Departments Jobs Locations	Countries	Locations	Country_Id		
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($T_{[i]}$)
```

buf

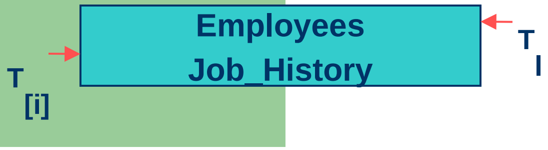
Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id

Employees
Job_History
T_l
$T_{[i]}$

```
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	



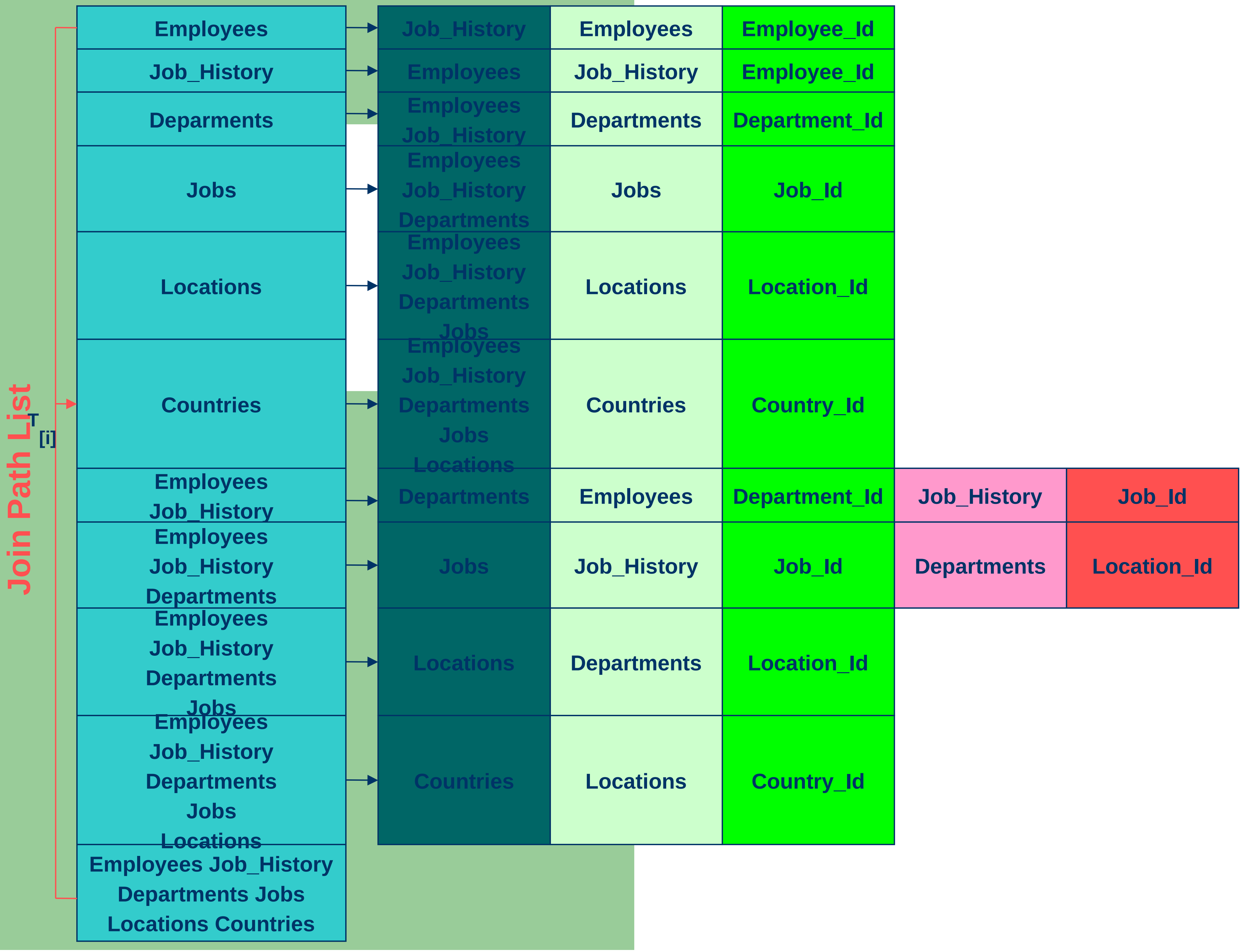
```
create a structure buf with 2 fields: Table and Key
for all the tables in JoinPathList going downward do
    take one $T_{[i]}$ at a time
    for all Base Tables in$T_{[i]}$ do
        take one $T_k$ at a time
        for every buf.Table = $T_k$ do
            if (buf.key != Key($T_{[i]}$) ) and (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($T_{[i]}$)
```

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

Employees
Job_History
$T_{[i]}$
T_l

```
  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	Employees	Employee_Id		
Job_History	Employees	Job_History	Employee_Id		
Departments	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 Jobs Locations	Countries	Country_Id		
Employees Job_History	Departments	Employees	Department_Id	Job_History	Job_Id
Employees Job_History Departments	Jobs	Job_History	Job_Id	Departments	Location_Id
Employees Job_History Departments Jobs	Locations	Departments	Location_Id		
Employees Job_History Departments Jobs Locations	Countries	Locations	Country_Id		
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($T_{[i]}$)
```

buf

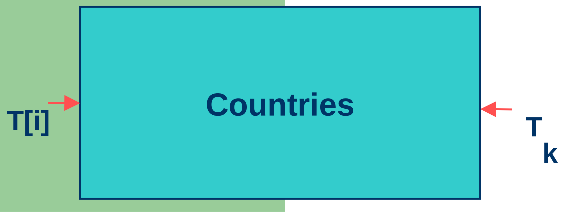
Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

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($T_{[i]}$)
```

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

T[i]
Countries
T
k

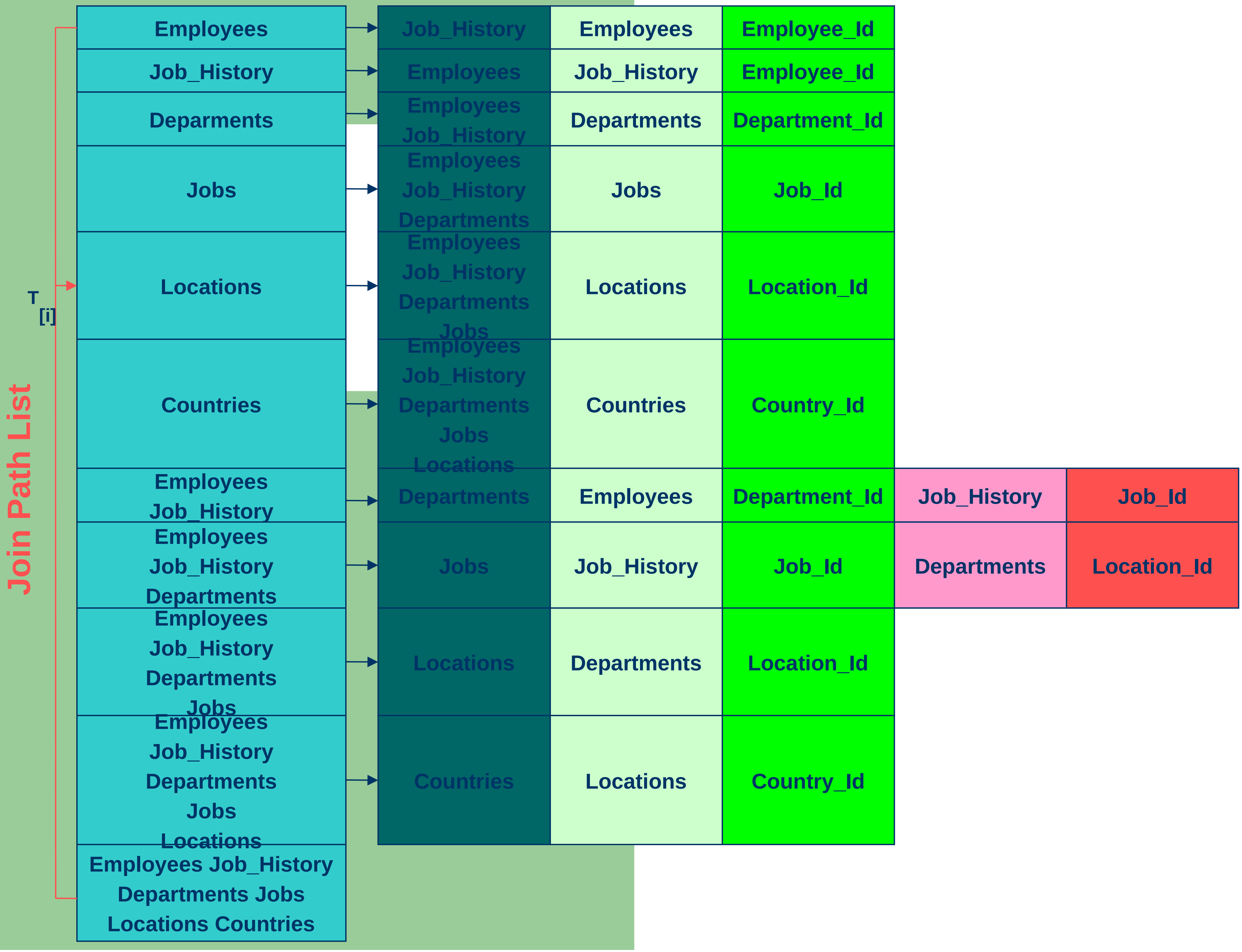
```
create a structure buf with 2 fields: Table and Key
for all the tables in JoinPathList going downward do
    take one $T_{[i]}$ at a time
    for all Base Tables in$T_{[i]}$ do
        take one $T_k$ at a time
        for every buf.Table = $T_k$ do
→           if (buf.key != Key($T_{[i]}$) ) and (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($T_{[i]}$)
```



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 Jobs Locations	Countries	Country_Id		
Employees Job_History	Departments	Employees	Department_Id	Job_History	Job_Id
Employees Job_History Departments	Jobs	Job_History	Job_Id	Departments	Location_Id
Employees Job_History Departments Jobs	Locations	Departments	Location_Id		
Employees Job_History Departments Jobs Locations	Countries	Locations	Country_Id		
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($T_{[i]}$)
```

buf

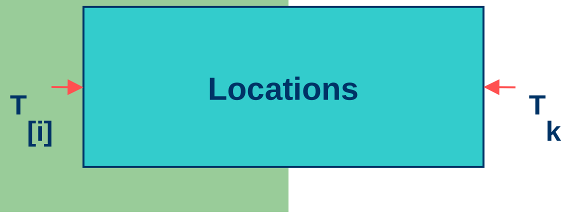
Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

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($T_{[i]}$)
```

buf

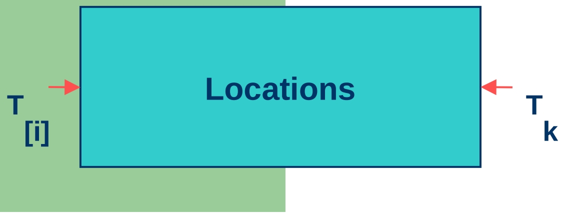
Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id



```
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

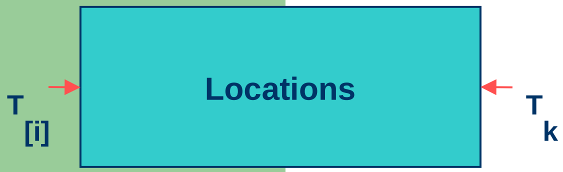
Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id



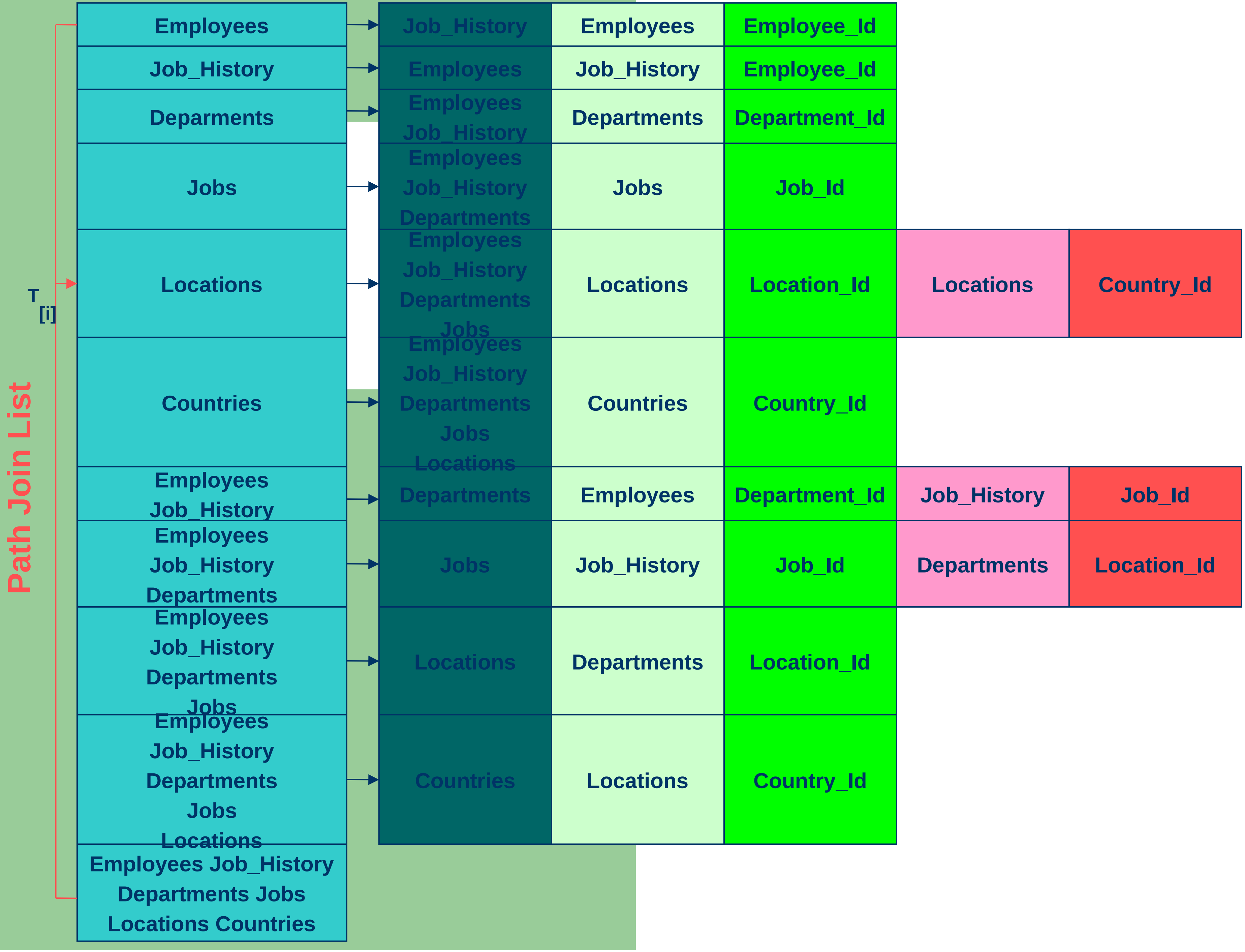
```
create a structure buf with 2 fields: Table and Key
for all the tables in JoinPathList going downward do
    take one $T_{[i]}$ at a time
    for all Base Tables in$T_{[i]}$ do
        take one $T_k$ at a time
        for every buf.Table = $T_k$ do
            if (buf.key != Key($T_{[i]}$) ) and (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

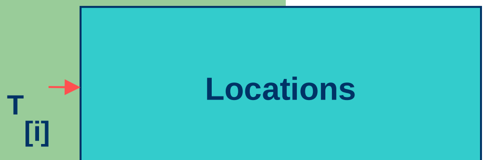
T [i]

Employees	Job_History	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	Locations	Country_Id
Countries	Employees Job_History Departments Jobs Locations	Countries	Country_Id		
Employees Job_History	Departments	Employees	Department_Id	Job_History	Job_Id
Employees Job_History Departments	Jobs	Job_History	Job_Id	Departments	Location_Id
Employees Job_History Departments Jobs	Locations	Departments	Location_Id		
Employees Job_History Departments Jobs Locations	Countries	Locations	Country_Id		
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($T_{[i]}$)
```

buf

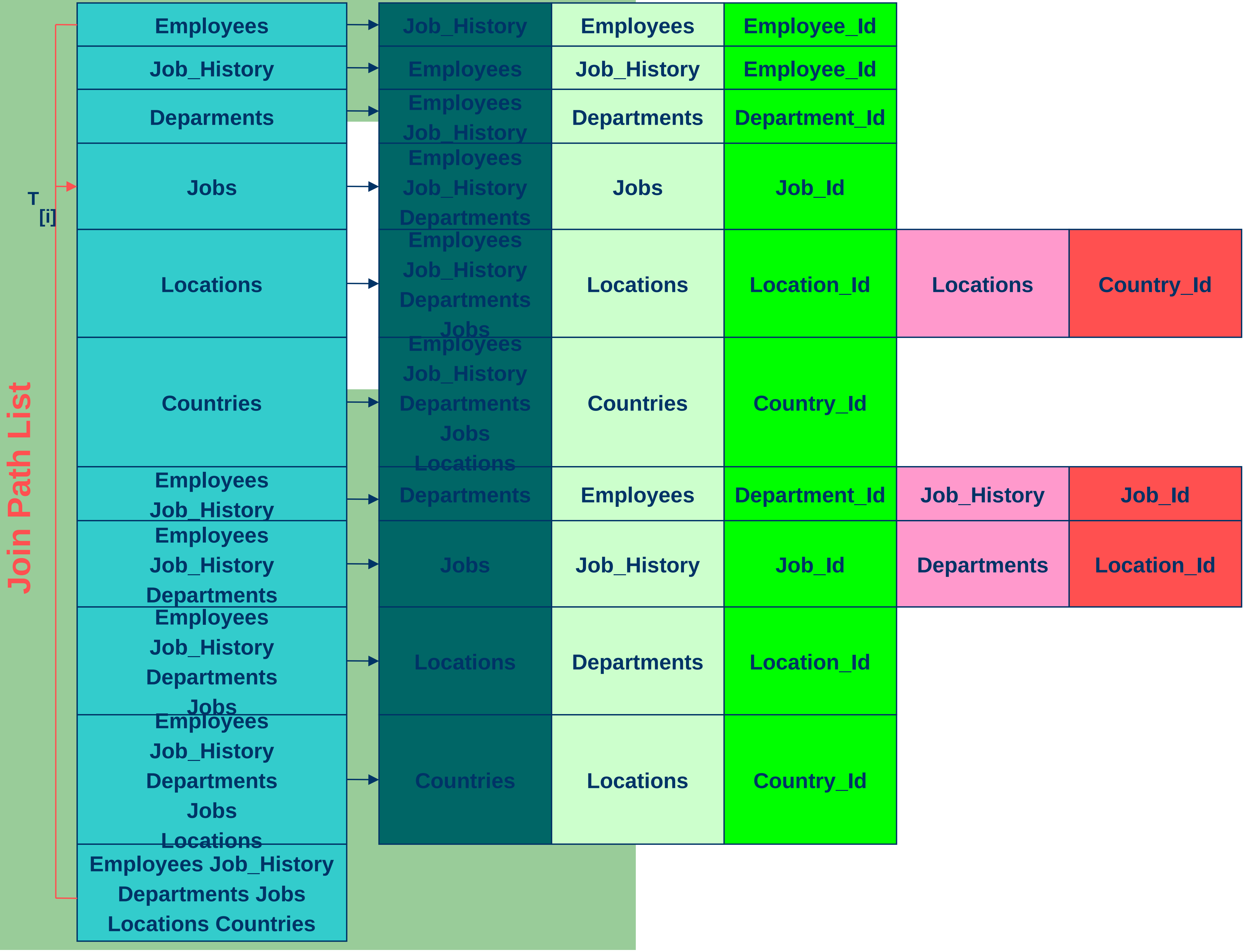
Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id



$T_{[i]}$

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($T_{[i]}$)
```



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	Locations	Country_Id
Countries	Employees Job_History Departments Jobs Locations	Countries	Country_Id		
Employees Job_History	Departments	Employees	Department_Id	Job_History	Job_Id
Employees Job_History Departments	Jobs	Job_History	Job_Id	Departments	Location_Id
Employees Job_History Departments Jobs	Locations	Departments	Location_Id		
Employees Job_History Departments Jobs Locations	Countries	Locations	Country_Id		
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($T_{[i]}$)
```

buf

Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

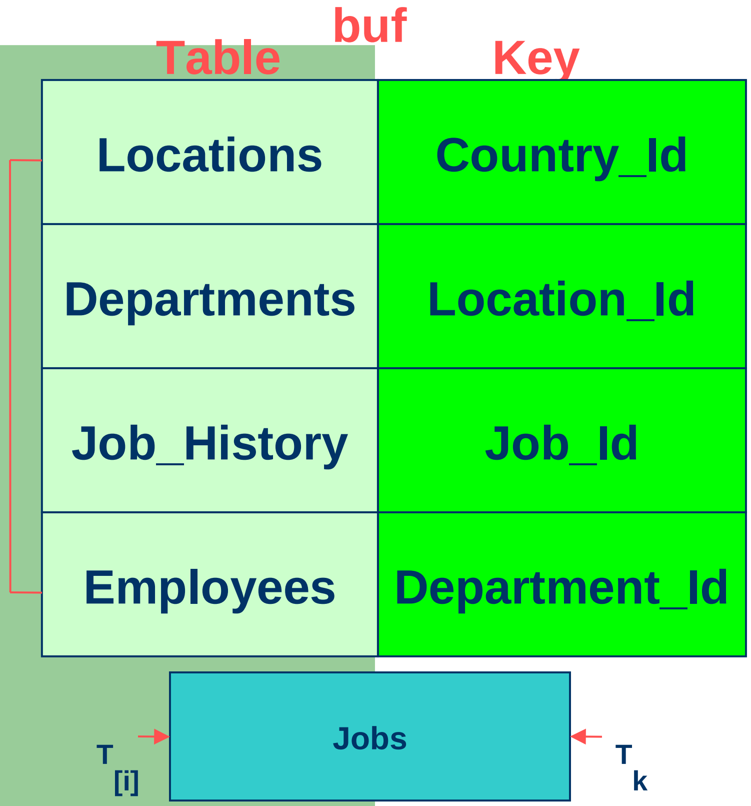
```
create a structure buf with 2 fields: Table and Key
for all the tables in JoinPathList going downward do
          take one $T_{[i]}$ at a time
→         for all Base Tables in$T_{[i]}$ do
                    take one $T_k$ at a time
                    for every buf.Table = $T_k$ do
                              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

$T_{[i]}$ → Jobs ← T_k

```
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

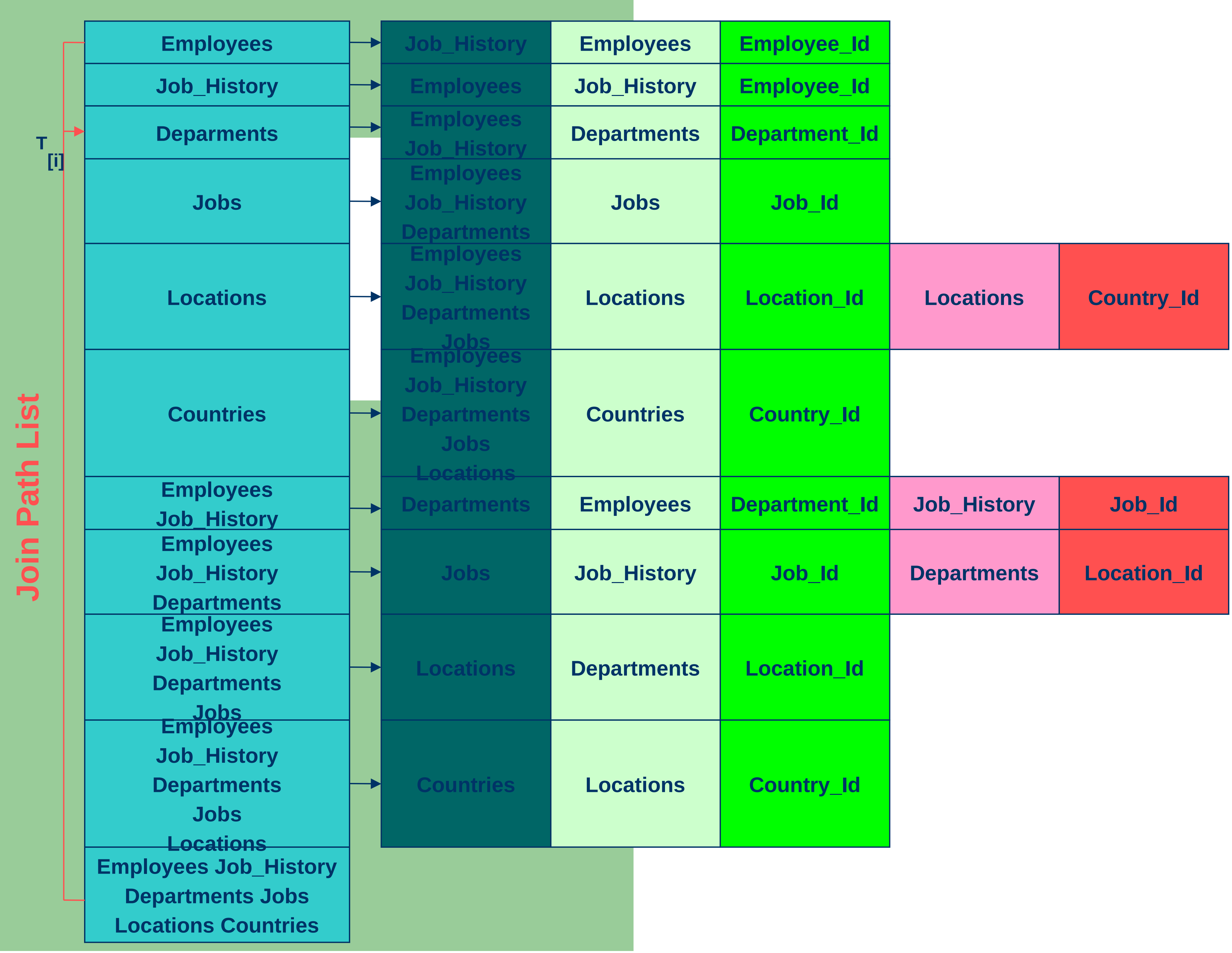
```
create a structure buf with 2 fields: Table and Key
for all the tables in JoinPathList going downward do
    take one $T_{[i]}$ at a time
    for all Base Tables in$T_{[i]}$ do
        take one $T_k$ at a time
        for every buf.Table = $T_k$ do
            if (buf.key != Key($T_{[i]}$) ) and (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

$T_{[i]}$ → 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($T_{[i]}$)
```



Employees	Job_History	Employees	Employee_Id		
Job_History	Employees	Job_History	Employee_Id		
Departments	Employees Job_History	Departments	Department_Id		
Jobs	Employees Job_History Departments	Jobs	Job_Id		
Locations	Employees Job_History Departments Jobs	Locations	Location_Id	Locations	Country_Id
Countries	Employees Job_History Departments Jobs Locations	Countries	Country_Id		
Employees Job_History	Departments	Employees	Department_Id	Job_History	Job_Id
Employees Job_History Departments	Jobs	Job_History	Job_Id	Departments	Location_Id
Employees Job_History Departments Jobs	Locations	Departments	Location_Id		
Employees Job_History Departments Jobs Locations	Countries	Locations	Country_Id		
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($T_{[i]}$)
```

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

$T_{[i]}$
Deparments
T_k

```
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
T_k

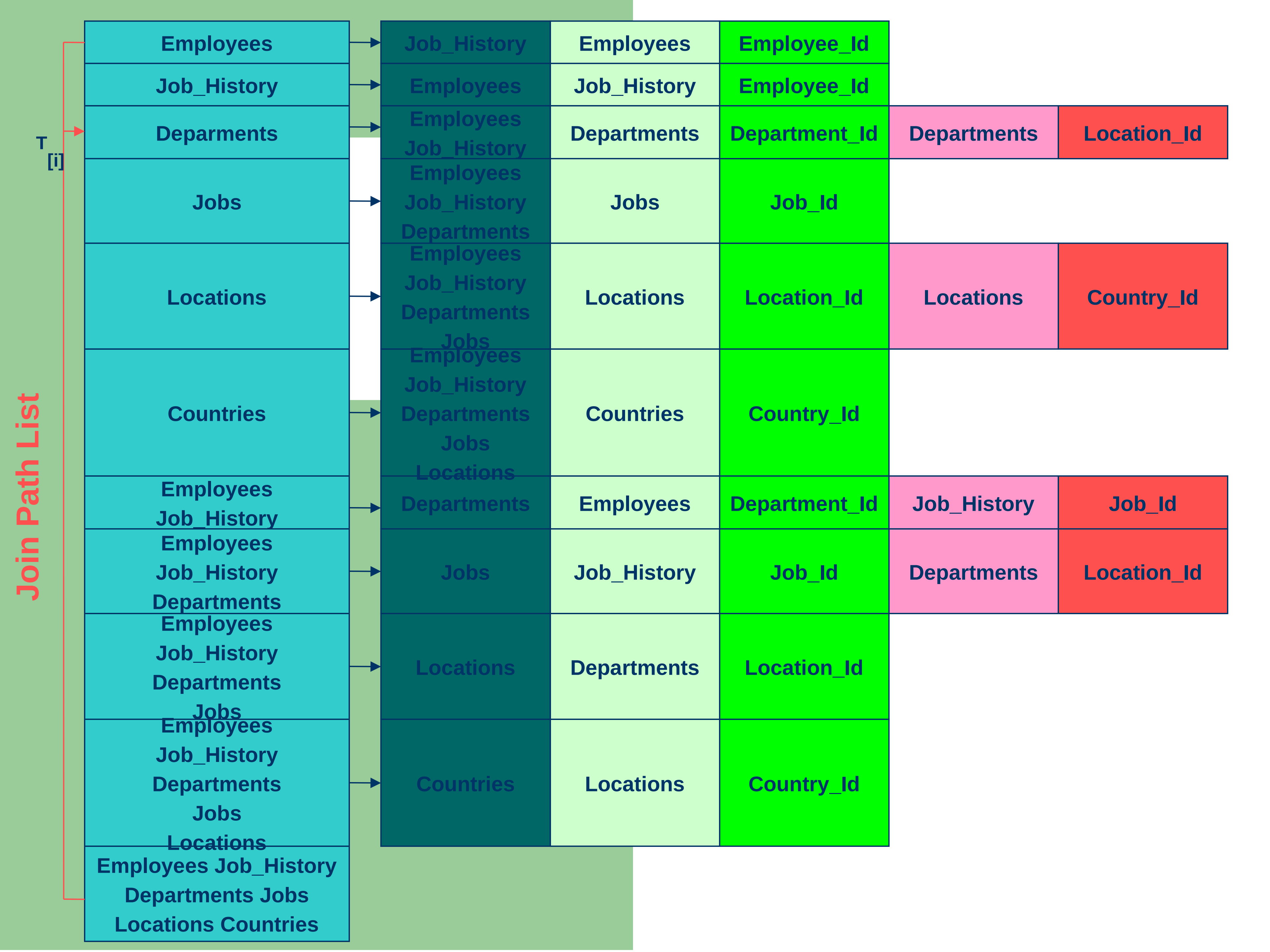
```
create a structure buf with 2 fields: Table and Key
for all the tables in JoinPathList going downward do
    take one $T_{[i]}$ at a time
    for all Base Tables in$T_{[i]}$ do
        take one $T_k$ at a time
        for every buf.Table = $T_k$ do
            if (buf.key != Key($T_{[i]}$) ) and (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
T_k

```
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($T_{[i]}$)
```



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

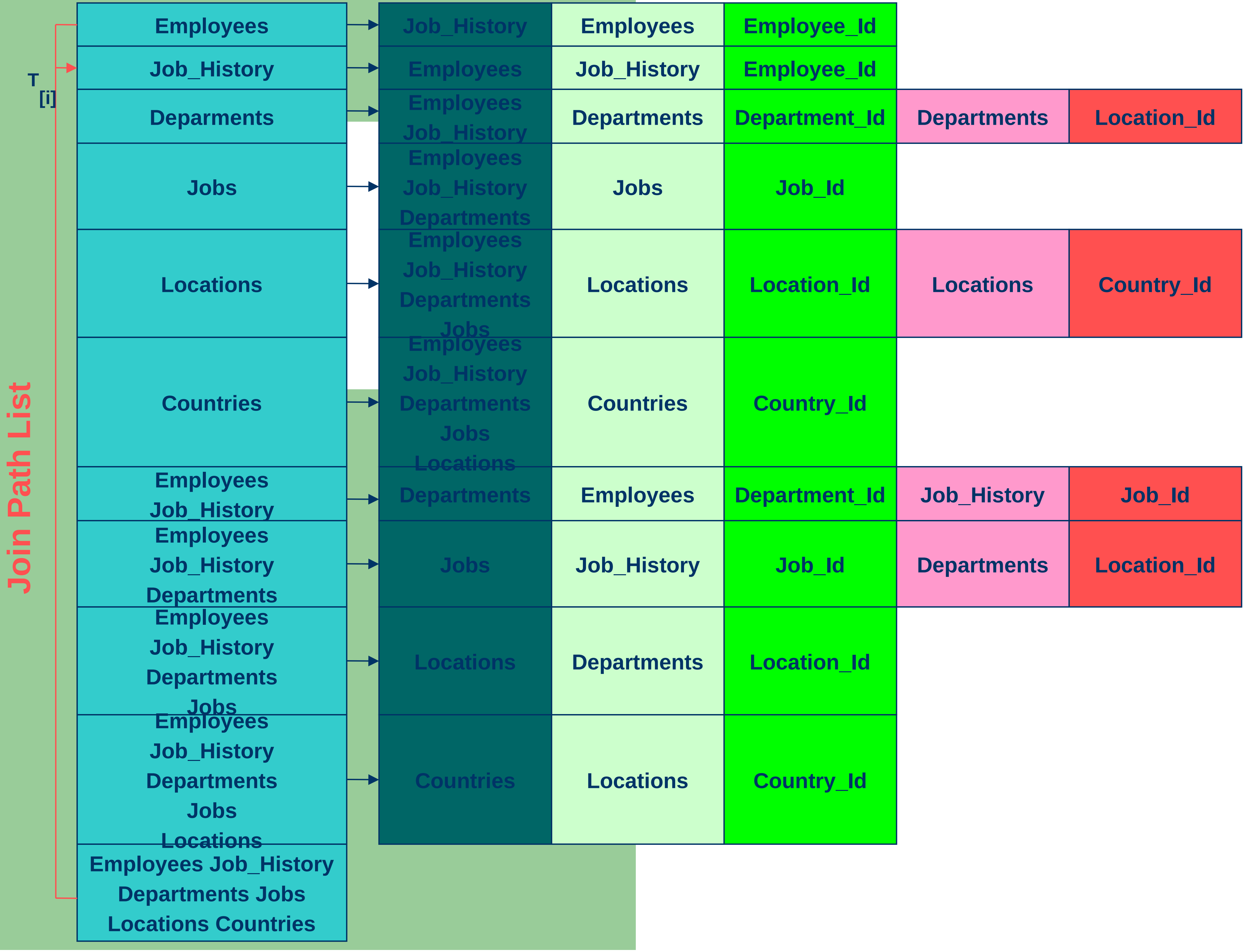
```
create a structure buf with 2 fields: Table and Key
for all the tables in JoinPathList going downward do
    take one $T_{[i]}$ at a time
    for all Base Tables in$T_{[i]}$ do
        take one $T_k$ at a time
        for every buf.Table = $T_k$ do
            if (buf.key != Key($T_{[i]}$) ) and (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($T_{[i]}$)
```

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($T_{[i]}$)
```



Employees	Job_History	Employees	Employee_Id		
Job_History	Employees	Job_History	Employee_Id		
Deparments	Employees Job_History	Departments	Department_Id	Departments	Location_Id
Jobs	Employees Job_History Departments	Jobs	Job_Id		
Locations	Employees Job_History Departments Jobs	Locations	Location_Id	Locations	Country_Id
Countries	Employees Job_History Departments Jobs Locations	Countries	Country_Id		
Employees Job_History	Departments	Employees	Department_Id	Job_History	Job_Id
Employees Job_History Departments	Jobs	Job_History	Job_Id	Departments	Location_Id
Employees Job_History Departments Jobs	Locations	Departments	Location_Id		
Employees Job_History Departments Jobs Locations	Countries	Locations	Country_Id		
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($T_{[i]}$)
```

buf

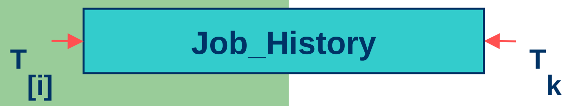
Table	Key
Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

$T_{[i]}$
Job_History

```
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

T
[i]
Job_History
T
k

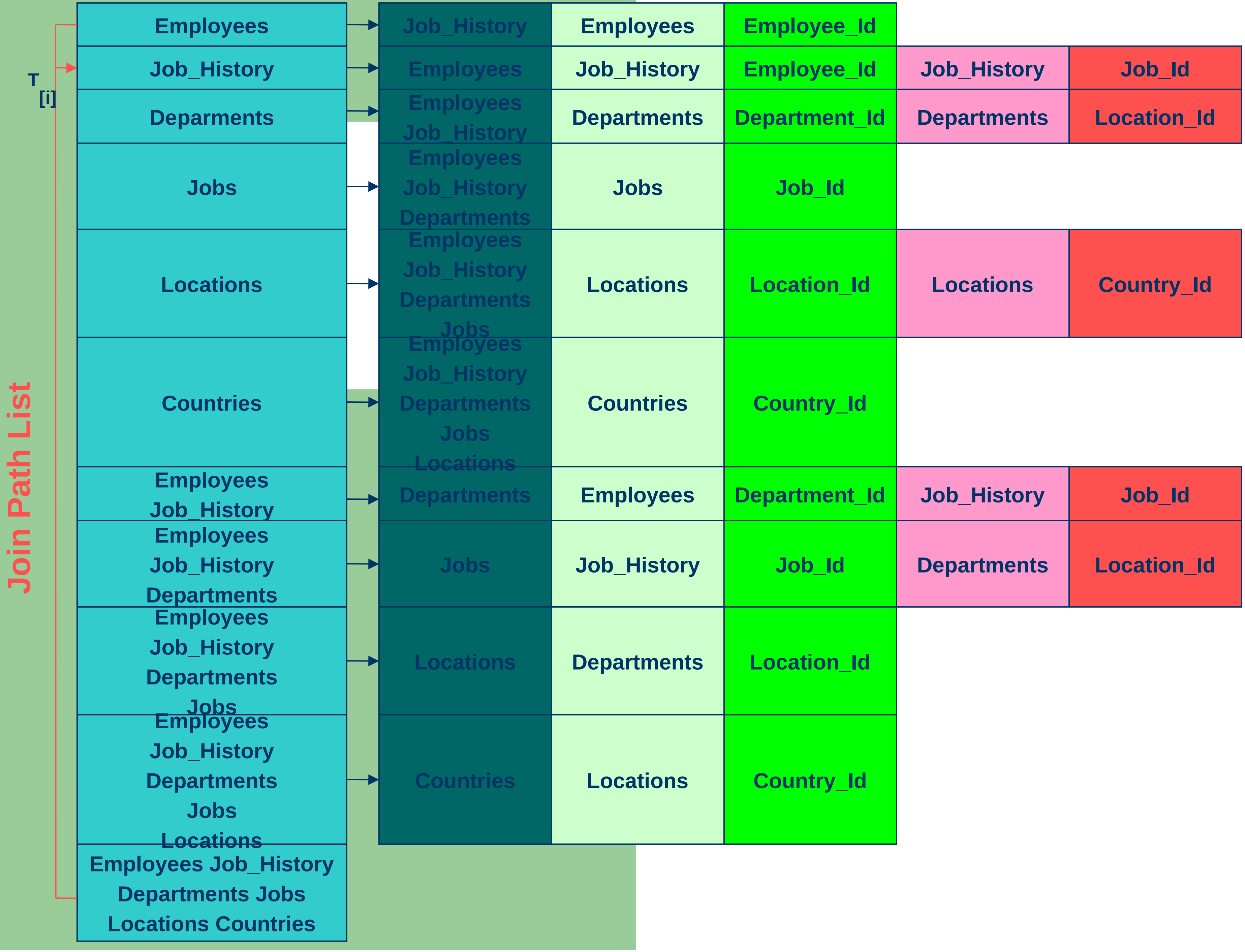
```
create a structure buf with 2 fields: Table and Key
for all the tables in JoinPathList going downward do
    take one $T_{[i]}$ at a time
    for all Base Tables in$T_{[i]}$ do
        take one $T_k$ at a time
        for every buf.Table = $T_k$ do
            if (buf.key != Key($T_{[i]}$) ) and (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

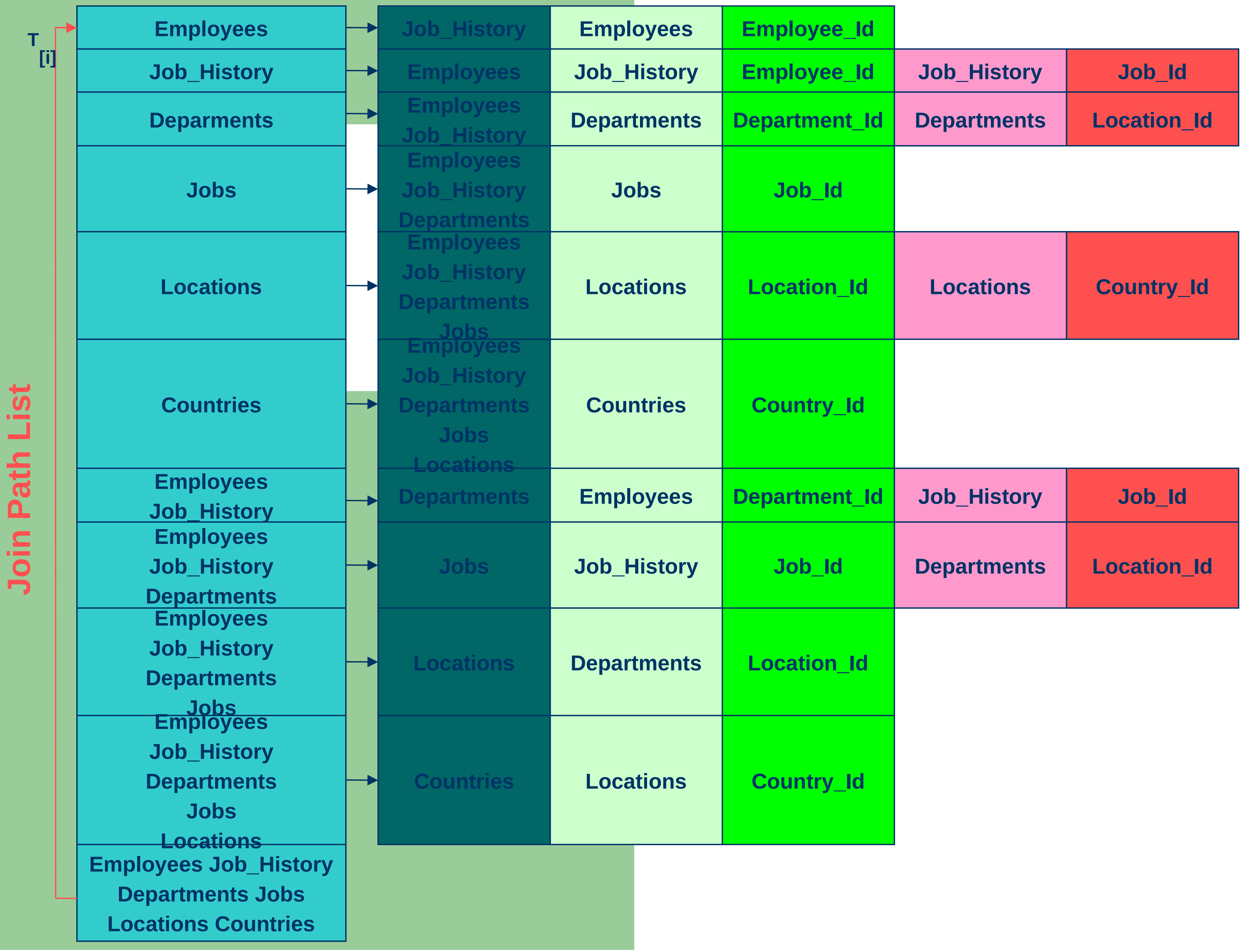
$T_{[i]}$
Job_History
T_k

```
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($T_{[i]}$)
```



Employees	Job_History	Employees	Employee_Id		
Job_History	Employees	Job_History	Employee_Id	Job_History	Job_Id
Deparments	Employees Job_History	Departments	Department_Id	Departments	Location_Id
Jobs	Employees Job_History Departments	Jobs	Job_Id		
Locations	Employees Job_History Departments Jobs	Locations	Location_Id	Locations	Country_Id
Countries	Employees Job_History Departments Jobs Locations	Countries	Country_Id		
Employees Job_History	Departments	Employees	Department_Id	Job_History	Job_Id
Employees Job_History Departments	Jobs	Job_History	Job_Id	Departments	Location_Id
Employees Job_History Departments Jobs	Locations	Departments	Location_Id		
Employees Job_History Departments Jobs Locations	Countries	Locations	Country_Id		
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($T_{[i]}$)
```



Employees	Job_History	Employees	Employee_Id		
Job_History	Employees	Job_History	Employee_Id	Job_History	Job_Id
Departments	Employees Job_History	Departments	Department_Id	Departments	Location_Id
Jobs	Employees Job_History Departments	Jobs	Job_Id		
Locations	Employees Job_History Departments Jobs	Locations	Location_Id	Locations	Country_Id
Countries	Employees Job_History Departments Jobs Locations	Countries	Country_Id		
Employees Job_History	Departments	Employees	Department_Id	Job_History	Job_Id
Employees Job_History Departments	Jobs	Job_History	Job_Id	Departments	Location_Id
Employees Job_History Departments Jobs	Locations	Departments	Location_Id		
Employees Job_History Departments Jobs Locations	Countries	Locations	Country_Id		
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($T_{[i]}$)
```

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

$T_{[i]}$

Employees

T_k

```
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

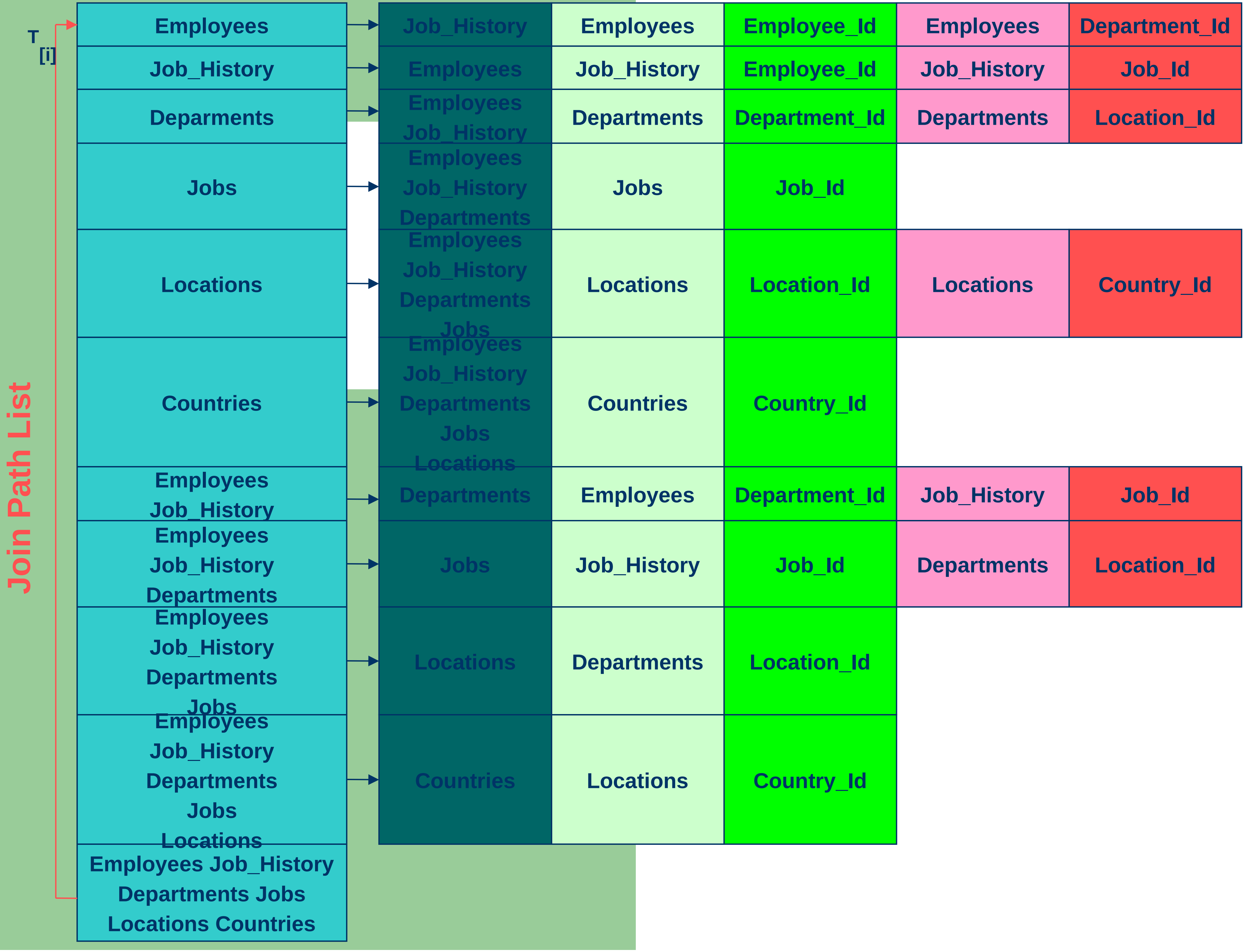
```
create a structure buf with 2 fields: Table and Key
for all the tables in JoinPathList going downward do
    take one $T_{[i]}$ at a time
    for all Base Tables in$T_{[i]}$ do
        take one $T_k$ at a time
        for every buf.Table = $T_k$ do
            if (buf.key != Key($T_{[i]}$) ) and (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

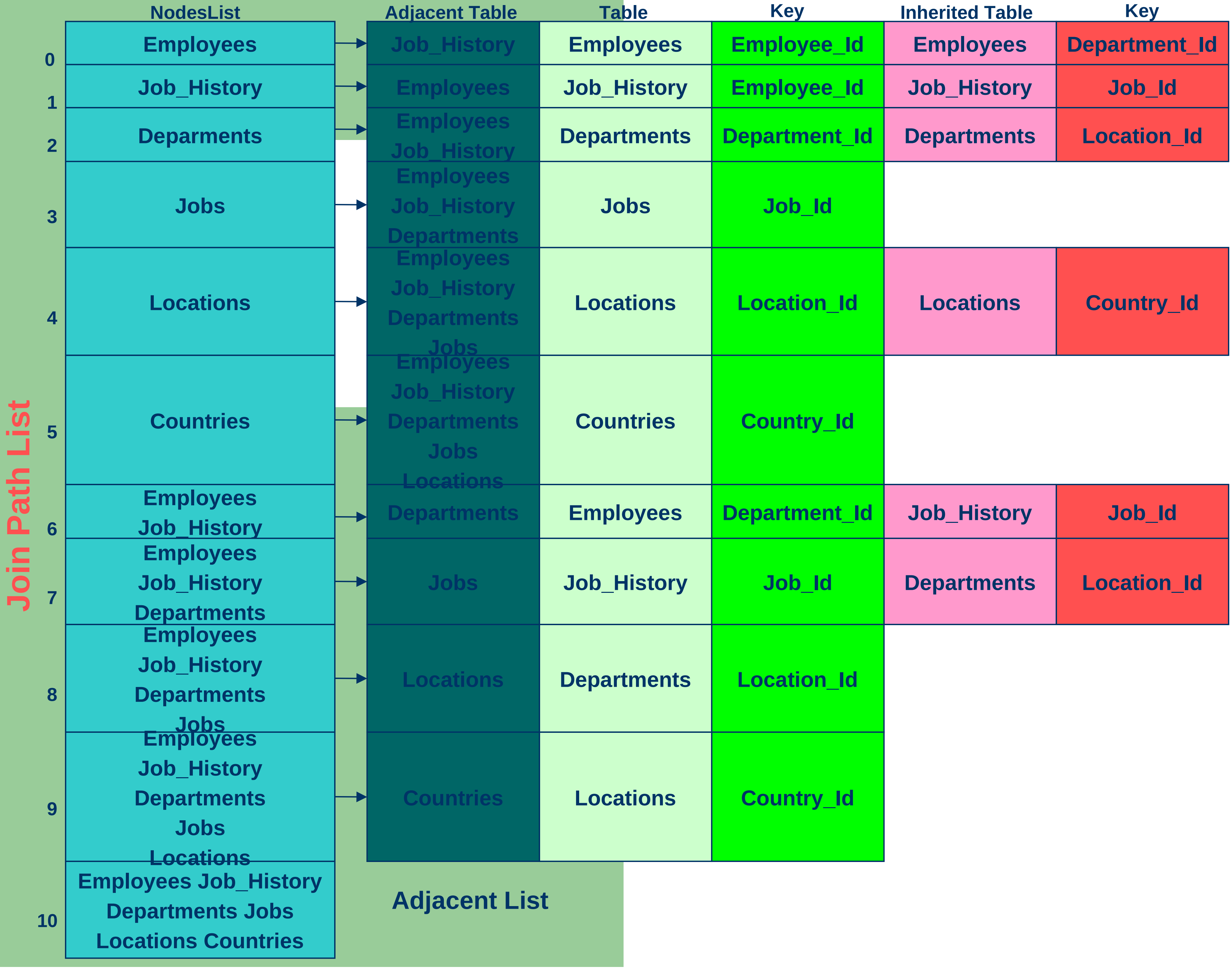
$T_{[i]}$
Employees
T_k

```
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($T_{[i]}$)
```



Employees	Job_History	Employees	Employee_Id	Employees	Department_Id
Job_History	Employees	Job_History	Employee_Id	Job_History	Job_Id
Departments	Employees Job_History	Departments	Department_Id	Departments	Location_Id
Jobs	Employees Job_History Departments	Jobs	Job_Id		
Locations	Employees Job_History Departments Jobs	Locations	Location_Id	Locations	Country_Id
Countries	Employees Job_History Departments Jobs Locations	Countries	Country_Id		
Employees Job_History	Departments	Employees	Department_Id	Job_History	Job_Id
Employees Job_History Departments	Jobs	Job_History	Job_Id	Departments	Location_Id
Employees Job_History Departments Jobs	Locations	Departments	Location_Id		
Employees Job_History Departments Jobs Locations	Countries	Locations	Country_Id		
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($T_{[i]}$)
```



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	Departments	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$^{\bowtie}$Tree.

Give for every entry in the JoinPathList a B^{+}Tree index with name as the B$^{\bowtie}$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.

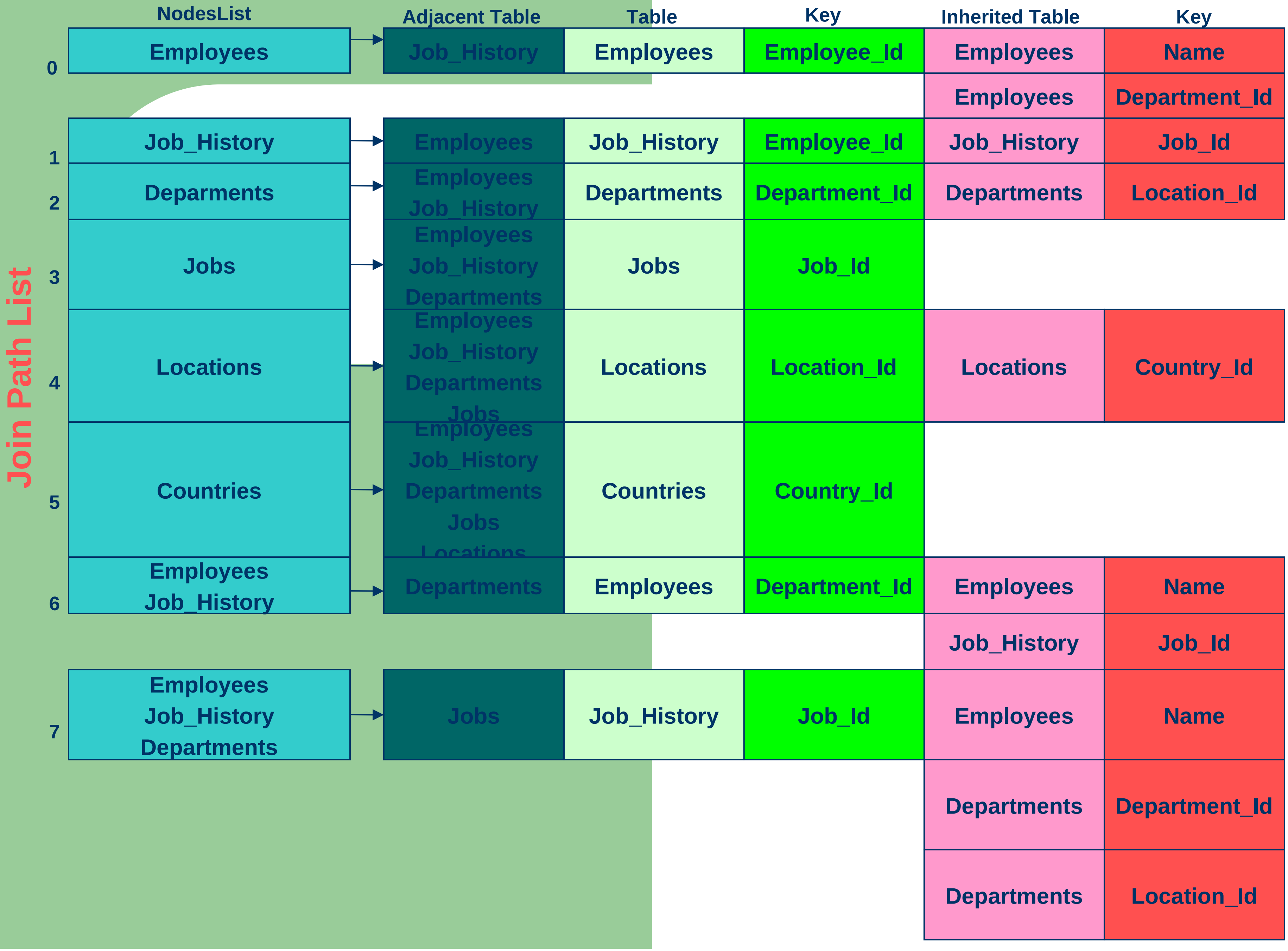
If the table is in join with itself, consider the table twice as aliases.

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

Implementation:

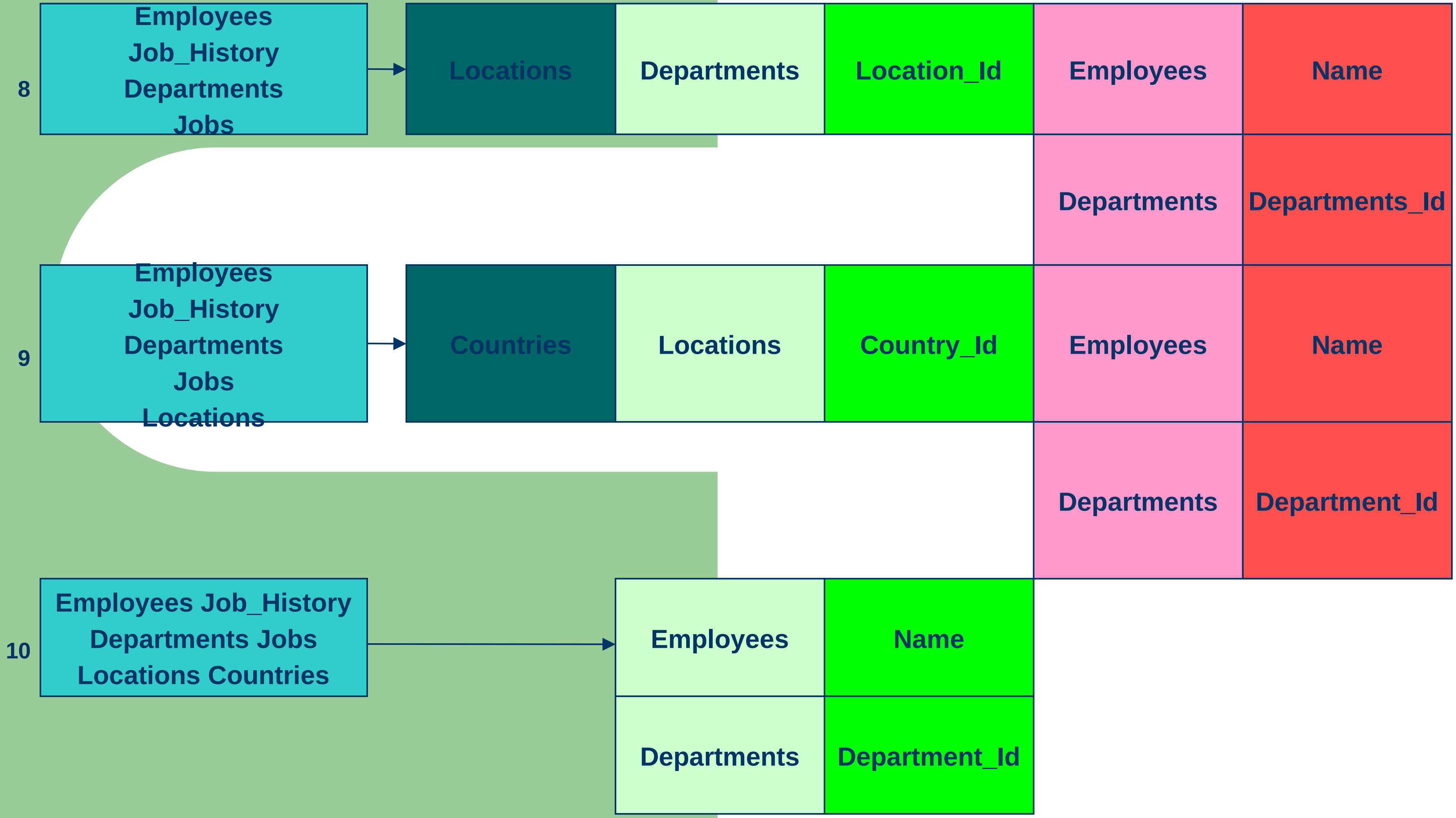
Use a big buffer and from the Data Dictionary divide it by the keys length, inherited keys length and space for the number of Data Pointers.

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



Join Path List

	NodesList	Adjacent Table	Table	Key	Inherited Table	Key
0	Employees	Job_History	Employees	Employee_Id	Employees Employees	Name 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	Employees Job_History	Name Job_Id
7	Employees Job_History Departments	Jobs	Job_History	Job_Id	Employees Departments Departments	Name Department_Id Location_Id



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

Create B$^+$Trees

- The Nodes (Vertexes) in the JoinPathList represents all the base tables + virtual tables constituting from the base tables by adding one at a time in mode that the one added is at least in direct join with its precedents.
- Defining a B$^+$Tree for every node, the ones for the virtual tables have for every key a set of data pointers equal to the number of base tables constituting it and from definition of the virtual tables, combining the rows pointed by those data pointers we obtain a joined row.

The algorithm for creating B^{+}Trees is the following:

```
create B+Trees(in PathJoinList; out B+Trees);
give a general name for the BJoinTree
for all entries in JoinPathList do
        take one node at a time
        create a B+Tree for the node defined as
                name of the B+Tree equal to the name of BJoinTree follow by the
                   index number of the node entry
                Number of data pointers equal to the number of base tables
                   constituting the virtual table of the node
                Key is defined by the pair <Table, Key> in the adjacent list of the
                   node
                Inherited Keys are defined by the pairs <Table, Inheritred Key> in
                   the adjacent list of the node
```

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.

Notice that $Keys_i$, $InheritedKeys_i$ and DP_i are relative to the row R_m from table T_i

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])

Employees table

	EMPLOYEE_ID	NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID
0	101	Mark Stench	mstench	233-4268	12/02/1998	FI_MGR	60000	FIN
1	102	Jorge Perez	jperez	448-5268	05/14/1999	AC_MGR	60000	ACC
2	103	Edward Cartier	ecartier	742-8429	03/01/2003	SA_MGR	60000	SAL
3	104	Teresa Gonzalez	tgonzalez	134-8329	12/20/2002	AC_AUD	55000	ACC
4	105	Michelle Blanche	mblanche	745-7496	01/02/2001	SA_REP	35000	SAL

Job_History table

	EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
0	101	12/16/1998	12/15/1999	AC_AUD	ACC
1	102	05/16/1999	05/15/2001	AC_AUD	ACC
2	101	12/16/1999	12/15/2001	SA_REP	SAL
3	103	03/16/2003	03/15/2004	AC_AUD	ACC

Departments table

	Deparment_Id	Department_Name	Manager_Id	Location_Id
0	FIN	FINANCE	101	1000
1	ACC	ACCOUNTING	102	1010
2	SAL	SALES	103	1020

Jobs Table

	JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
0	AC_AUD	Accounting Auditor	30000	60000
1	AC_MGR	Accounting Manager	60000	70000
2	FI_MGR	Finance Manager	50000	70000
3	SA_MGR	Sales Manager	50000	60000
4	SA_REP	Sales Representative	30000	40000

Locations table

	LOCATION_ ID	STREET_ADDRESS	POSTAL_ CODE	CITY	STATE PROVINCE	COUNTRY_ ID
0	1000	22220 Cochrane Drive	V6V 2T9	Richmond	B.C.	ca
1	1010	Calle Sermiento numero 300	62547	Guadalajara	Baja	me
2	1020	Rue des fleurs n. 345	78921	Toulouse	Moyenne	fr

Countries table

	Country_Id	Country_Name
0	ca	Canada
1	fr	France
2	me	Mexico

Inserting first row from table Employees

Base Table

Employees
0

EMPLOYEE_ID	NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID
101	Mark Stench	mstench	233-4268	12/02/1998	FI_MGR	60000	FIN

```
insertRoutine(in BaseTable $T_i$, Row $R_m$, DataRef $DP_i$)
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
        call AddJoinKey($T_i$,$Keys_i$,$InheritedKeys_i$,$DP_i$)
```

Base Table

Employees
0

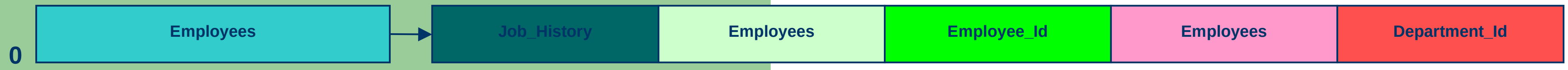
T_i

	EMPLOYEE_ID	NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID
DP_{i} 0	101	Mark Stench	mstench	233-4268	12/02/1998	FI_MGR	60000	FIN

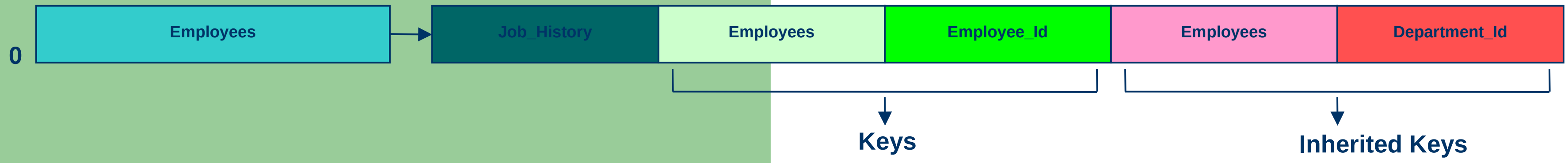
DataRef

Row R_m

```
insertRoutine(in BaseTable $T_i$ , Row $R_m$ , DataRef $DP_i$ )
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
        call AddJoinKey($T_i$ ,$Keys_i$ ,$InheritedKeys_i$ ,$DP_i$ )
```



```
insertRoutine(in BaseTable $T_i$ , Row $R_m$ , DataRef $DP_i$ )
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
        call AddJoinKey($T_i$ ,$Keys_i$ ,$InheritedKeys_i$ ,$DP_i$ )
```



```
insertRoutine(in BaseTable $T_i$ , Row $R_m$ , DataRef $DP_i$ )
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
        call AddJoinKey($T_i$ ,$Keys_i$ ,$InheritedKeys_i$ ,$DP_i$ )
```

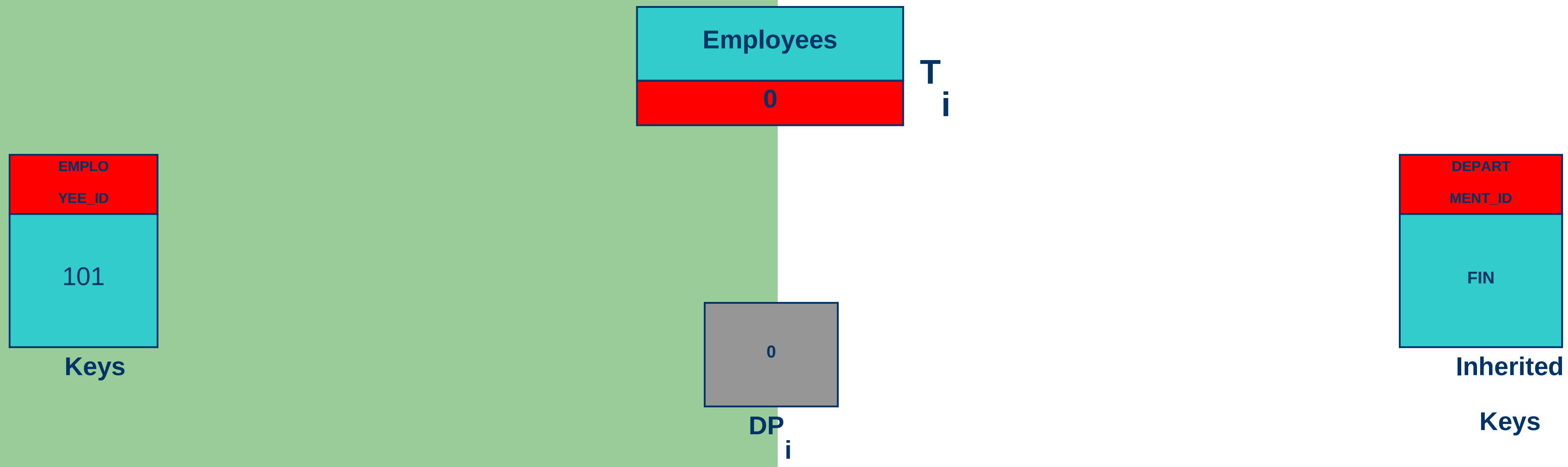
	EMPLOYEE_ID	NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID
DP_i 0	101	Mark Stench	mstench	233-4268	12/02/1998	FI_MGR	60000	FIN

Keys

Row R_m

Inherited Keys

```
insertRoutine(in BaseTable $T_i$, Row $R_m$, DataRef $DP_i$)
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
→       call AddJoinKey($T_i$,$Keys_i$,$InheritedKeys_i$,$DP_i$)
```



Employees

0

T_i

EMPLOYEE_ID

101

Keys

0

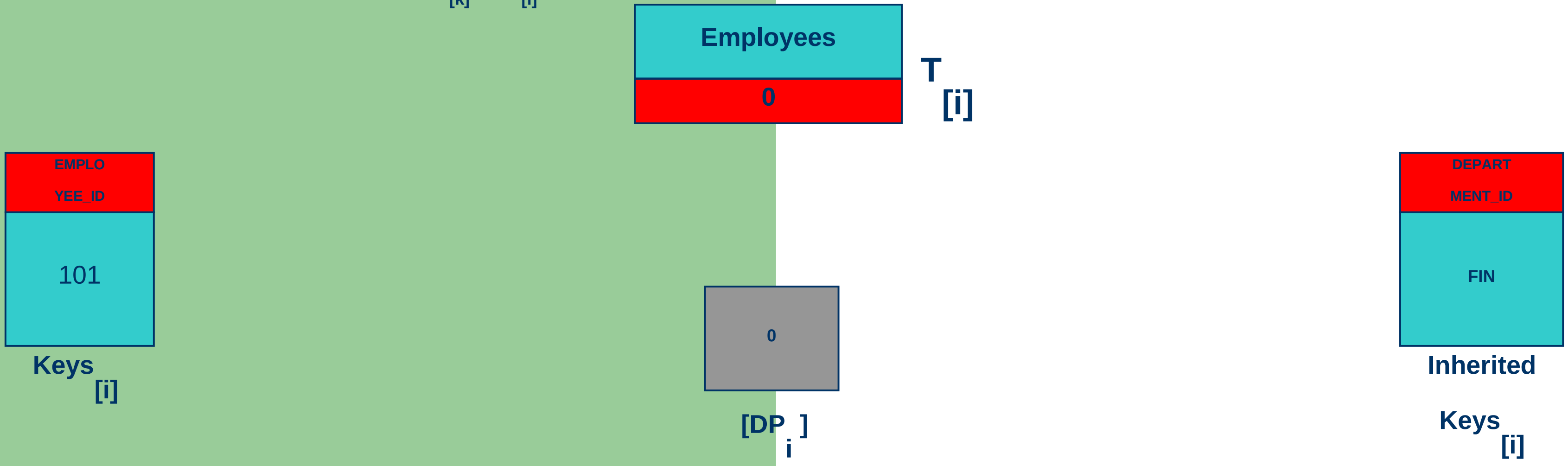
DP_i

DEPARTMENT_ID

FIN

Inherited Keys

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```



Employees

0

$T_{[i]}$

EMPLOYEE_ID

101

$Keys_{[i]}$

0

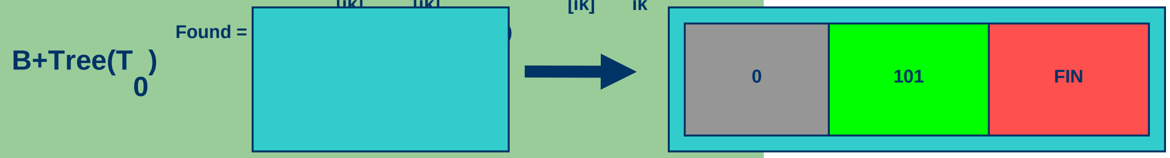
$[DP_i]$

DEPARTMENT_ID

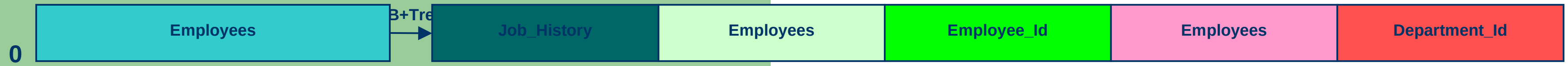
FIN

Inherited Keys$_{[i]}$

```
addJoinKey(in Virtual Table $T_{[i]}$, Keys$_{[i]}$, InheritedKeys$_{[i]}$, [DP$_i$])
    call addKey(B+Tree($T_{[i]}$),Keys$_{[i]}$,InheritedKeys$_{[i]}$,[DP$_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,Keys$_{[k]}$,InheritedKeys$_{[k]}$,[DP$_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,Keys$_{[ik]}$,InheritedKeys$_{[ik]}$,[DP$_{ik}$])
            Found =                    )
```

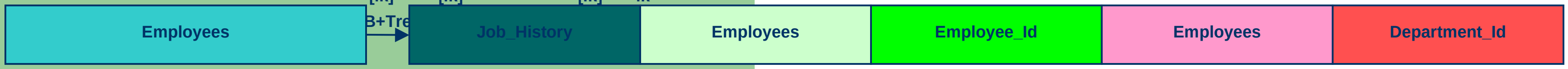


```
addJoinKey(in Virtual Table $T_{[i]}$, Keys$_{[i]}$, InheritedKeys$_{[i]}$, [DP$_i$])
    call addKey(B+Tree($T_{[i]}$),Keys$_{[i]}$,InheritedKeys$_{[i]}$,[DP$_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,Keys$_{[k]}$,InheritedKeys$_{[k]}$,[DP$_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,Keys$_{[ik]}$,InheritedKeys$_{[ik]}$,[DP$_{ik}$])
            B+Tre
```



```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
→   from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
```

0

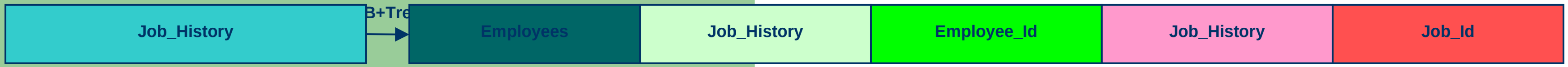


Adjacent

Table

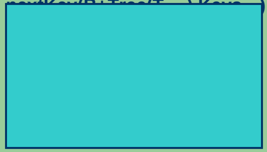
```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
→       locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
```

1



```
addJoinKey(in Virtual Table $T_{[i]}$, Keys$_{[i]}$, InheritedKeys$_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),Keys$_{[i]}$,InheritedKeys$_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
  →     Found = findKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,Keys$_{[k]}$,InheritedKeys$_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,Keys$_{[ik]}$,InheritedKeys$_{[ik]}$,[$DP_{ik}$])
            Found = nextKey(B+Tree(T ),Keys )
```

B+Tree(T_1)



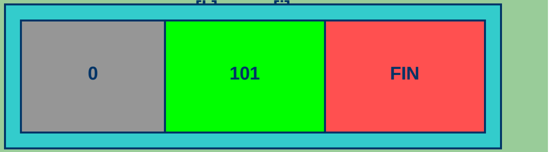
Found: FALSE

```
addJoinKey(in Virtual Table $T_{[i]}$, Keys$_{[i]}$, InheritedKeys$_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),Keys$_{[i]}$,InheritedKeys$_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
  →     Found = findKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,Keys$_{[k]}$,InheritedKeys$_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,Keys$_{[ik]}$,InheritedKeys$_{[ik]}$,[$DP_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
```

Found: FALSE

```
addJoinKey(in Virtual Table $T_{[i]}$, Keys$_{[i]}$, InheritedKeys$_{[i]}$, [DP$_i$])
    call addKey(B+Tree($T_{[i]}$),Keys$_{[i]}$,InheritedKeys$_{[i]}$,[DP$_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,Keys$_{[k]}$,InheritedKeys$_{[k]}$,[DP$_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,Keys$_{[ik]}$,InheritedKeys$_{[ik]}$,[DP$_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
```

B+Tree(T_0)



0	101	FIN

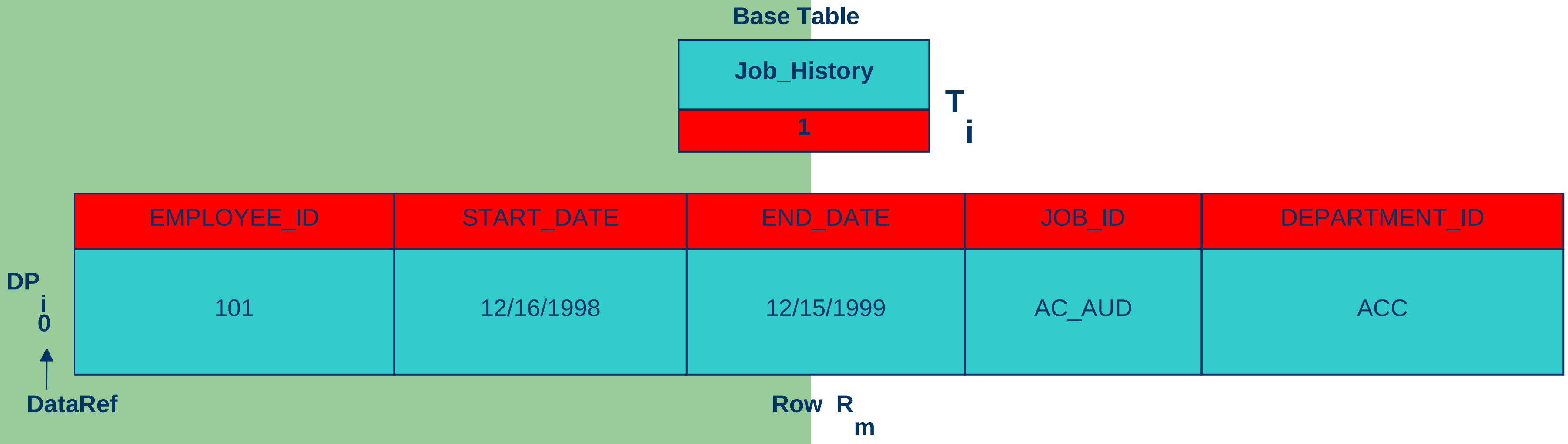
Inserting first row from table Job_History

Base Table

Job_History
1

EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
101	12/16/1998	12/15/1999	AC_AUD	ACC

```
→ insertRoutine(in BaseTable $T_i$, Row $R_m$, DataRef $DP_i$)
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
        call AddJoinKey($T_i$,$Keys_i$,$InheritedKeys_i$,$DP_i$)
```



Base Table

Job_History
1

T_i

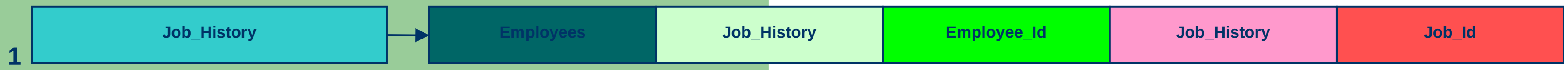
DP_{i0}

DataRef

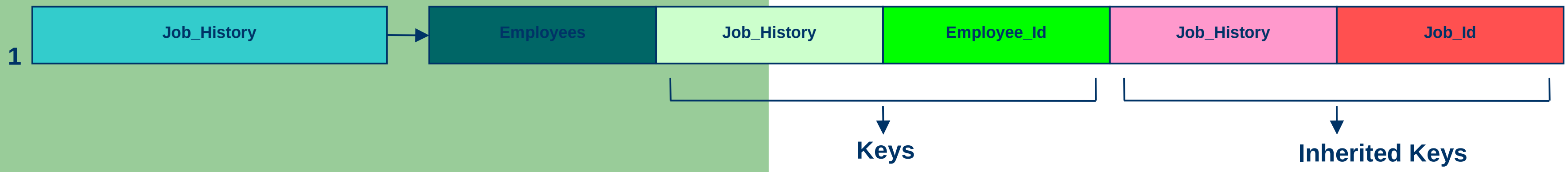
EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
101	12/16/1998	12/15/1999	AC_AUD	ACC

Row R_m

```
insertRoutine(in BaseTable $T_i$, Row $R_m$, DataRef $DP_i$)
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
        call AddJoinKey($T_i$,$Keys_i$,$InheritedKeys_i$,$DP_i$)
```



```
insertRoutine(in BaseTable $T_i$, Row $R_m$, DataRef $DP_i$)
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
        call AddJoinKey($T_i$,$Keys_i$,$InheritedKeys_i$,$DP_i$)
```



```
insertRoutine(in BaseTable $T_i$, Row $R_m$, DataRef $DP_i$)
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
        call AddJoinKey($T_i$,$Keys_i$,$InheritedKeys_i$,$DP_i$)
```

DP_{i0}

EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
101	12/16/1998	12/15/1999	AC_AUD	ACC

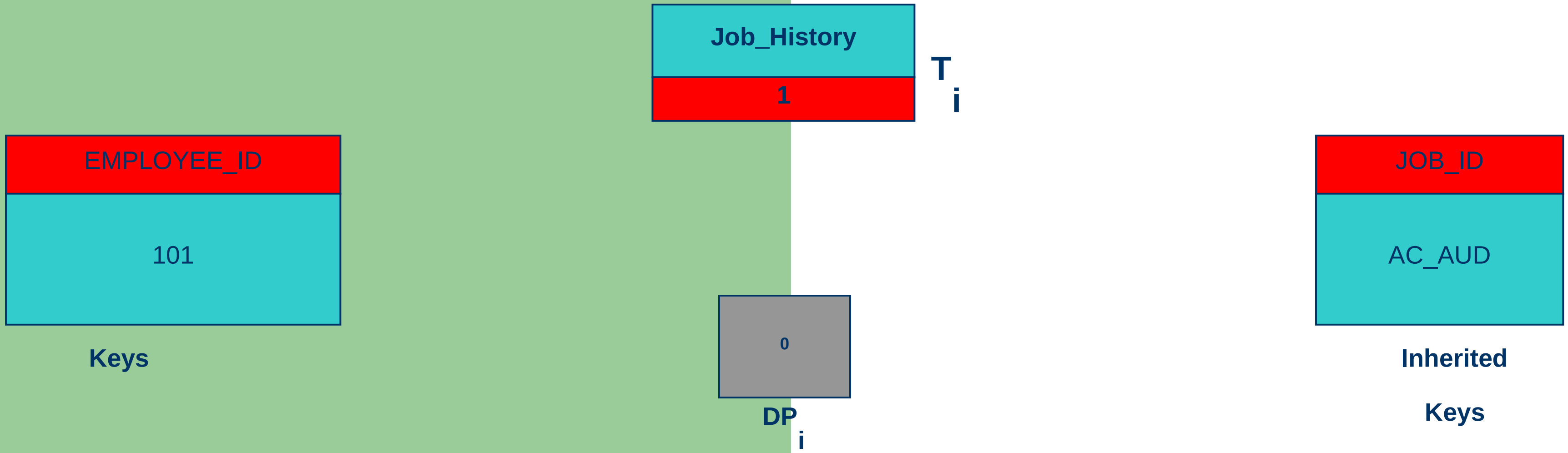
Keys

Row R_m

Inherited

Keys

```
insertRoutine(in BaseTable $T_i$, Row $R_m$, DataRef $DP_i$)
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
        call AddJoinKey($T_i$, $Keys_i$, $InheritedKeys_i$, $DP_i$)
```



Job_History

1

T_i

EMPLOYEE_ID

101

Keys

0

DP_i

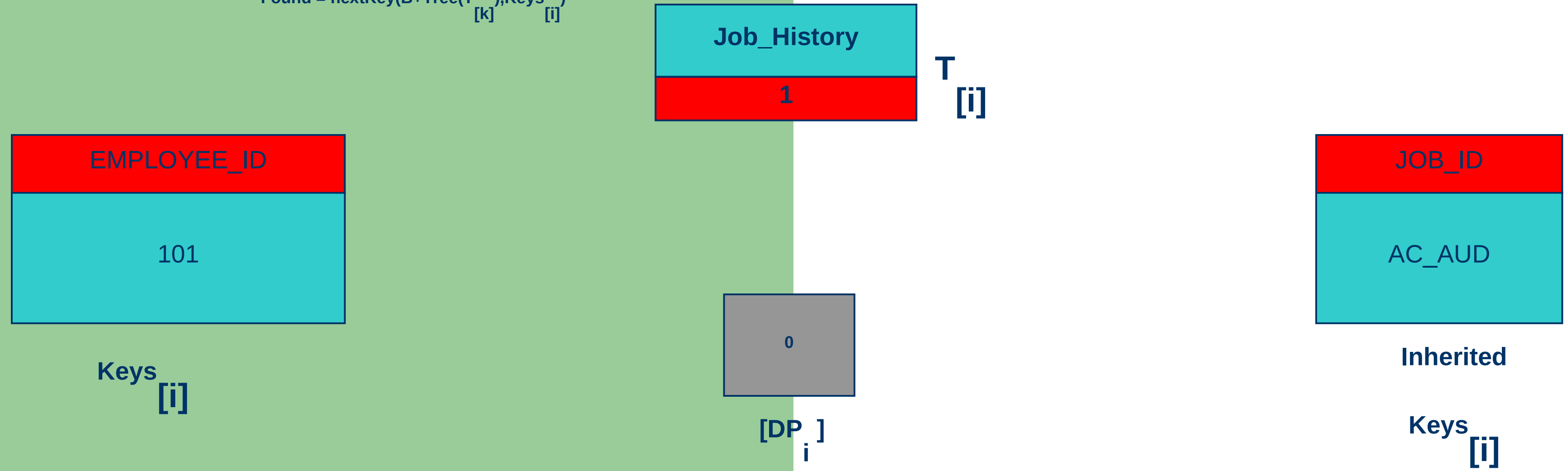
JOB_ID

AC_AUD

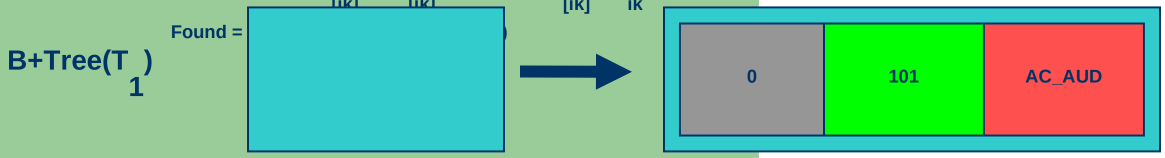
Inherited

Keys

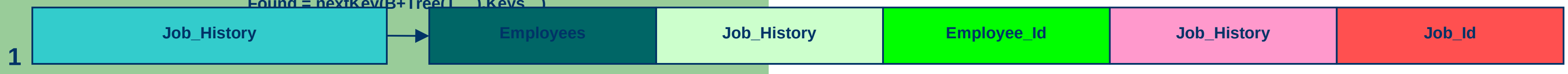
```
→ addJoinKey(in Virtual Table $T_{[i]}$, Keys$_{[i]}$, InheritedKeys$_{[i]}$, [DP$_i$])
      call addKey(B+Tree($T_{[i]}$),Keys$_{[i]}$,InheritedKeys$_{[i]}$,[DP$_i$])
      locate the entry of $T_{[i]}$ in JoinPathList
      from its adjacentList, locate the table $T_{[k]}$ adjacent to it
            locate the entry of $T_{[k]}$ in JoinPathList
            Found = findKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
            while found do
                  returnKeys(B+Tree($T_{[k]}$,Keys$_{[k]}$,InheritedKeys$_{[k]}$,[DP$_k$])
                  locate the entry of $T_{[ik]}$ in JoinPathList
                  from its adjacentList, locate the definition of Keys & Inherited Keys
                  from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
                  call AddJoinKey($T_{[ik]}$,Keys$_{[ik]}$,InheritedKeys$_{[ik]}$,[DP$_{ik}$])
                  Found = nextKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
```



```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found =                )
```



```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree(T   ),Keys   )
```



```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
→   from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
```

1	Job_History	B+Tre →	Employees	Job_History	Employee_Id	Job_History	Job_Id

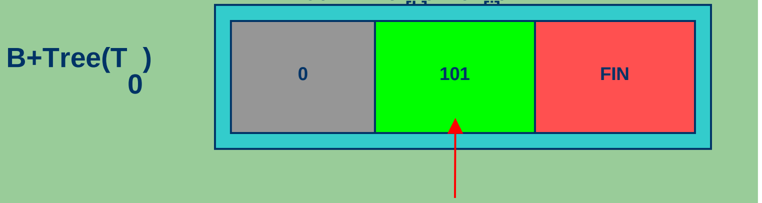
Adjacent

Table

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
→       locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
```

0	Employees	B+Tre →	Job_History	Employees	Employee_Id	Employees	Department_Id

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
→       Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```



Found: TRUE

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
→       while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

Found: TRUE

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

EMPLOYEE_ID
101

$Keys_{[k]}$

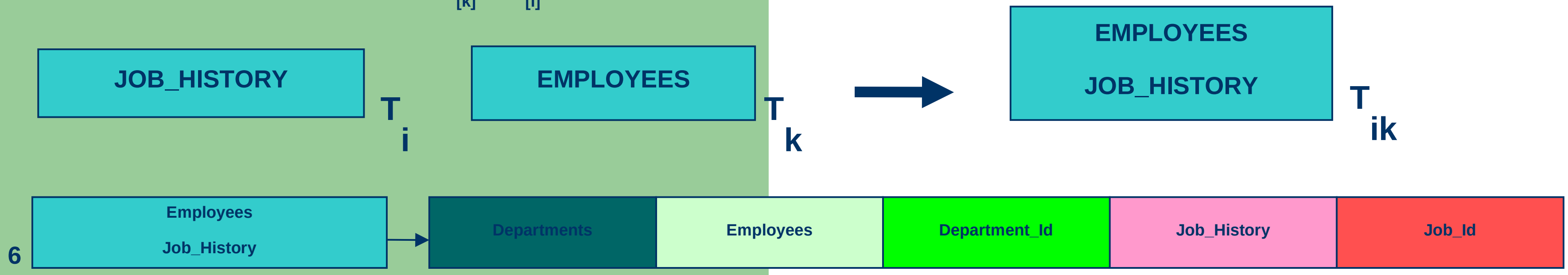
0

$[DP_k]$

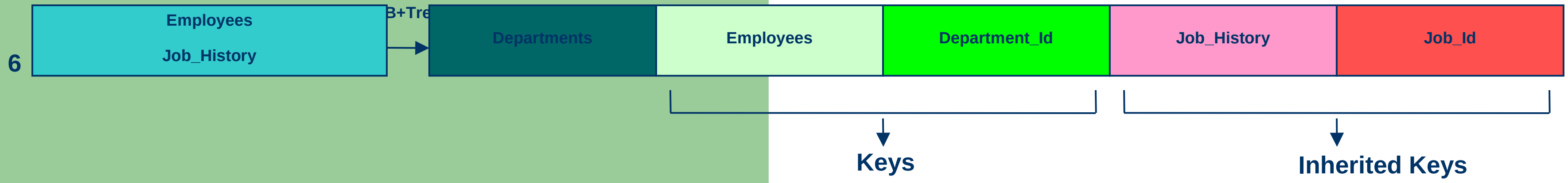
DEPARTMENT_ID
FIN

Inherited $Keys_{[i]}$

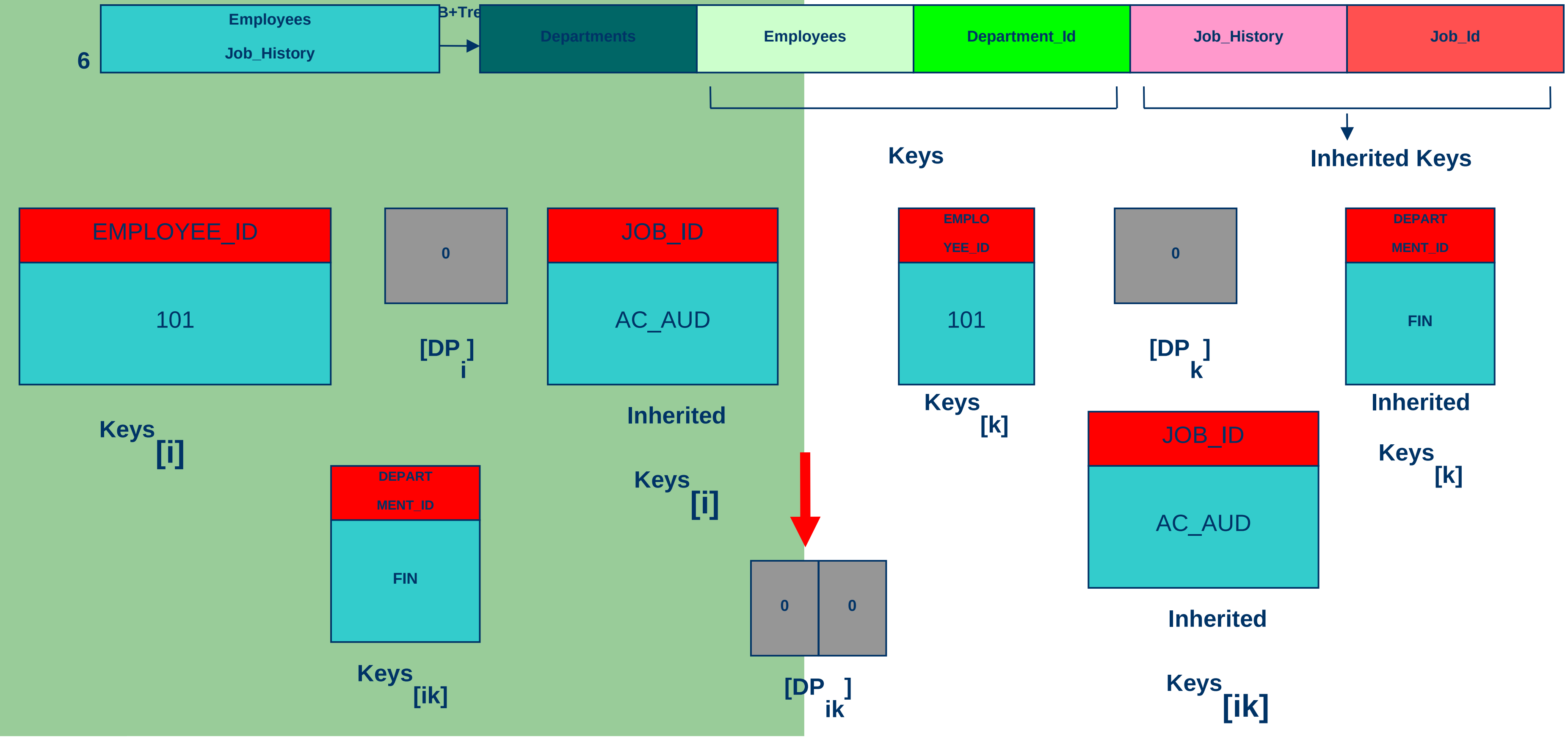
```
addJoinKey(in Virtual Table $T_{[i]}$, Keys$_{[i]}$, InheritedKeys$_{[i]}$, [DP$_i$])
    call addKey(B+Tree($T_{[i]}$),Keys$_{[i]}$,InheritedKeys$_{[i]}$,[DP$_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,Keys$_{[k]}$,InheritedKeys$_{[k]}$,[DP$_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,Keys$_{[ik]}$,InheritedKeys$_{[ik]}$,[DP$_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
```



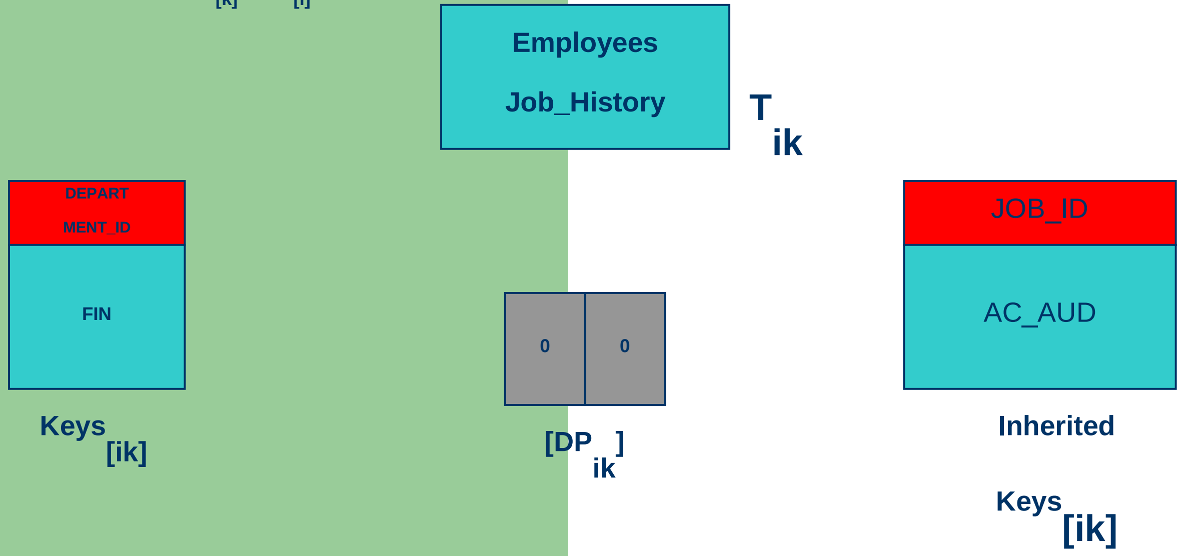
```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
```



```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
```



```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```



Employees

Job_History

T_{ik}

DEPARTMENT_ID

FIN

$Keys_{[ik]}$

0 0

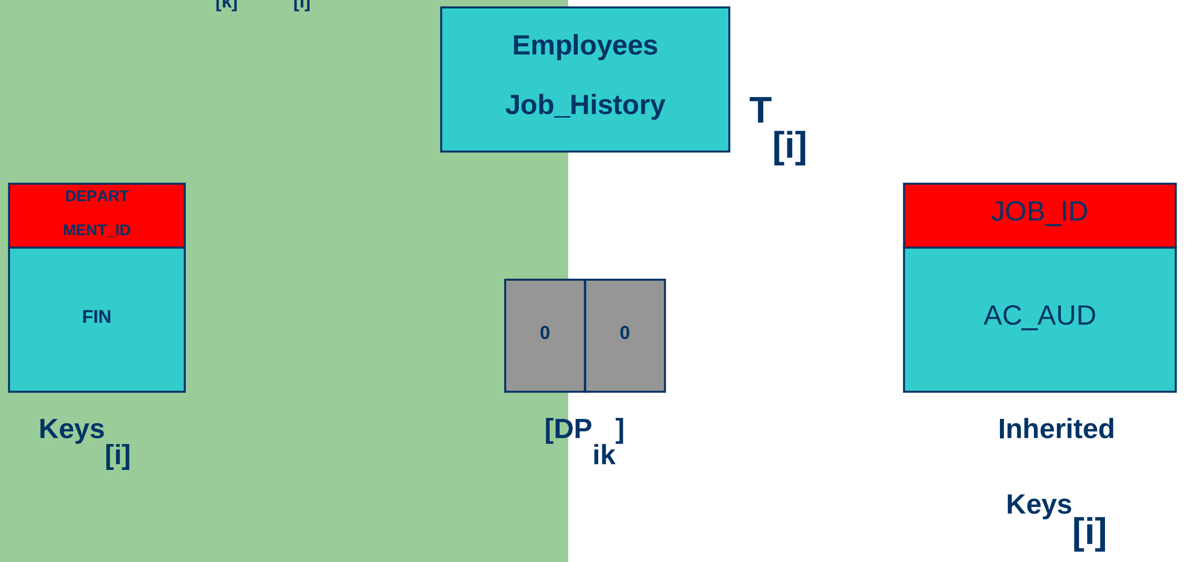
$[DP_{ik}]$

JOB_ID

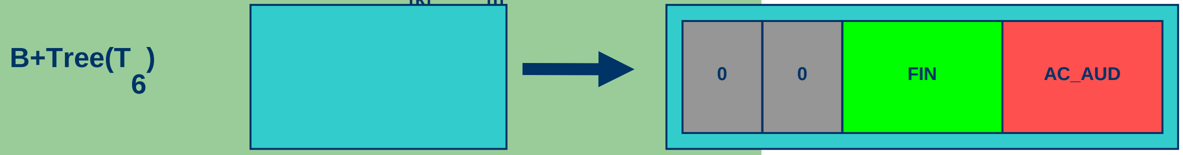
AC_AUD

Inherited Keys$_{[ik]}$

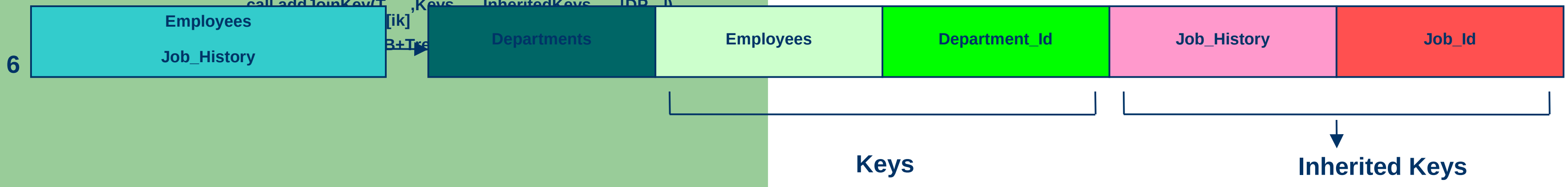
```
addJoinKey(in Virtual Table $T_{[i]}$, Keys$_{[i]}$, InheritedKeys$_{[i]}$, [DP$_i$])
    call addKey(B+Tree($T_{[i]}$),Keys$_{[i]}$,InheritedKeys$_{[i]}$,[DP$_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,Keys$_{[k]}$,InheritedKeys$_{[k]}$,[DP$_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,Keys$_{[ik]}$,InheritedKeys$_{[ik]}$,[DP$_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
```



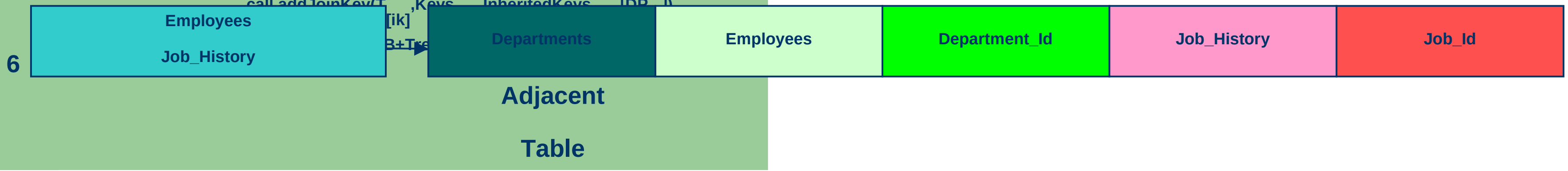
```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_{i}]$)
→   call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_{i}]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_{k}]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```



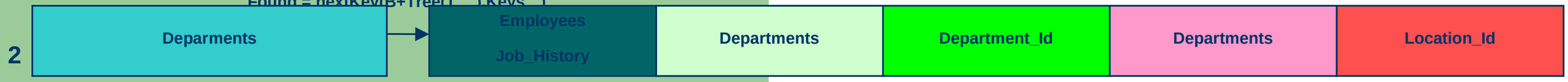
```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,Keys    InheritedKeys    [DP  ])
B+Tre
```



```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,Keys    InheritedKeys    [DP  ])
B+Tre
```



```
addJoinKey(in Virtual Table $T_{[i]}$, Keys$_{[i]}$, InheritedKeys$_{[i]}$, [DP$_i$])
          call addKey(B+Tree($T_{[i]}$),Keys$_{[i]}$,InheritedKeys$_{[i]}$,[DP$_i$])
          locate the entry of $T_{[i]}$ in JoinPathList
          from its adjacentList, locate the table $T_{[k]}$ adjacent to it
→                   locate the entry of $T_{[k]}$ in JoinPathList
                    Found = findKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
                    while found do
                              returnKeys(B+Tree($T_{[k]}$,Keys$_{[k]}$,InheritedKeys$_{[k]}$,[DP$_k$])
                              locate the entry of $T_{[ik]}$ in JoinPathList
                              from its adjacentList, locate the definition of Keys & Inherited Keys
                              from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
                              call addJoinKey($T_{[ik]}$,Keys$_{[ik]}$,InheritedKeys$_{[ik]}$,[DP$_{ik}$])
                              Found = nextKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
```



```
addJoinKey(in Virtual Table $T_{[i]}$, Keys$_{[i]}$, InheritedKeys$_{[i]}$, [DP$_i$])
          call addKey(B+Tree($T_{[i]}$),Keys$_{[i]}$,InheritedKeys$_{[i]}$,[DP$_i$])
          locate the entry of $T_{[i]}$ in JoinPathList
          from its adjacentList, locate the table $T_{[k]}$ adjacent to it
                    locate the entry of $T_{[k]}$ in JoinPathList
→                   Found = findKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
                    while found do
                              returnKeys(B+Tree($T_{[k]}$,Keys$_{[k]}$,InheritedKeys$_{[k]}$,[DP$_k$])
                              locate the entry of $T_{[ik]}$ in JoinPathList
                              from its adjacentList, locate the definition of Keys & Inherited Keys
                              from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
                              call addJoinKey($T_{[ik]}$,Keys$_{[ik]}$,InheritedKeys$_{[ik]}$,[DP$_{ik}$])
                              Found = nextKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
```

B+Tree(T_2)

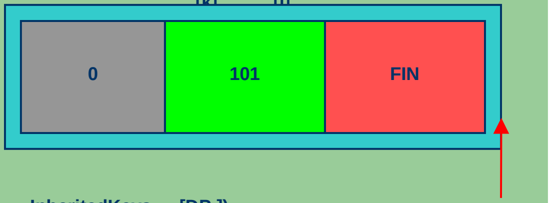
Found: FALSE

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
→       Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)        Found: FALSE
```

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
→           call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
→           call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

B+Tree(T_0)



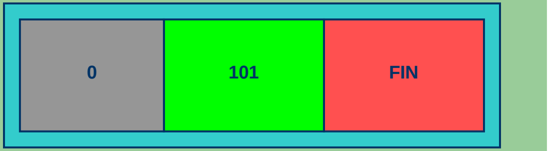
Found: FALSE

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
→       Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

Found: FALSE

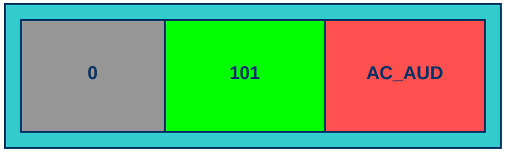
```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
→           call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

B+Tree(T_0)



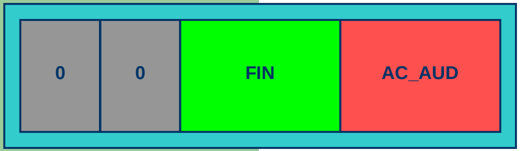
0 | 101 | FIN

B+Tree(T_1)



0 | 101 | AC_AUD

B+Tree(T_6)



0 | 0 | FIN | AC_AUD

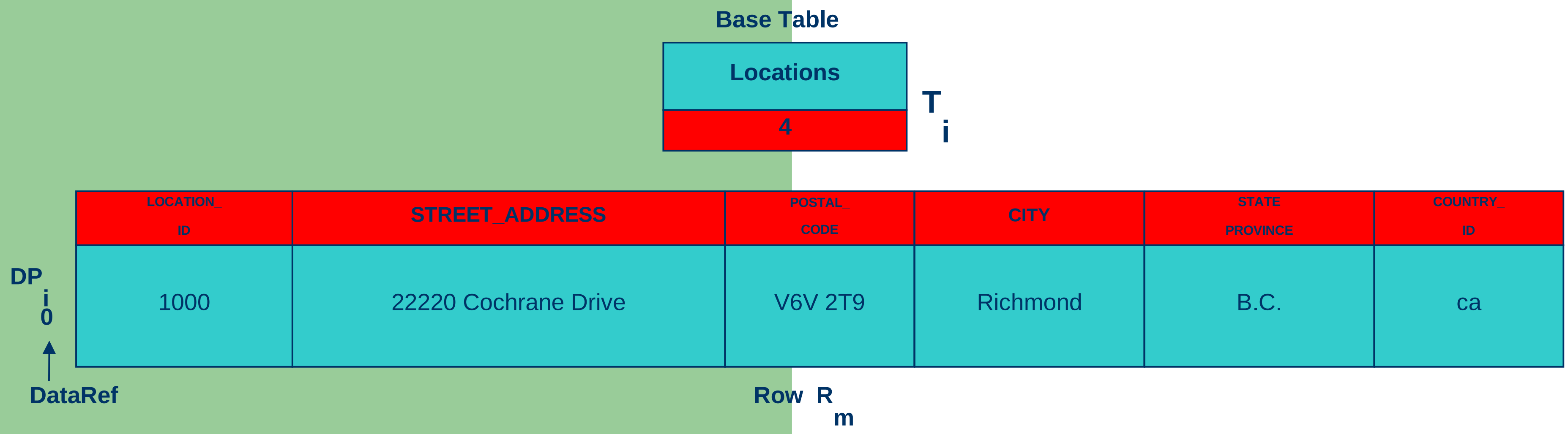
Inserting first row from table Locations

Base Table

Locations
4

LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE PROVINCE	COUNTRY_ID
1000	22220 Cochrane Drive	V6V 2T9	Richmond	B.C.	ca

```
insertRoutine(in BaseTable $T_i$, Row $R_m$, DataRef $DP_i$)
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
        call AddJoinKey($T_i$,$Keys_i$,$InheritedKeys_i$,$DP_i$)
```



Base Table

Locations

4

T_i

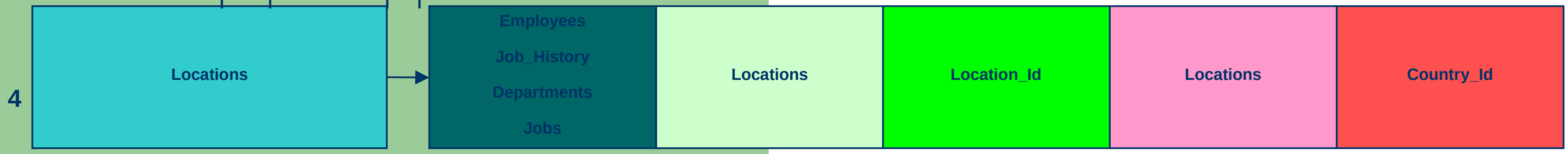
LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE PROVINCE	COUNTRY_ID
1000	22220 Cochrane Drive	V6V 2T9	Richmond	B.C.	ca

DP_{i0}

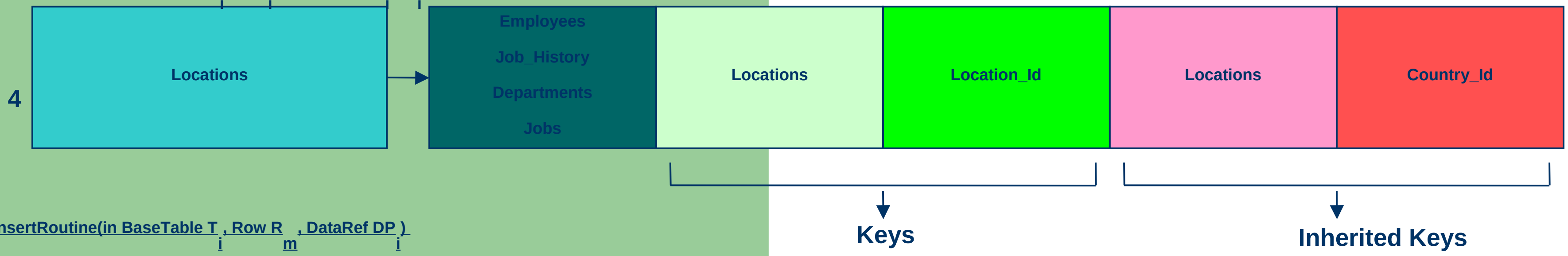
DataRef

Row R_m

```
insertRoutine(in BaseTable $T_i$ , Row $R_m$ , DataRef $DP_i$ )
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
        call AddJoinKey($T_i$ ,$Keys_i$ ,$InheritedKeys_i$ ,$DP_i$ )
```



```
insertRoutine(in BaseTable $T_i$ , Row $R_m$ , DataRef $DP_i$ )
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
        call AddJoinKey($T_i$ ,$Keys_i$ ,$InheritedKeys_i$ ,$DP_i$ )
```



```
insertRoutine(in BaseTable $T_i$ , Row $R_m$ , DataRef $DP_i$ )
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
        call AddJoinKey($T_i$ ,$Keys_i$ ,$InheritedKeys_i$ ,$DP_i$ )
```

DP_{i_0}

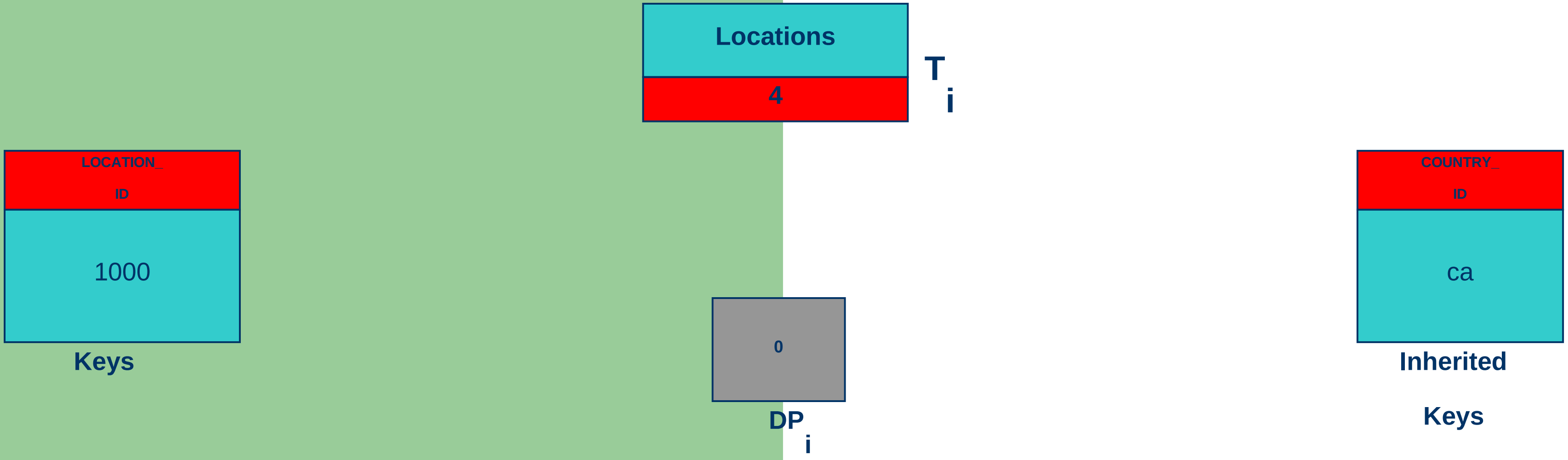
LOCATION_ ID	STREET_ADDRESS	POSTAL_ CODE	CITY	STATE PROVINCE	COUNTRY_ ID
1000	22220 Cochrane Drive	V6V 2T9	Richmond	B.C.	ca

Keys

Row R_m

Inherited Keys

```
insertRoutine(in BaseTable $T_i$, Row $R_m$, DataRef $DP_i$)
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
→       call AddJoinKey($T_i$,$Keys_i$,$InheritedKeys_i$,$DP_i$)
```



Locations

4

T_i

LOCATION_ ID

1000

Keys

0

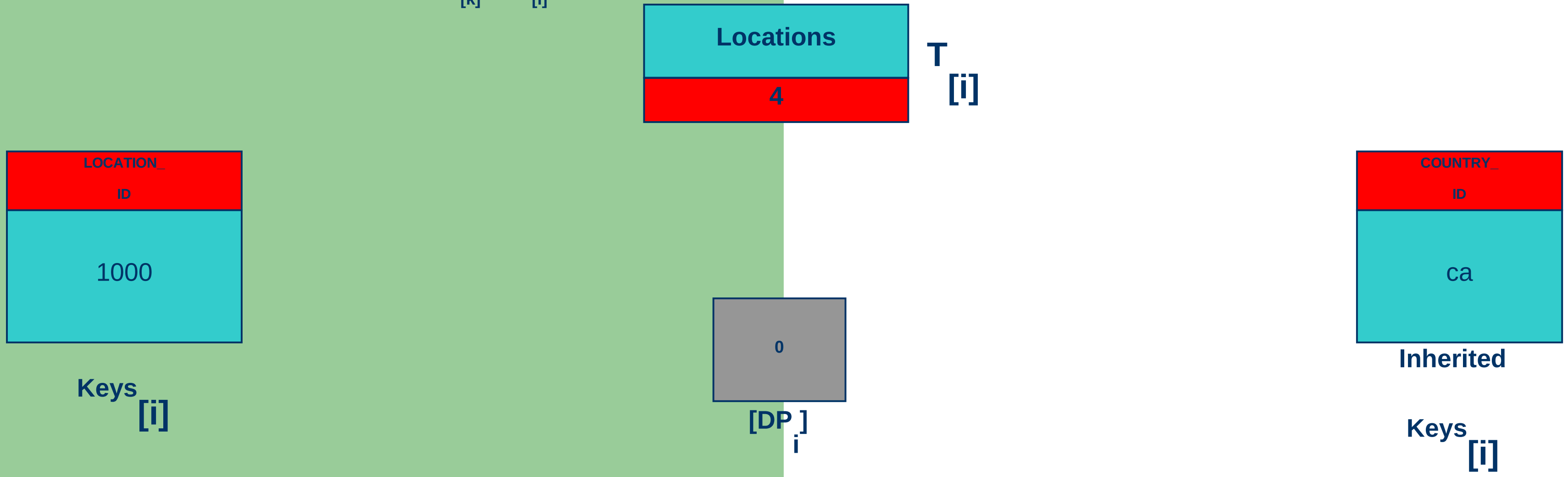
DP_i

COUNTRY_ ID

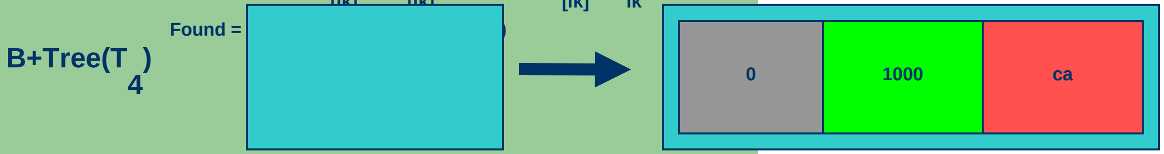
ca

Inherited Keys

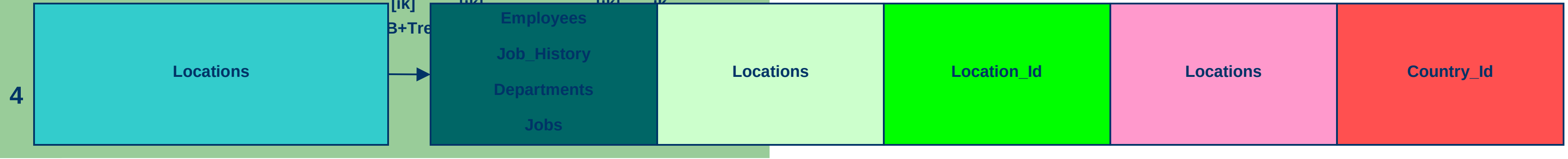
```
→ addJoinKey(in Virtual Table $T_{[i]}$ , $Keys_{[i]}$ , $InheritedKeys_{[i]}$ , $[DP_i]$)
          call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
          locate the entry of $T_{[i]}$ in JoinPathList
          from its adjacentList, locate the table $T_{[k]}$ adjacent to it
                    locate the entry of $T_{[k]}$ in JoinPathList
                    Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
                    while found do
                              returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
                              locate the entry of $T_{[ik]}$ in JoinPathList
                              from its adjacentList, locate the definition of Keys & Inherited Keys
                              from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
                              call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
                              Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```



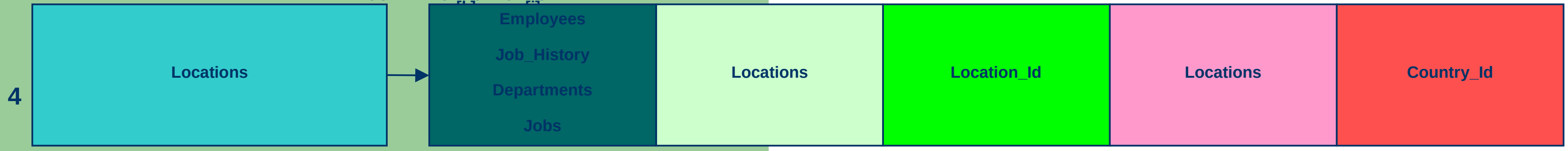
```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
→   call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found =                                      )
```



```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
→   locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            B+Tre
```



```
addJoinKey(in Virtual Table $T_{[i]}$, Keys$_{[i]}$, InheritedKeys$_{[i]}$, [DP$_i$])
    call addKey(B+Tree($T_{[i]}$),Keys$_{[i]}$,InheritedKeys$_{[i]}$,[DP$_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,Keys$_{[k]}$,InheritedKeys$_{[k]}$,[DP$_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,Keys$_{[ik]}$,InheritedKeys$_{[ik]}$,[DP$_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
```



4

Locations

Employees

Job_History

Departments

Jobs

Locations

Location_Id

Locations

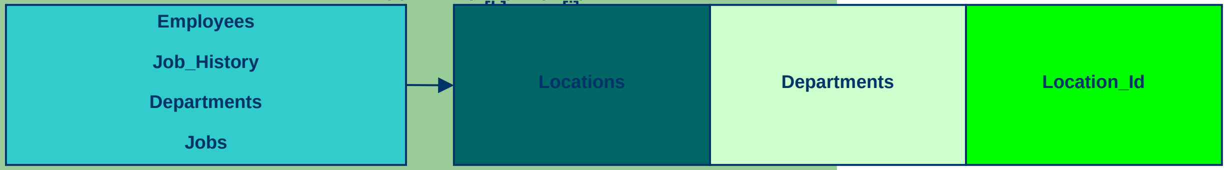
Country_Id

Adjacent

Table

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_{i}$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_{i}$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_{k}$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
            Found = nextKey(B+Tree($T$),Keys)
```

8



```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
→       Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
            Found = nextKey(B+Tree(T),Keys)
```

B+Tree(T_8)

Found: FALSE

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
→       Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

Found: FALSE

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
→           call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

B+Tree(T_0)

0	101	FIN

B+Tree(T_1)

0	101	AC_AUD

B+Tree(T_4)

0	1000	ca

B+Tree(T_6)

0	0	FIN	AC_AUD

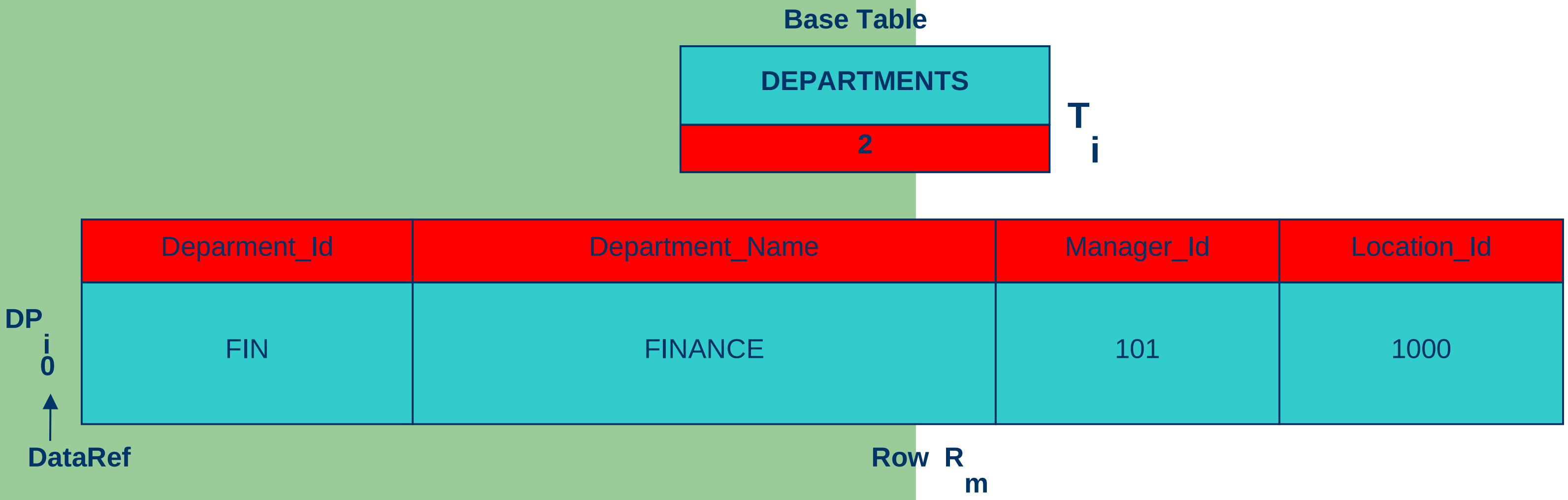
Inserting first row from table Departments

Base Table

DEPARTMENTS
2

Deparment_Id	Department_Name	Manager_Id	Location_Id
FIN	FINANCE	101	1000

```
→ insertRoutine(in BaseTable $T_i$, Row $R_m$, DataRef $DP_i$)
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
        call AddJoinKey($T_i$,$Keys_i$,$InheritedKeys_i$,$DP_i$)
```



Base Table

DEPARTMENTS
2

T_i

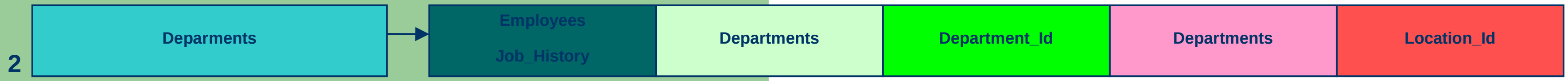
Deparment_Id	Department_Name	Manager_Id	Location_Id
FIN	FINANCE	101	1000

DP_{i0}

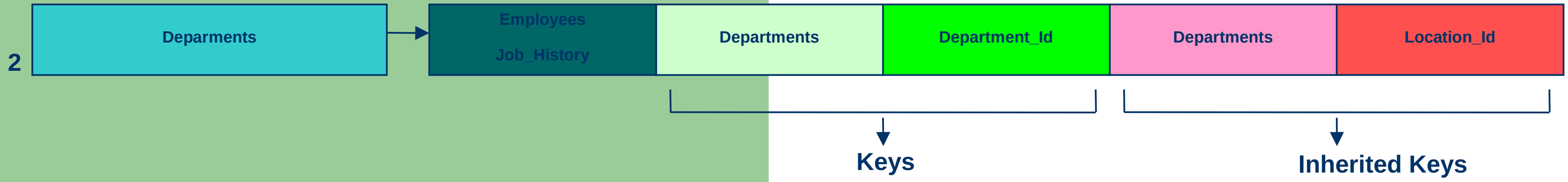
DataRef

Row R_m

```
insertRoutine(in BaseTable $T_i$, Row $R_m$, DataRef $DP_i$)
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
        call AddJoinKey($T_i$,$Keys_i$,$InheritedKeys_i$,$DP_i$)
```



```
insertRoutine(in BaseTable $T_i$, Row $R_m$, DataRef $DP_i$)
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
        call AddJoinKey($T_i$,$Keys_i$,$InheritedKeys_i$,$DP_i$)
```



```
insertRoutine(in BaseTable $T_i$, Row $R_m$, DataRef $DP_i$)
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
        call AddJoinKey($T_i$,$Keys_i$,$InheritedKeys_i$,$DP_i$)
```

DP_{i0}

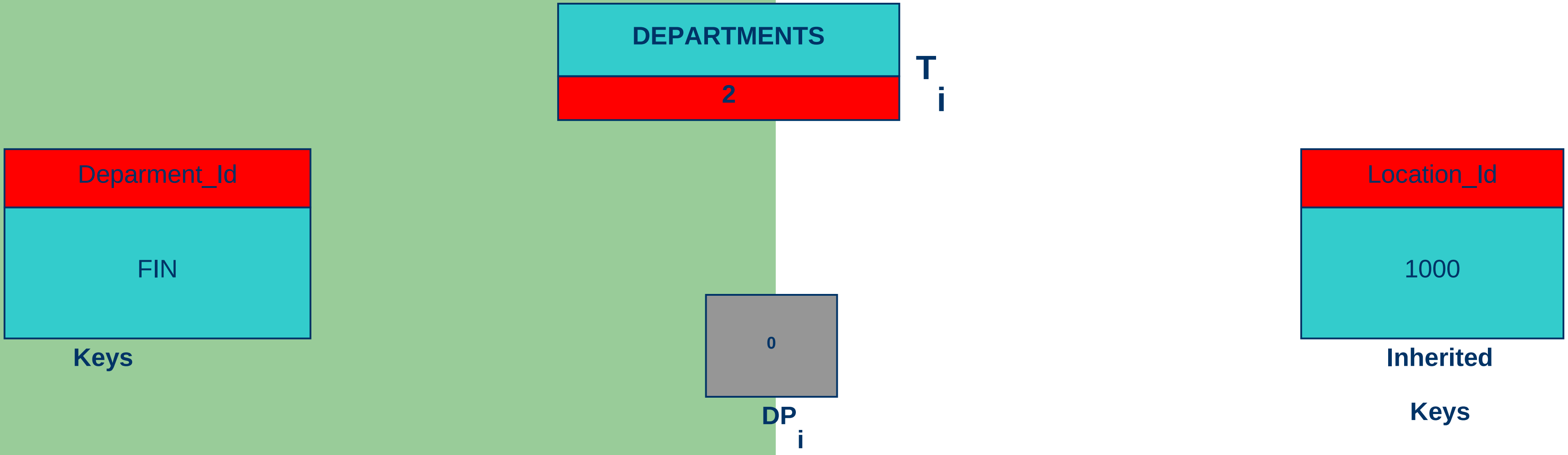
Deparment_Id	Department_Name	Manager_Id	Location_Id
FIN	FINANCE	101	1000

Keys

Row R_m

Inherited Keys

```
insertRoutine(in BaseTable $T_i$ , Row $R_m$ , DataRef $DP_i$ )
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
→       call AddJoinKey($T_i$ ,$Keys_i$ ,$InheritedKeys_i$ ,$DP_i$ )
```



DEPARTMENTS

2

T_i

Deparment_Id

FIN

Keys

Location_Id

1000

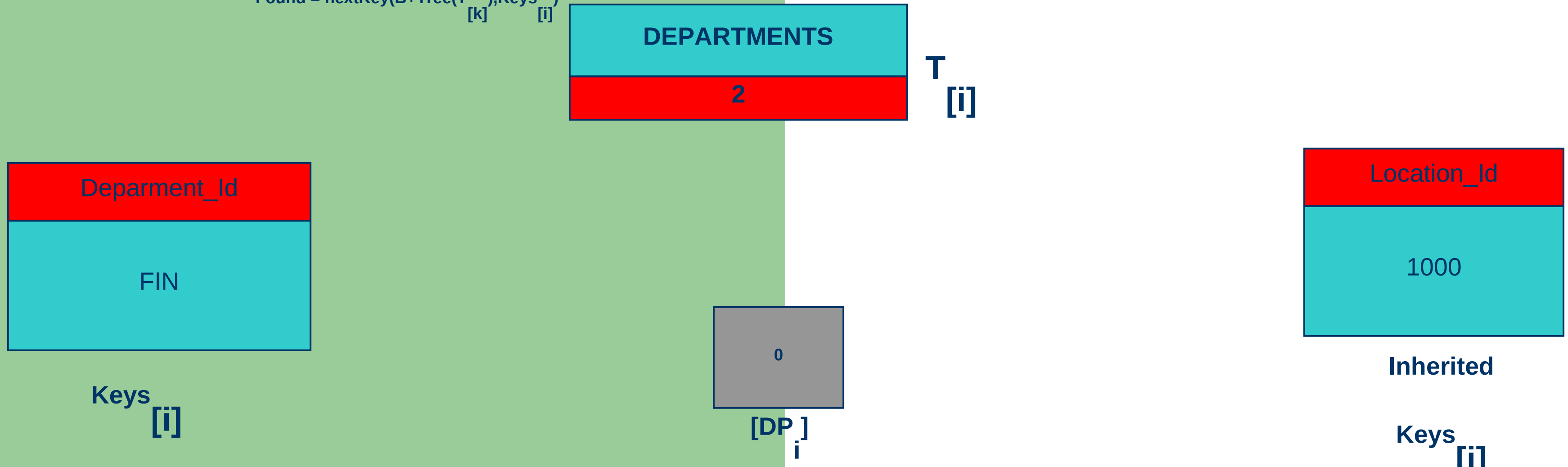
Inherited

Keys

0

DP_i

```
→ addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
          call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
          locate the entry of $T_{[i]}$ in JoinPathList
          from its adjacentList, locate the table $T_{[k]}$ adjacent to it
                locate the entry of $T_{[k]}$ in JoinPathList
                Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
                while found do
                      returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
                      locate the entry of $T_{[ik]}$ in JoinPathList
                      from its adjacentList, locate the definition of Keys & Inherited Keys
                      from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
                      call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
                      Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```



DEPARTMENTS

2

$T_{[i]}$

Deparment_Id

FIN

$Keys_{[i]}$

0

[DP_i]

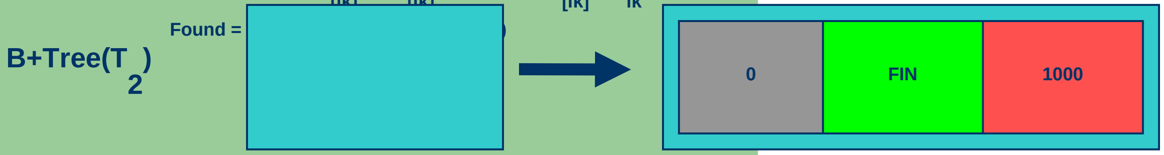
Location_Id

1000

Inherited

$Keys_{[i]}$

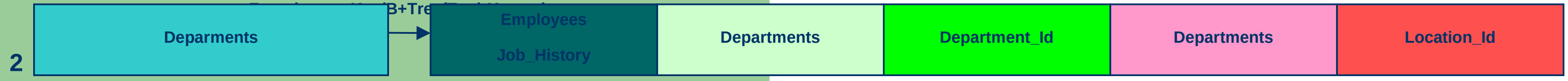
```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = )
```



B+Tree(T_2)

0	FIN	1000

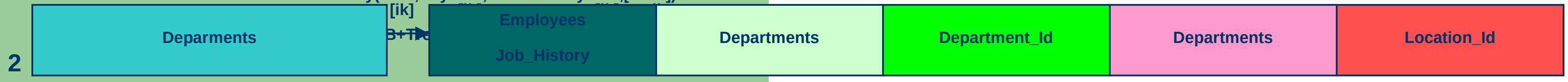
```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            B+Tre
```



2

Deparments	Employees Job_History	Departments	Department_Id	Departments	Location_Id

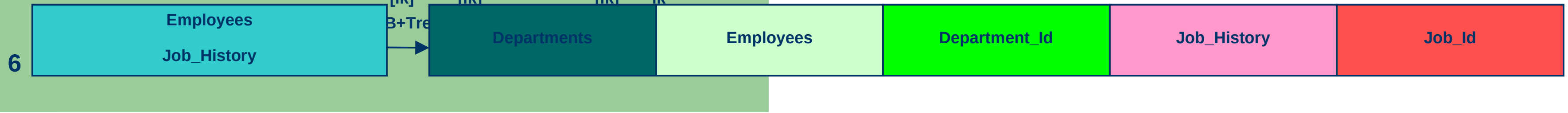
```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
→   from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,Keys,InheritedKeys,[DP ])
            B+Tre
```



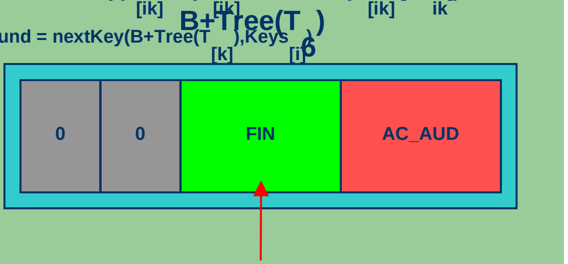
Adjacent

Table

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
→       locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
            B+Tre
```



```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
→       Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```



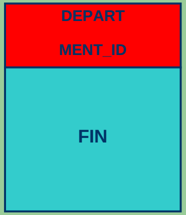
0	0	FIN	AC_AUD

Found: TRUE

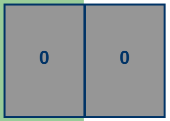
```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
→       Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

Found: TRUE

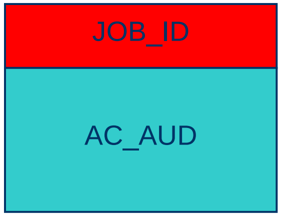
```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```



$Keys_{[k]}$



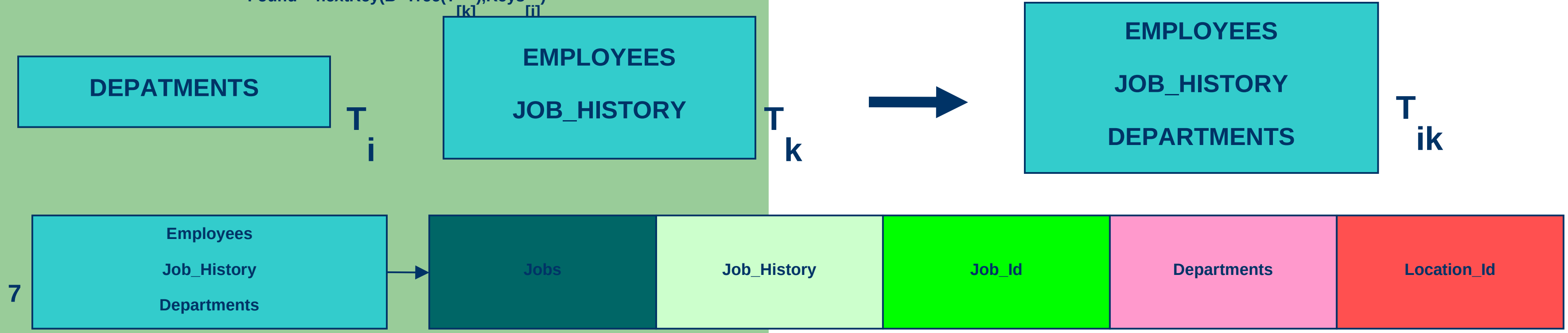
$[DP_k]$



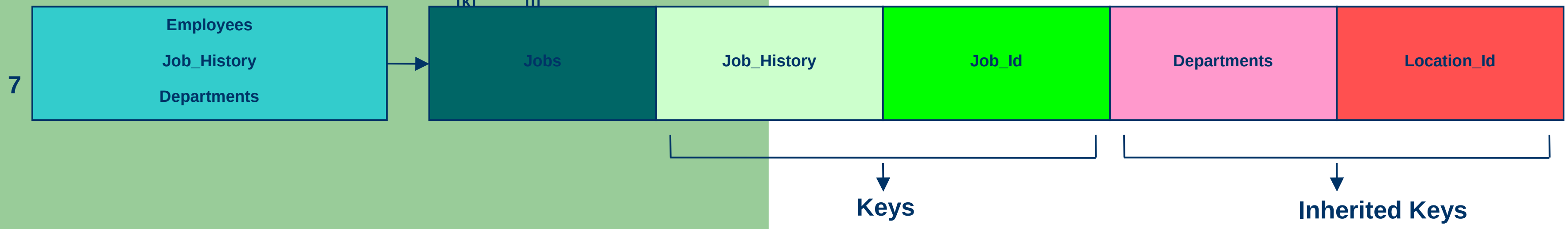
Inherited

$Keys_{[i]}$

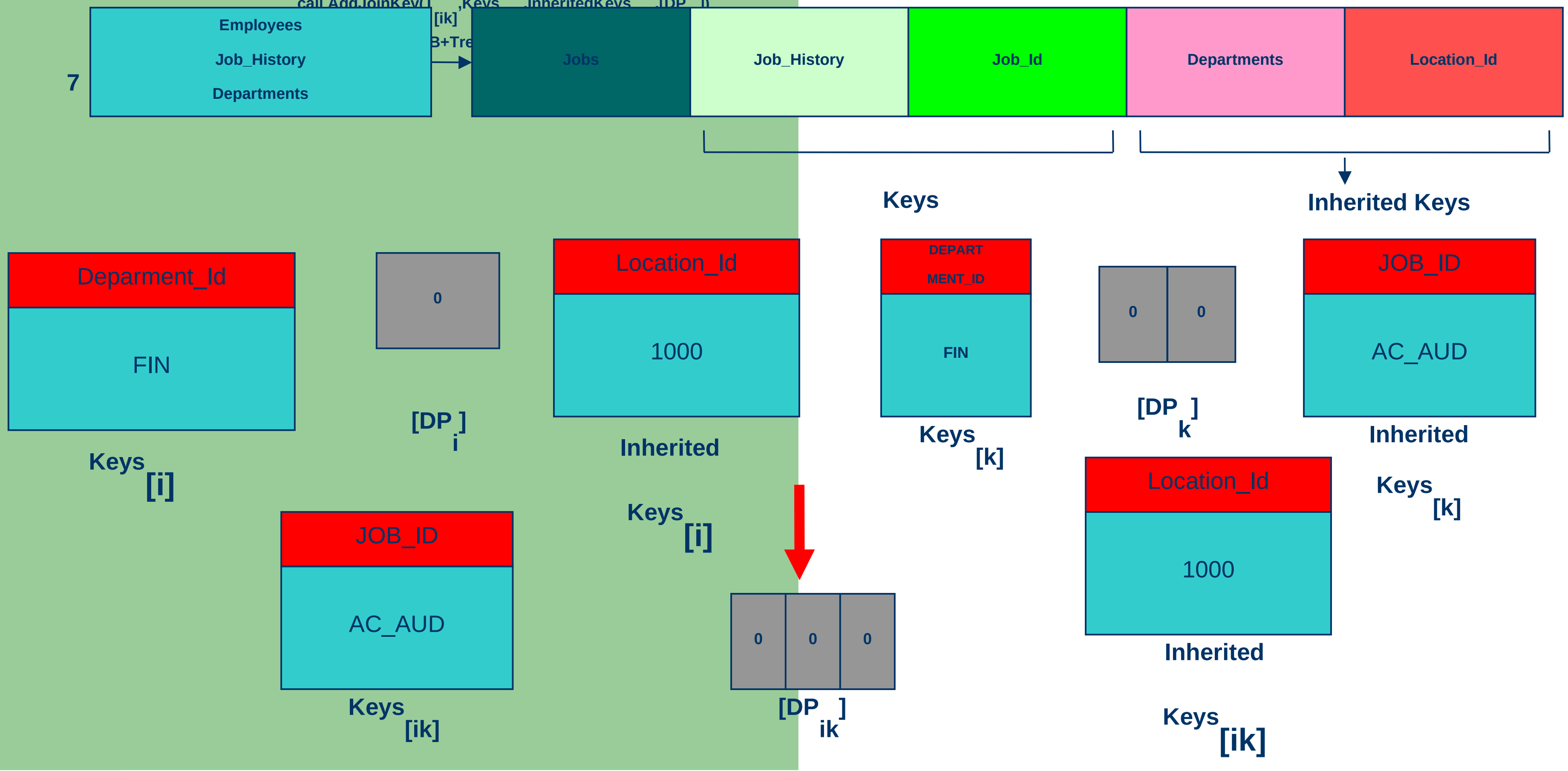
```
addJoinKey(in Virtual Table $T_{[i]}$ , $Keys_{[i]}$ , $InheritedKeys_{[i]}$ , $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```



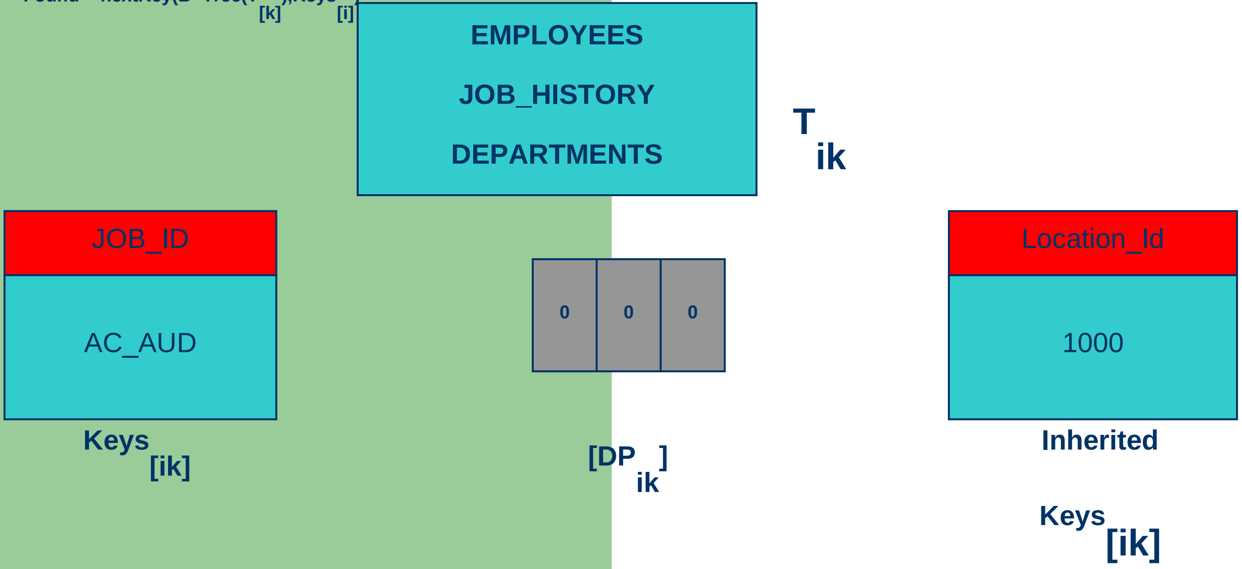
```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```



```
addJoinKey(in Virtual Table $T_{[i]}$ , $Keys_{[i]}$ , $InheritedKeys_{[i]}$ , [$DP_i$])
        call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
        locate the entry of $T_{[i]}$ in JoinPathList
        from its adjacentList, locate the table $T_{[k]}$ adjacent to it
                locate the entry of $T_{[k]}$ in JoinPathList
                Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
                while found do
                        returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
                        locate the entry of $T_{[ik]}$ in JoinPathList
                        from its adjacentList, locate the definition of Keys & Inherited Keys
                        from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
                        call AddJoinKey($T_{[ik]}$ ,Keys     ,InheritedKeys     ,[DP   ])
                        B+Tre
```



```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```



EMPLOYEES

JOB_HISTORY

DEPARTMENTS

T_{ik}

JOB_ID

AC_AUD

$Keys_{[ik]}$

0	0	0

$[DP_{ik}]$

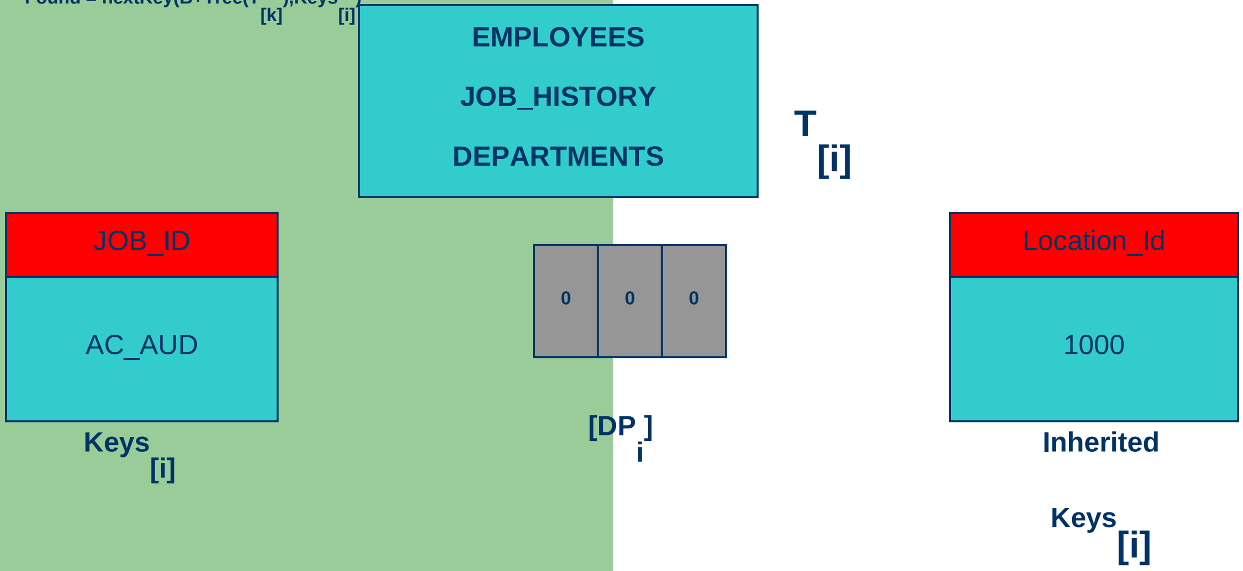
Location_Id

1000

Inherited

$Keys_{[ik]}$

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
→       call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
        locate the entry of $T_{[i]}$ in JoinPathList
        from its adjacentList, locate the table $T_{[k]}$ adjacent to it
            locate the entry of $T_{[k]}$ in JoinPathList
            Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
            while found do
                returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
                locate the entry of $T_{[ik]}$ in JoinPathList
                from its adjacentList, locate the definition of Keys & Inherited Keys
                from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
                call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
                Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```



EMPLOYEES

JOB_HISTORY

DEPARTMENTS

$T_{[i]}$

JOB_ID

AC_AUD

$Keys_{[i]}$

0 0 0

$[DP_i]$

Location_Id

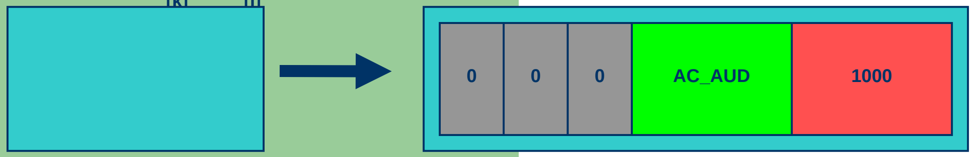
1000

Inherited

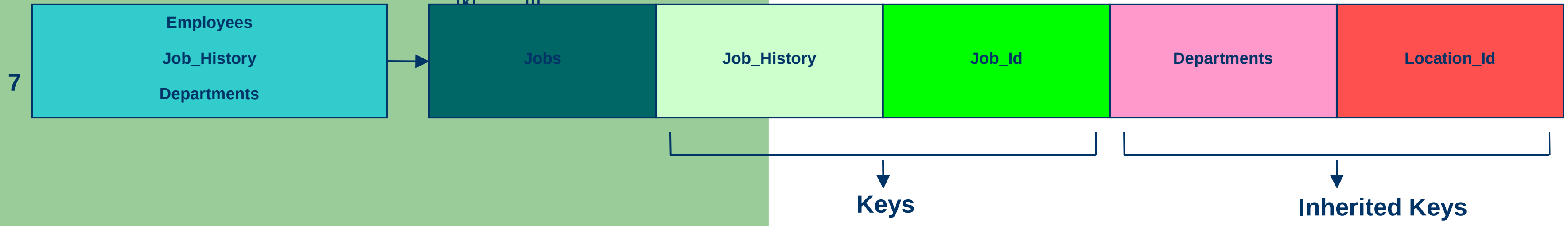
$Keys_{[i]}$

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_{i}]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_{i}]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_{k}]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

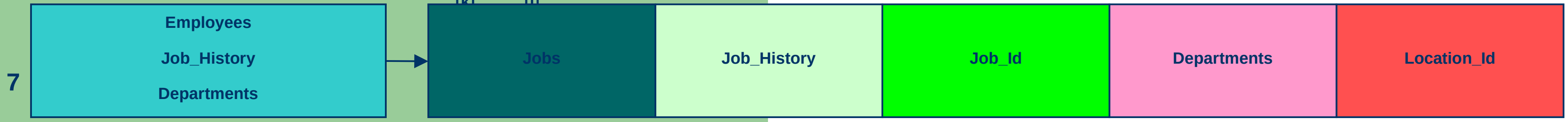
B+Tree(T_7)



```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```



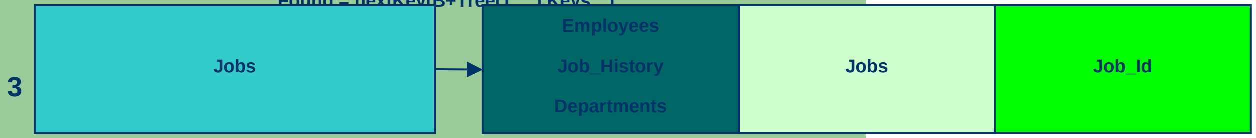
```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
        call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
        locate the entry of $T_{[i]}$ in JoinPathList
→       from its adjacentList, locate the table $T_{[k]}$ adjacent to it
                locate the entry of $T_{[k]}$ in JoinPathList
                Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
                while found do
                        returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
                        locate the entry of $T_{[ik]}$ in JoinPathList
                        from its adjacentList, locate the definition of Keys & Inherited Keys
                        from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
                        call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
                        Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```



Adjacent

Table

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
            Found = nextKey(B+Tree($T$ ),Keys )
```



```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

B+Tree(T_3)

Found: FALSE

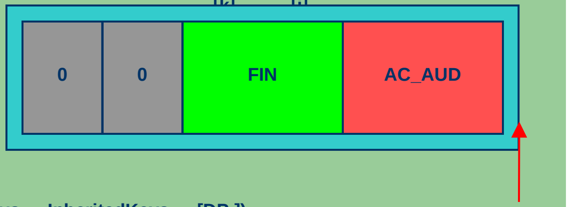
```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_{i}]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_{i}]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
→       Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_{k}]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

Found: FALSE

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_{i}]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_{i}]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_{k}]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
→           call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

B+Tree(T_6)

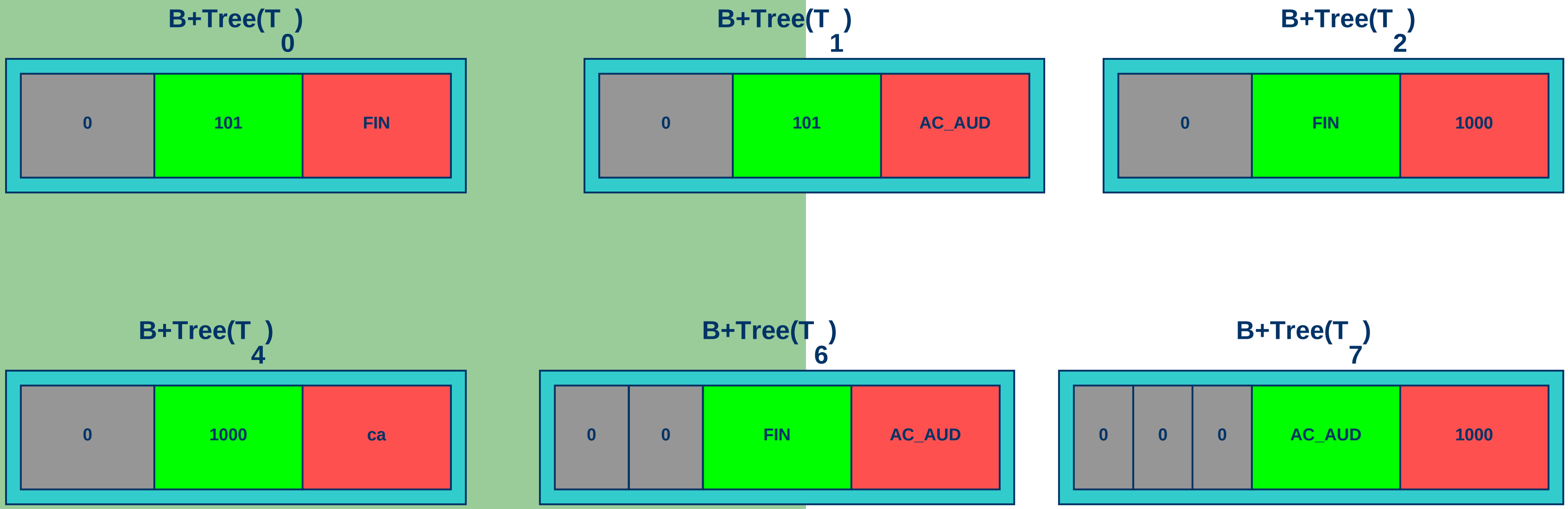


Found: FALSE

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

Found: FALSE

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```



B+Tree(T_0)

0	101	FIN

B+Tree(T_1)

0	101	AC_AUD

B+Tree(T_2)

0	FIN	1000

B+Tree(T_4)

0	1000	ca

B+Tree(T_6)

0	0	FIN	AC_AUD

B+Tree(T_7)

0	0	0	AC_AUD	1000

Inserting first row from table Jobs

Base Table

JOBS
3

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
AC_AUD	Accounting Auditor	30000	60000

```
→ insertRoutine(in BaseTable $T_i$, Row $R_m$, DataRef $DP_i$)
          locate the entry of $T_i$ in JoinPathList
          from its adjacentList, locate the definition of Keys & Inherited Keys
          from row $R_m$ get the columns constituting the keys & Inherited Keys
          call AddJoinKey($T_i$,$Keys_i$,$InheritedKeys_i$,$DP_i$)
```

Base Table

JOBS
3

T_i

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
AC_AUD	Accounting Auditor	30000	60000

DP_i

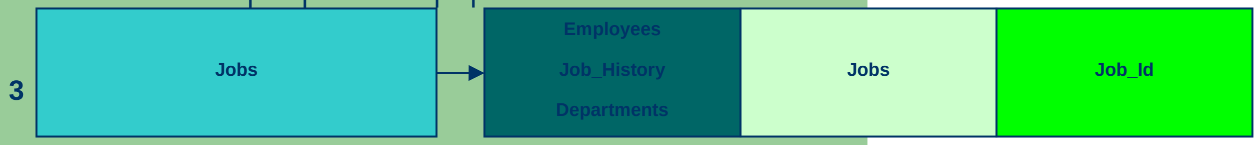
0

↑

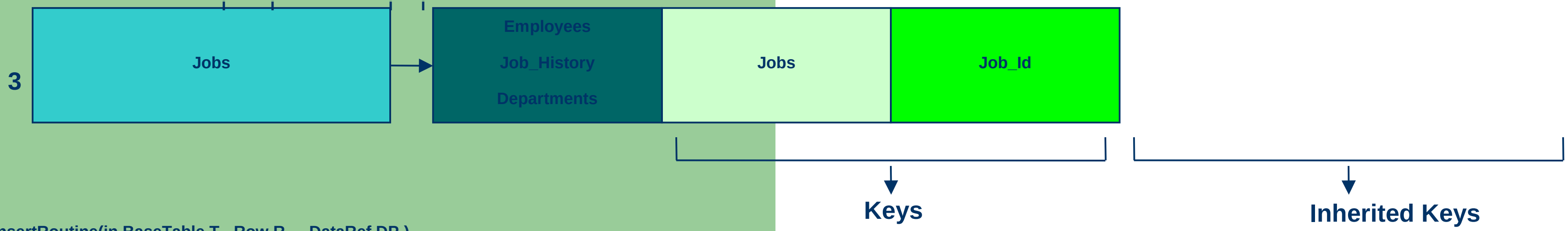
DataRef

Row R_m

```
insertRoutine(in BaseTable $T_i$, Row $R_m$, DataRef $DP_i$)
        locate the entry of $T_i$ in JoinPathList
→       from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
        call AddJoinKey($T_i$,$Keys_i$,$InheritedKeys_i$,$DP_i$)
```



```
insertRoutine(in BaseTable $T_i$, Row $R_m$, DataRef $DP_i$)
        locate the entry of $T_i$ in JoinPathList
→       from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
        call AddJoinKey($T_i$,$Keys_i$,$InheritedKeys_i$,$DP_i$)
```



```
insertRoutine(in BaseTable $T_i$, Row $R_m$, DataRef $DP_i$)
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
→       from row $R_m$ get the columns constituting the keys & Inherited Keys
        call AddJoinKey($T_i$,$Keys_i$,$InheritedKeys_i$,$DP_i$)
```

DP_{i_0}

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
AC_AUD	Accounting Auditor	30000	60000

Keys

Row R_m

```
insertRoutine(in BaseTable $T_i$, Row $R_m$, DataRef $DP_i$)
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
→       call AddJoinKey($T_i$,$Keys_i$,$InheritedKeys_i$,$DP_i$)
```

JOBS

3

T_i

JOB_ID

AC_AUD

Keys

0

DP_i

```
→ addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
          call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
          locate the entry of $T_{[i]}$ in JoinPathList
          from its adjacentList, locate the table $T_{[k]}$ adjacent to it
                    locate the entry of $T_{[k]}$ in JoinPathList
                    Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
                    while found do
                              returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
                              locate the entry of $T_{[ik]}$ in JoinPathList
                              from its adjacentList, locate the definition of Keys & Inherited Keys
                              from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
                              call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
                              Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

JOBS

3

$T_{[i]}$

JOB_ID

AC_AUD

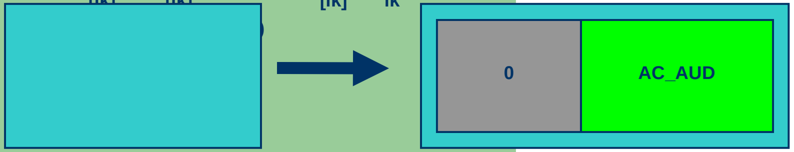
$Keys_{[i]}$

0

$[DP_i]$

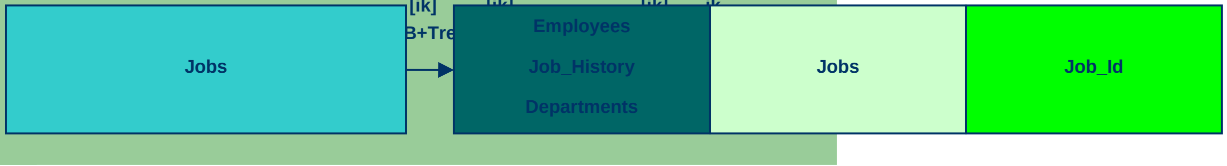
```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
→       call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
        locate the entry of $T_{[i]}$ in JoinPathList
        from its adjacentList, locate the table $T_{[k]}$ adjacent to it
                locate the entry of $T_{[k]}$ in JoinPathList
                Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
                while found do
                        returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
                        locate the entry of $T_{[ik]}$ in JoinPathList
                        from its adjacentList, locate the definition of Keys & Inherited Keys
                        from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
                        call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
                        Found =
```

B+Tree(T_3)

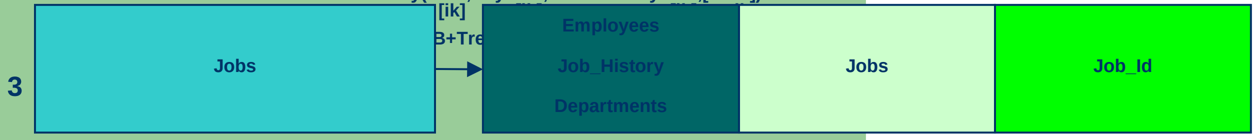


```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
        call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
→       locate the entry of $T_{[i]}$ in JoinPathList
        from its adjacentList, locate the table $T_{[k]}$ adjacent to it
                locate the entry of $T_{[k]}$ in JoinPathList
                Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
                while found do
                        returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
                        locate the entry of $T_{[ik]}$ in JoinPathList
                        from its adjacentList, locate the definition of Keys & Inherited Keys
                        from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
                        call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
```

3



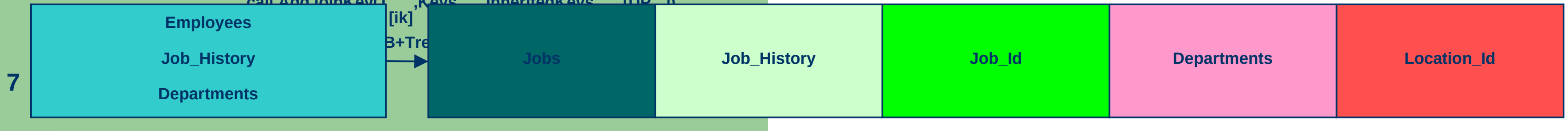
```
addJoinKey(in Virtual Table $T_{[i]}$, Keys$_{[i]}$, InheritedKeys$_{[i]}$, [DP$_i$])
    call addKey(B+Tree($T_{[i]}$),Keys$_{[i]}$,InheritedKeys$_{[i]}$,[DP$_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,Keys$_{[k]}$,InheritedKeys$_{[k]}$,[DP$_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,Keys,InheritedKeys,[DP])
            B+Tre
```



Adjacent

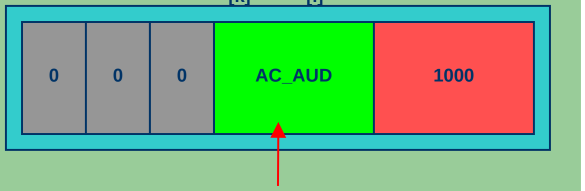
Table

```
addJoinKey(in Virtual Table $T_{[i]}$, Keys$_{[i]}$, InheritedKeys$_{[i]}$, [DP$_i$])
    call addKey(B+Tree($T_{[i]}$),Keys$_{[i]}$,InheritedKeys$_{[i]}$,[DP$_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,Keys$_{[k]}$,InheritedKeys$_{[k]}$,[DP$_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,Keys,InheritedKeys,[DP])
            B+Tre
```



```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

B+Tree(T)



0	0	0	AC_AUD	1000

Found: TRUE

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

Found: TRUE

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

JOB_ID
AC_AUD

$Keys_{[k]}$

0	0	0

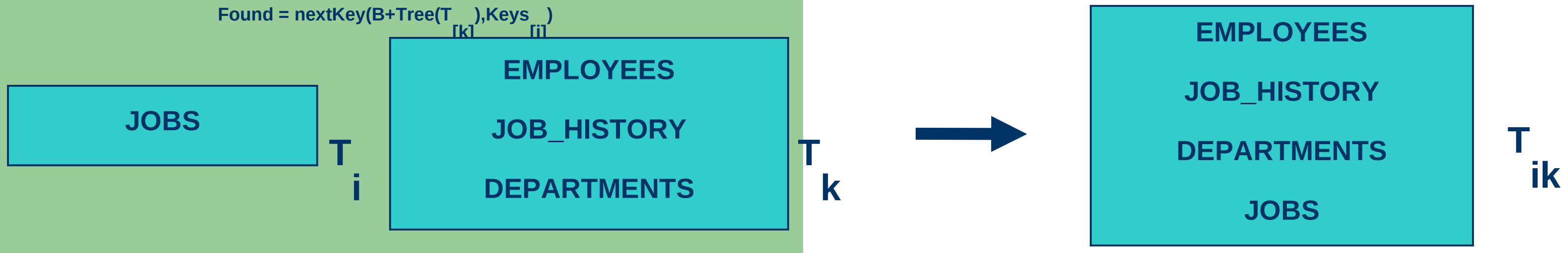
$[DP_k]$

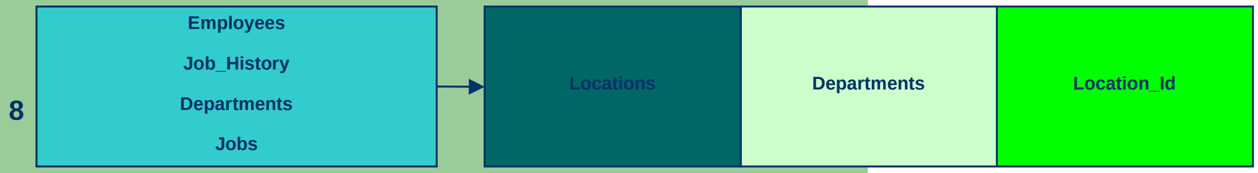
Location_Id
1000

Inherited

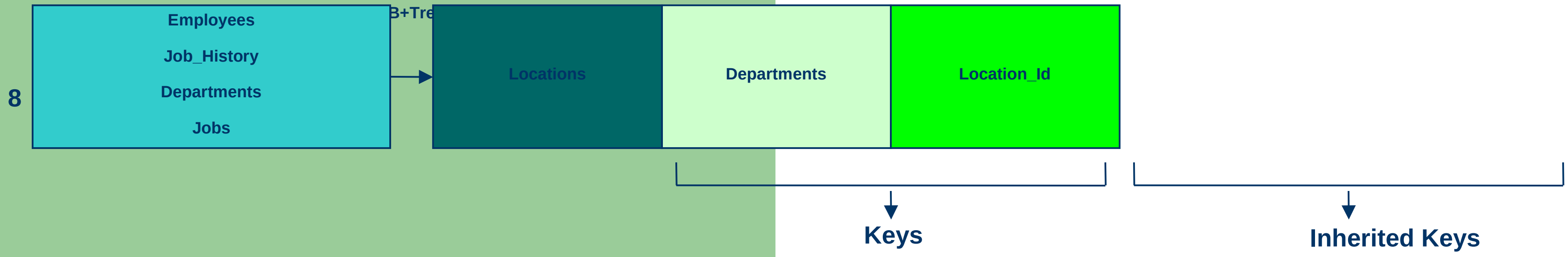
$Keys_{[i]}$

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
→           returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

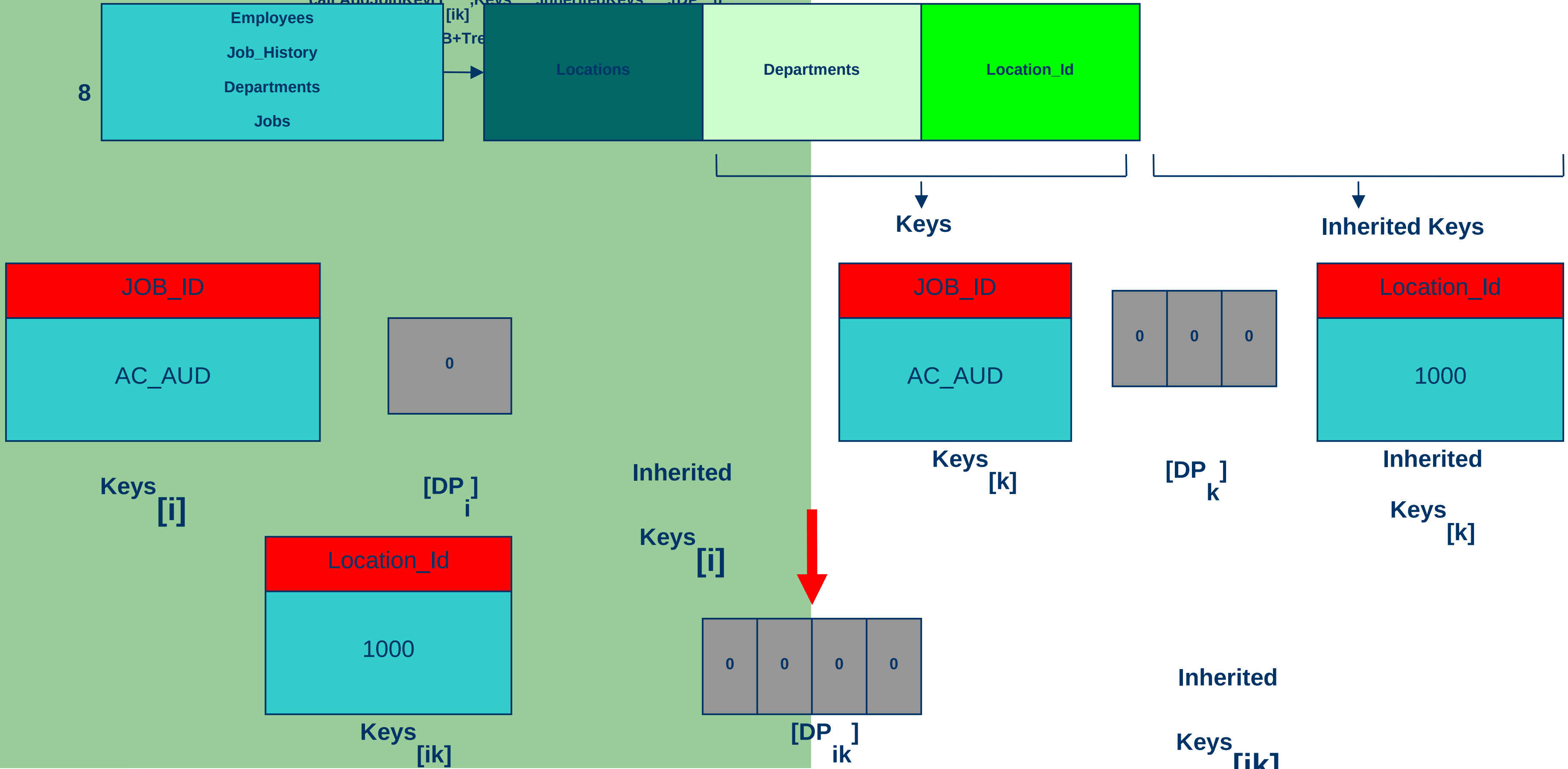




```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
```



```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
          call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
          locate the entry of $T_{[i]}$ in JoinPathList
          from its adjacentList, locate the table $T_{[k]}$ adjacent to it
                    locate the entry of $T_{[k]}$ in JoinPathList
                    Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
                    while found do
                              returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
                              locate the entry of $T_{[ik]}$ in JoinPathList
                              from its adjacentList, locate the definition of Keys & Inherited Keys
                              from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
                              call AddJoinKey($T_{[ik]}$,Keys   ,InheritedKeys   ,[DP   ])
                              B+Tre
```



8

Employees

Job_History

Departments

Jobs

Locations

Departments

Location_Id

Keys

Inherited Keys

JOB_ID

AC_AUD

Keys$_{[i]}$

0

$[DP_i]$

Inherited

Keys$_{[i]}$

Location_Id

1000

Keys$_{[ik]}$

0 0 0 0

$[DP_{ik}]$

JOB_ID

AC_AUD

Keys$_{[k]}$

0 0 0

$[DP_k]$

Location_Id

1000

Inherited

Keys$_{[k]}$

Inherited

Keys$_{[ik]}$

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
→           from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

EMPLOYEES

JOB_HISTORY

DEPARTMENTS

JOBS

T_{ik}

Location_Id
1000

$Keys_{[ik]}$

0	0	0	0

$[DP_{ik}]$

Inherited

$Keys_{[ik]}$

```
→ addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
      call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
      locate the entry of $T_{[i]}$ in JoinPathList
      from its adjacentList, locate the table $T_{[k]}$ adjacent to it
            locate the entry of $T_{[k]}$ in JoinPathList
            Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
            while found do
                  returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
                  locate the entry of $T_{[ik]}$ in JoinPathList
                  from its adjacentList, locate the definition of Keys & Inherited Keys
                  from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
                  call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
                  Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

EMPLOYEES

JOB_HISTORY

DEPARTMENTS

JOBS

$T_{[i]}$

Location_Id
1000

$Keys_{[i]}$

0	0	0	0

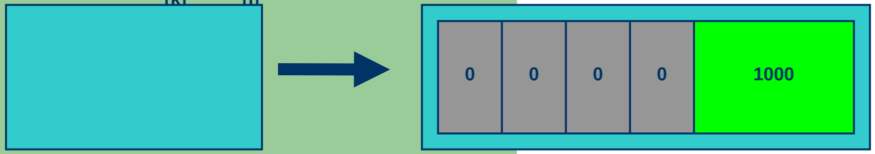
$[DP_i]$

Inherited

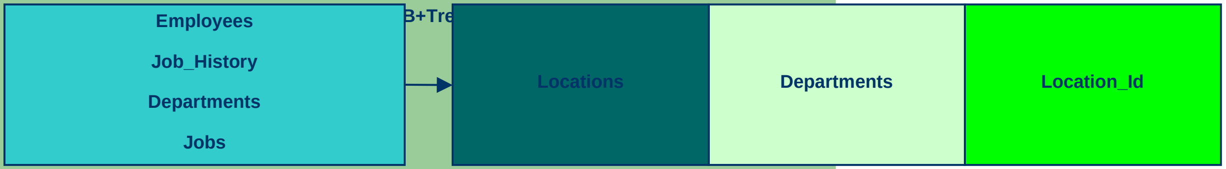
$Keys_{[i]}$

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

$B+Tree(T_8)$

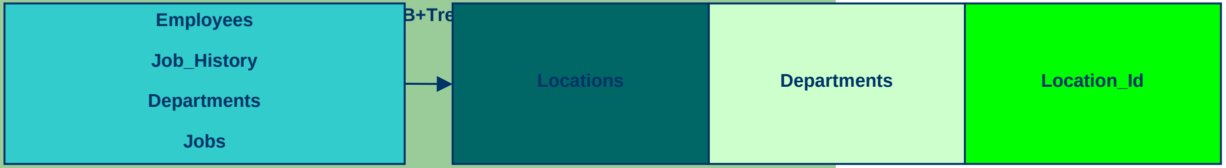


```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
            B+Tre
```



8

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
            B+Tre
```

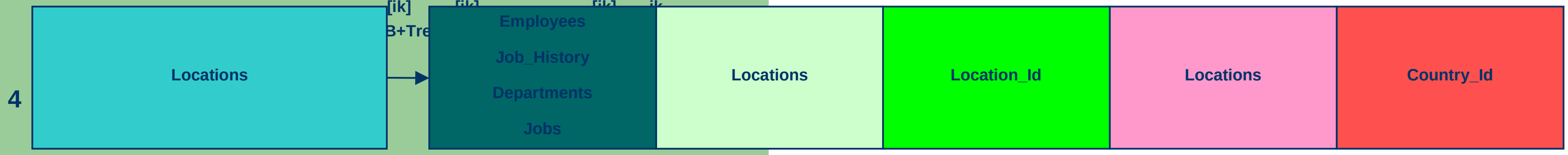


8

Adjacent

Table

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
          call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
          locate the entry of $T_{[i]}$ in JoinPathList
          from its adjacentList, locate the table $T_{[k]}$ adjacent to it
                    locate the entry of $T_{[k]}$ in JoinPathList
                    Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
                    while found do
                              returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
                              locate the entry of $T_{[ik]}$ in JoinPathList
                              from its adjacentList, locate the definition of Keys & Inherited Keys
                              from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
                              call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
                                        B+Tre
```



4

Locations

Employees

Job_History

Departments

Jobs

Locations

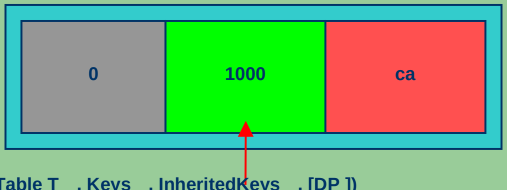
Location_Id

Locations

Country_Id

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

B+Tree(T_4)

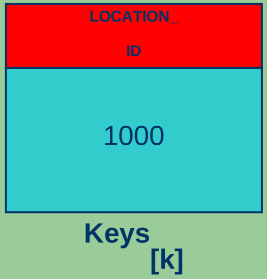


Found: TRUE

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

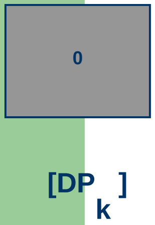
Found: TRUE

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

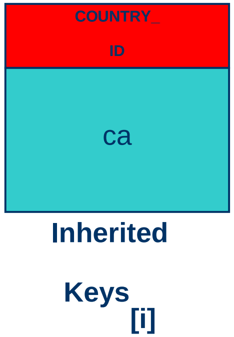

LOCATION_ID

1000

$Keys_{[k]}$


0

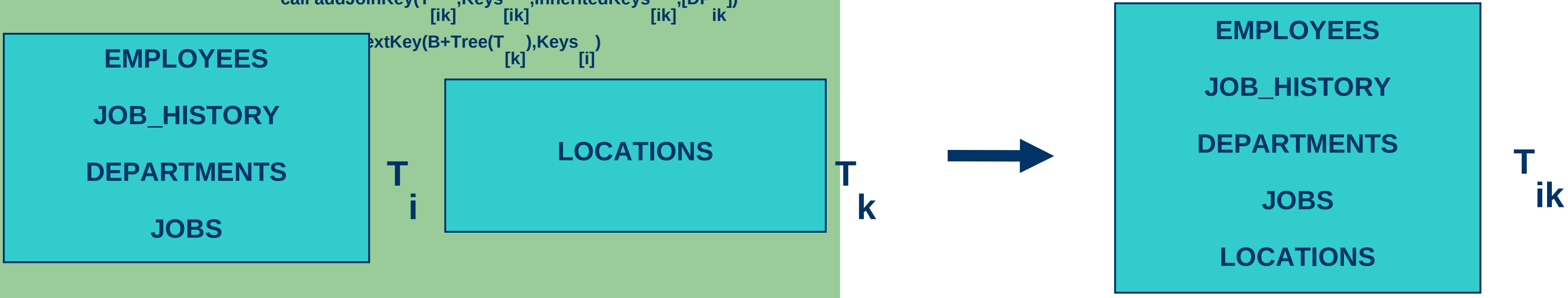
$[DP_k]$


COUNTRY_ID

ca

Inherited $Keys_{[i]}$

```
addJoinKey(in Virtual Table $T_{[i]}$, Keys$_{[i]}$, InheritedKeys$_{[i]}$, [DP$_i$])
    call addKey(B+Tree($T_{[i]}$),Keys$_{[i]}$,InheritedKeys$_{[i]}$,[DP$_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,Keys$_{[k]}$,InheritedKeys$_{[k]}$,[DP$_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,Keys$_{[ik]}$,InheritedKeys$_{[ik]}$,[DP$_{ik}$])
            extKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
```



EMPLOYEES
JOB_HISTORY
DEPARTMENTS
JOBS

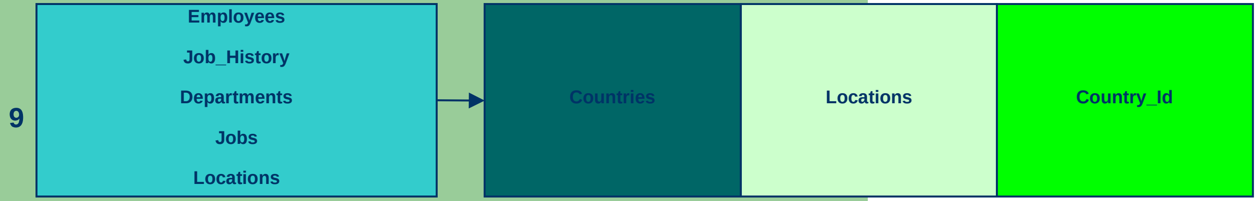
T_i

LOCATIONS

T_k

EMPLOYEES
JOB_HISTORY
DEPARTMENTS
JOBS
LOCATIONS

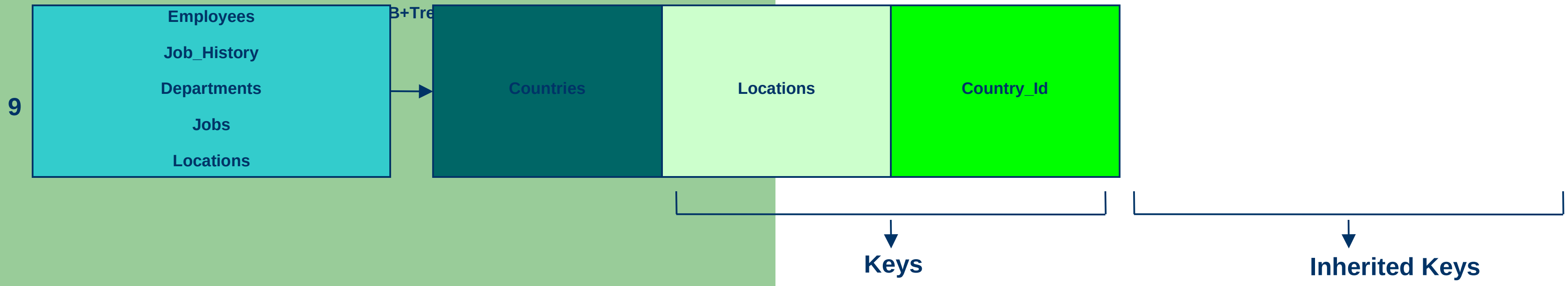
T_{ik}



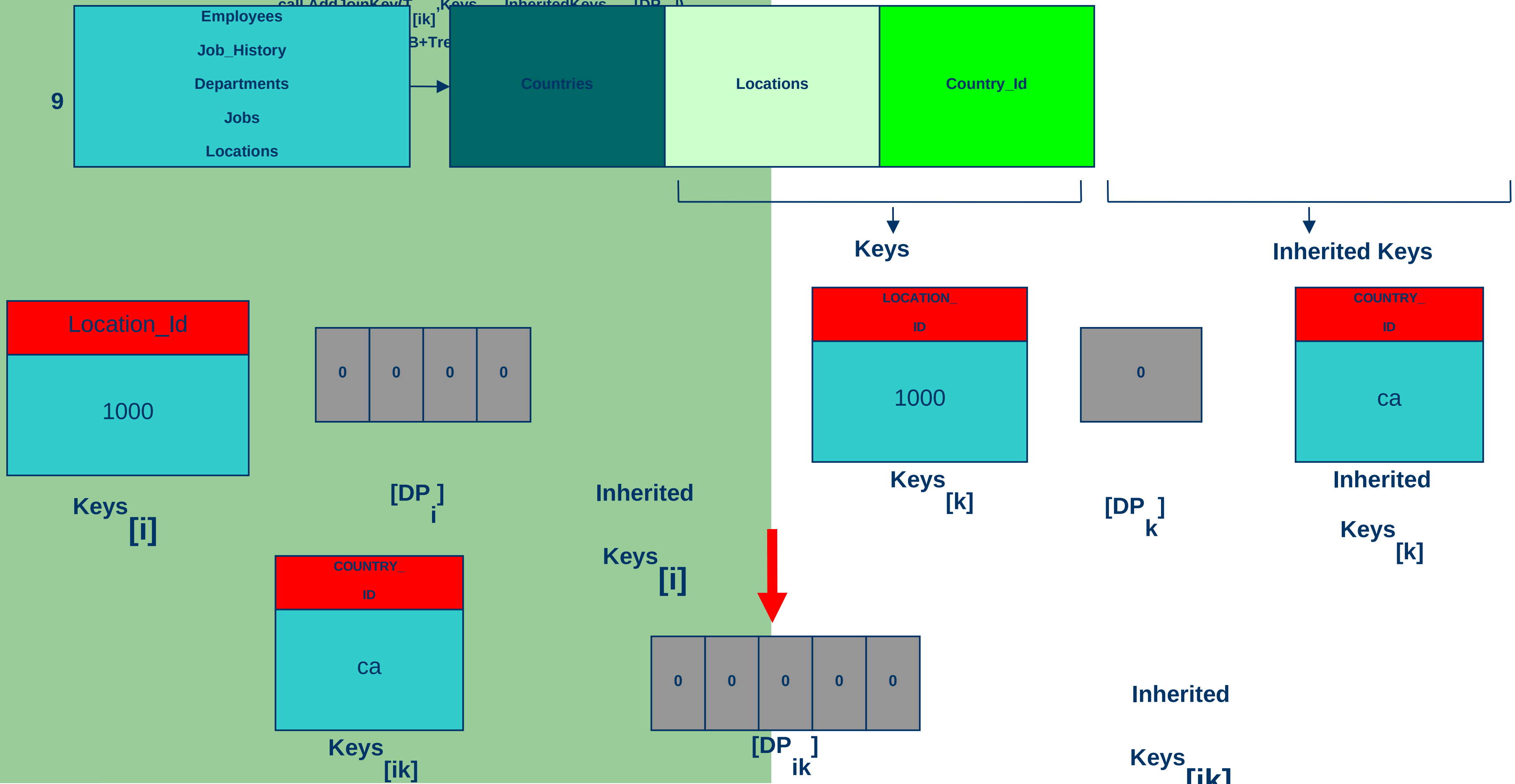
9

Employees Job_History Departments Jobs Locations	Countries	Locations	Country_Id

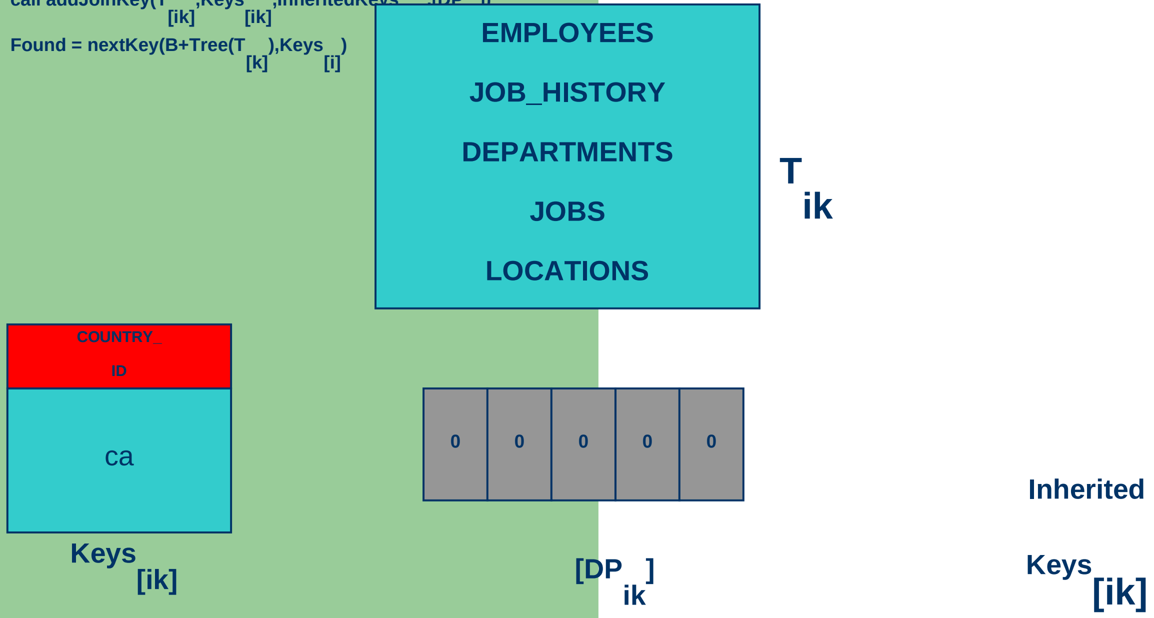
```
addJoinKey(in Virtual Table $T_{[i]}$ , $Keys_{[i]}$ , $InheritedKeys_{[i]}$ , $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
→           locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
```



```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,Keys ,InheritedKeys ,[DP ])
            B+Tre
```



```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,InheritedKeys ,[DP ])
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```



EMPLOYEES

JOB_HISTORY

DEPARTMENTS

JOBS

LOCATIONS

T_{ik}

COUNTRY_ ID

ca

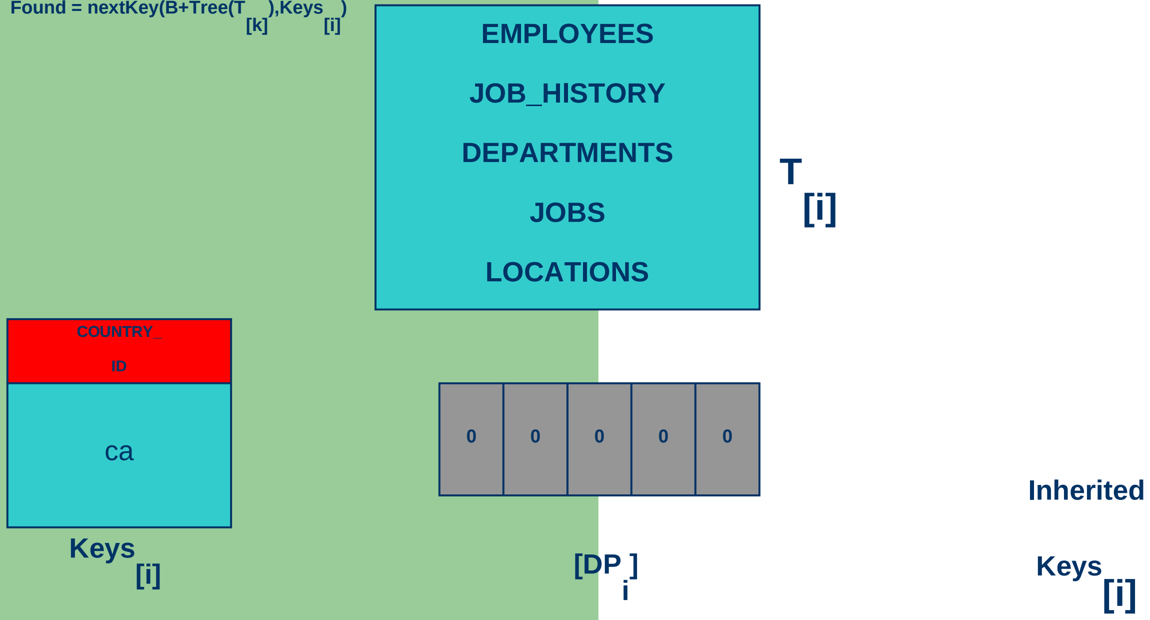
$Keys_{[ik]}$

0	0	0	0	0

[DP_{ik}]

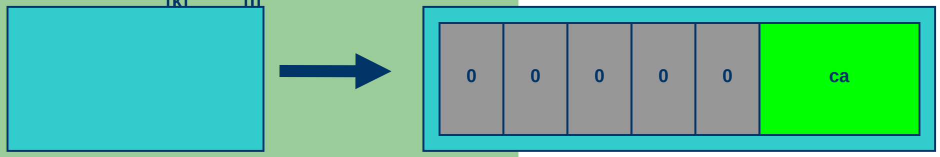
Inherited Keys$_{[ik]}$

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

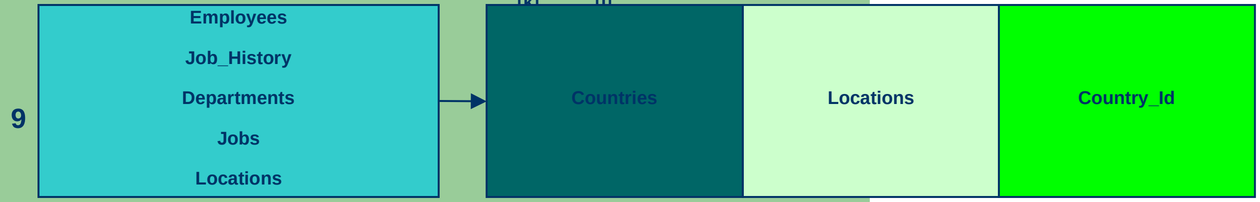


```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

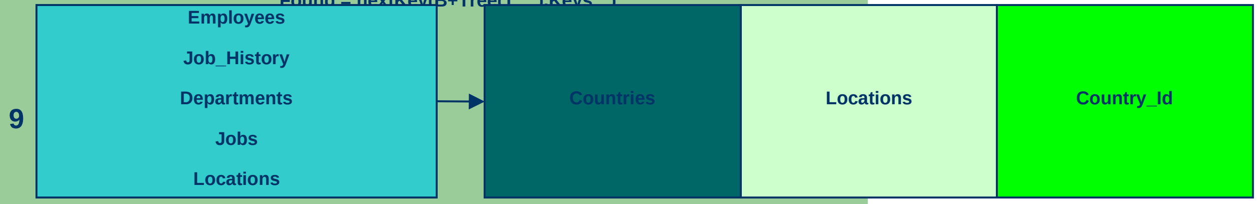
B+Tree(T_9)



```
addJoinKey(in Virtual Table $T_{[i]}$ , $Keys_{[i]}$ , $InheritedKeys_{[i]}$ , [$DP_i$ ])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```



```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T$ ),Keys )
```



9

Employees

Job_History

Departments

Jobs

Locations

Countries

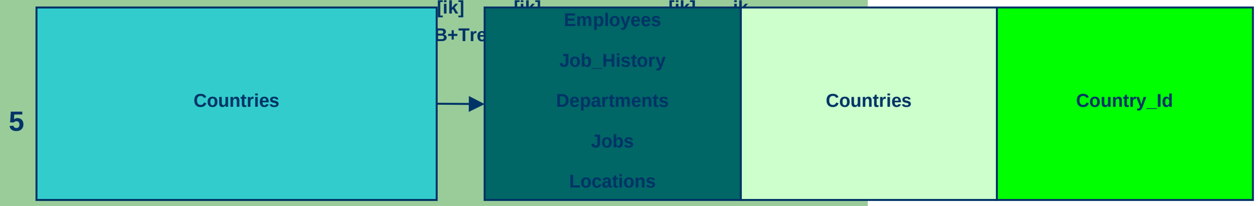
Locations

Country_Id

Adjacent

Table

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
→       locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            B+Tre
```



5

Countries

Employees

Job_History

Departments

Jobs

Locations

Countries

Country_Id

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
→       Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

B+Tree(T_5)

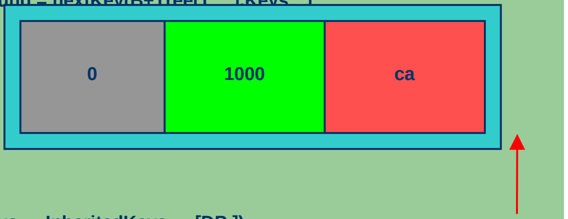
Found: FALSE

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
→       Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)            Found: FALSE
```

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
→           call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree(T ),Keys )
```

B+Tree(T_4)



Found: FALSE

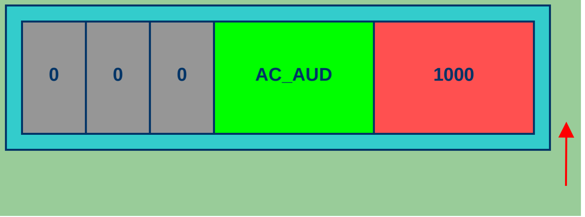
```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

Found: FALSE

```
addJoinKey(in Virtual Table $T_{[i]}$, Keys$_{[i]}$, InheritedKeys$_{[i]}$, [DP$_{i}$])
    call addKey(B+Tree($T_{[i]}$),Keys$_{[i]}$,InheritedKeys$_{[i]}$,[DP$_{i}$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,Keys$_{[k]}$,InheritedKeys$_{[k]}$,[DP$_{k}$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,Keys$_{[ik]}$,InheritedKeys$_{[ik]}$,[DP$_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
```

```
addJoinKey(in Virtual Table $T_{[i]}$, Keys$_{[i]}$, InheritedKeys$_{[i]}$, [DP$_{i}$])
    call addKey(B+Tree($T_{[i]}$),Keys$_{[i]}$,InheritedKeys$_{[i]}$,[DP$_{i}$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,Keys$_{[k]}$,InheritedKeys$_{[k]}$,[DP$_{k}$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,Keys$_{[ik]}$,InheritedKeys$_{[ik]}$,[DP$_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
```

B+Tree(T_7)

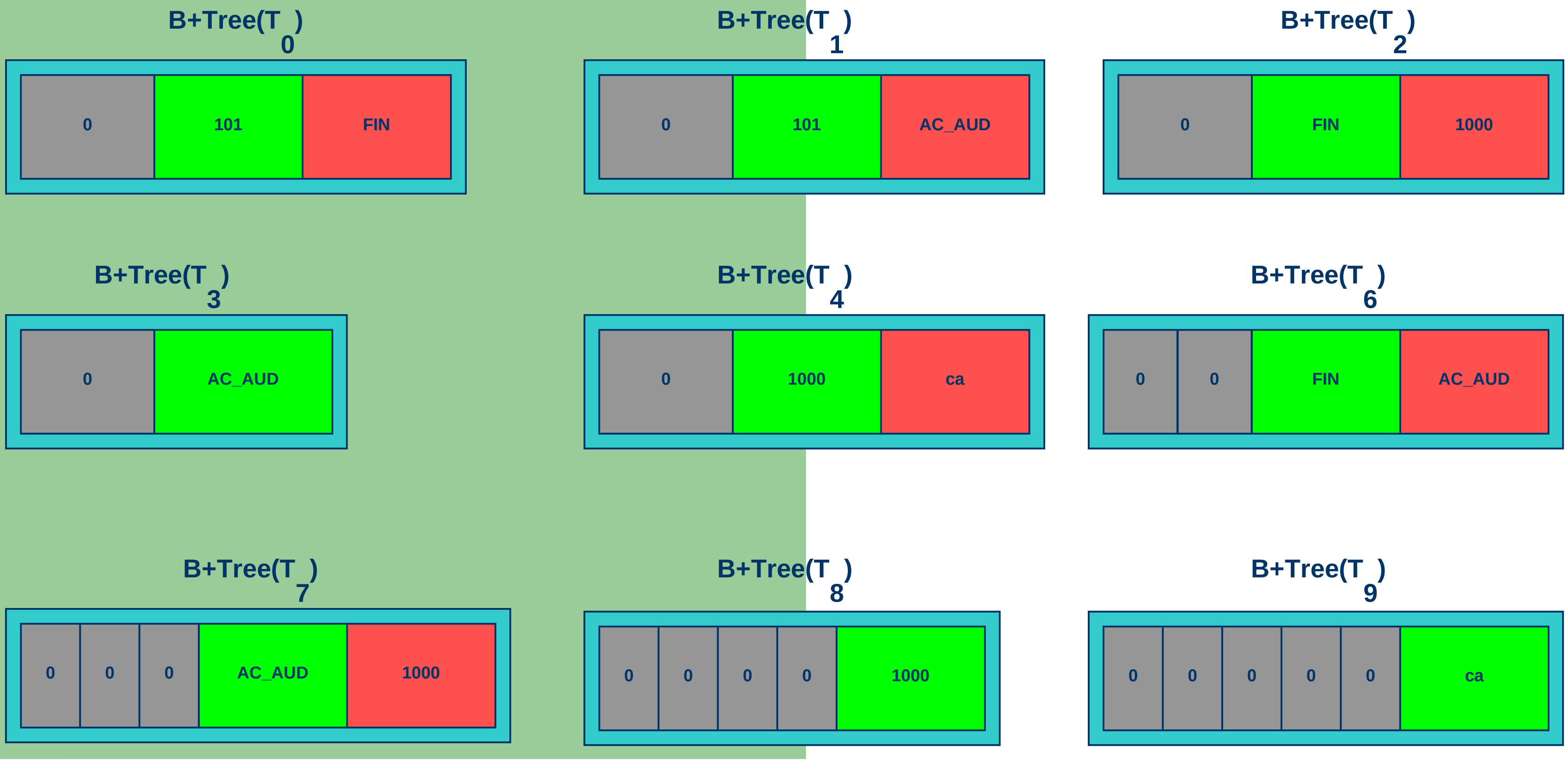


Found: FALSE

```
addJoinKey(in Virtual Table $T_{[i]}$, Keys$_{[i]}$, InheritedKeys$_{[i]}$, [DP$_i$])
    call addKey(B+Tree($T_{[i]}$),Keys$_{[i]}$,InheritedKeys$_{[i]}$,[DP$_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
→       while found do
            returnKeys(B+Tree($T_{[k]}$,Keys$_{[k]}$,InheritedKeys$_{[k]}$,[DP$_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,Keys$_{[ik]}$,InheritedKeys$_{[ik]}$,[DP$_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
```

Found: FALSE

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```



B+Tree(T_0)

0	101	FIN

B+Tree(T_1)

0	101	AC_AUD

B+Tree(T_2)

0	FIN	1000

B+Tree(T_3)

0	AC_AUD

B+Tree(T_4)

0	1000	ca

B+Tree(T_6)

0	0	FIN	AC_AUD

B+Tree(T_7)

0	0	0	AC_AUD	1000

B+Tree(T_8)

0	0	0	0	1000

B+Tree(T_9)

0	0	0	0	0	ca

Inserting first row from table Countries

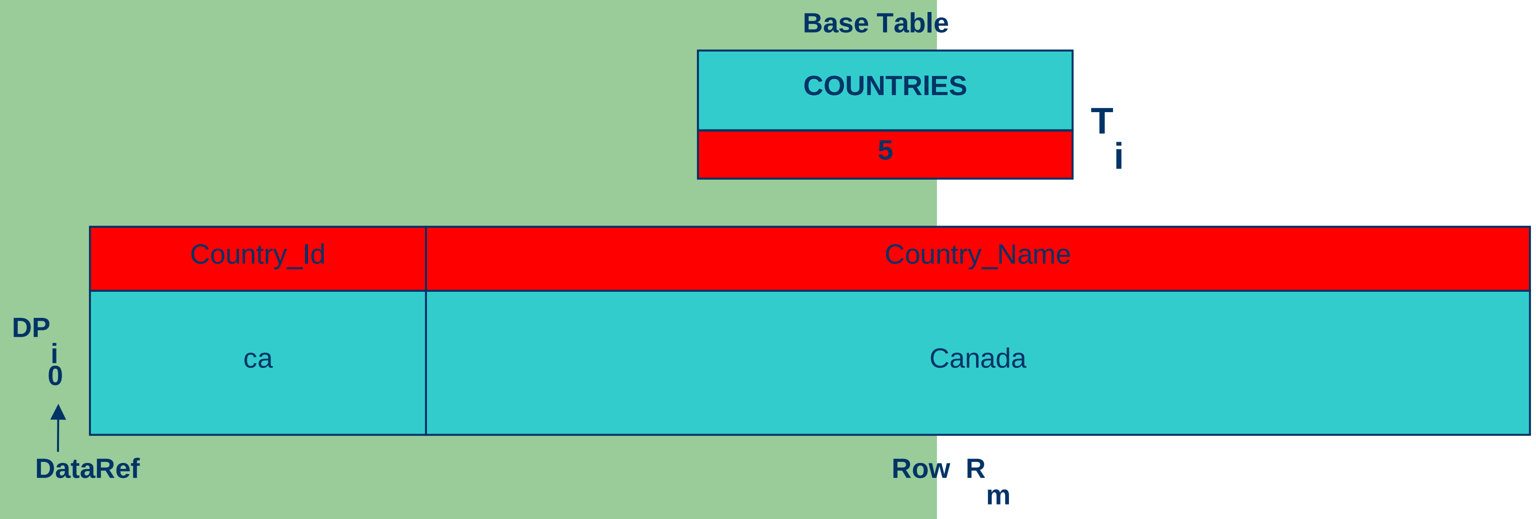
Base Table

COUNTRIES

5

Country_Id	Country_Name
ca	Canada

```
→ insertRoutine(in BaseTable $T_i$, Row $R_m$, DataRef $DP_i$)
          locate the entry of $T_i$ in JoinPathList
          from its adjacentList, locate the definition of Keys & Inherited Keys
          from row $R_m$ get the columns constituting the keys & Inherited Keys
          call AddJoinKey($T_i$,$Keys_i$,$InheritedKeys_i$,$DP_i$)
```



Base Table

COUNTRIES
5

T_i

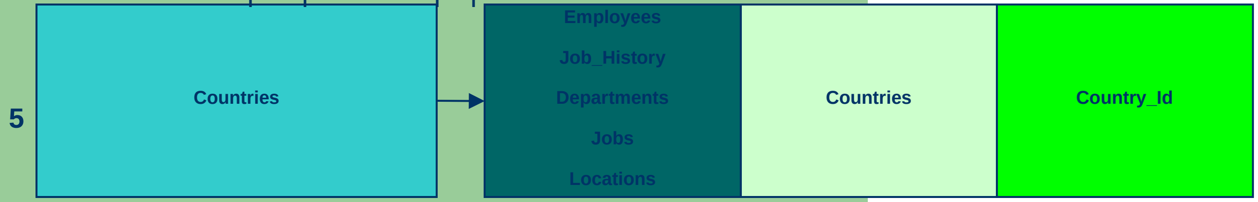
Country_Id	Country_Name
ca	Canada

DP_{i0}

DataRef

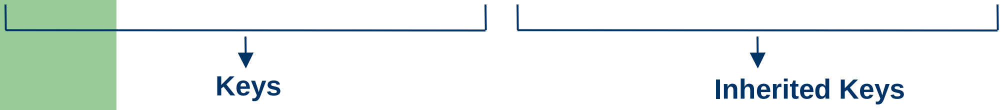
Row R_m

```
insertRoutine(in BaseTable $T_i$, Row $R_m$, DataRef $DP_i$)
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
        call AddJoinKey($T_i$,$Keys_i$,$InheritedKeys_i$,$DP_i$)
```



```
insertRoutine(in BaseTable $T_i$, Row $R_m$, DataRef $DP_i$)
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
        call AddJoinKey($T_i$,$Keys_i$,$InheritedKeys_i$,$DP_i$)
```

5
Countries
Employees
Job_History
Departments
Jobs
Locations
Countries
Country_Id



```
insertRoutine(in BaseTable $T_i$, Row $R_m$, DataRef $DP_i$)
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
        call AddJoinKey($T_i$,$Keys_i$,$InheritedKeys_i$,$DP_i$)
```

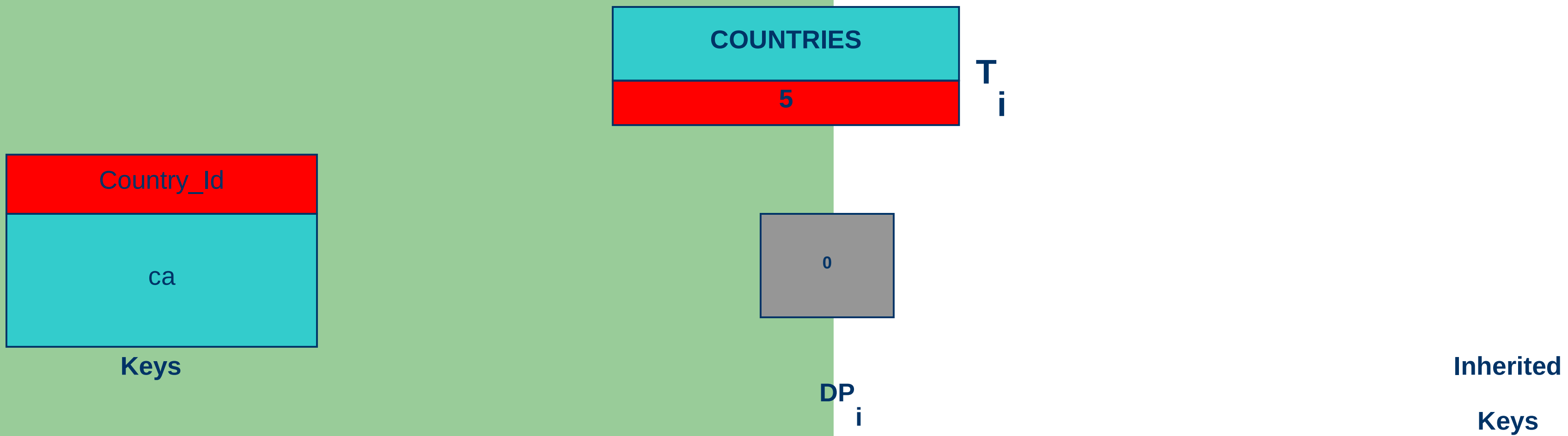
DP_{i0}

Country_Id	Country_Name
ca	Canada

Keys

Row R_m

```
insertRoutine(in BaseTable $T_i$, Row $R_m$, DataRef $DP_i$)
        locate the entry of $T_i$ in JoinPathList
        from its adjacentList, locate the definition of Keys & Inherited Keys
        from row $R_m$ get the columns constituting the keys & Inherited Keys
→       call AddJoinKey($T_i$,$Keys_i$,$InheritedKeys_i$,$DP_i$)
```



COUNTRIES

5

T_i

Country_Id

ca

Keys

0

DP_i

Inherited

Keys

```
→ addJoinKey(in Virtual Table $T_{[i]}$ , $Keys_{[i]}$ , $InheritedKeys_{[i]}$ , [$DP_i$ ])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

COUNTRIES

5

$T_{[i]}$

Country_Id

ca

$Keys_{[i]}$

0

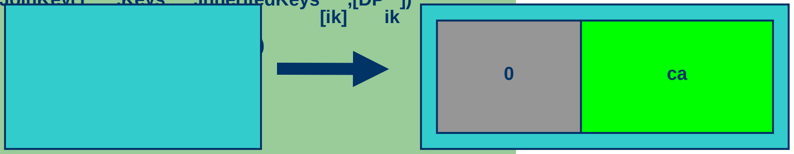
[DP_i]

Inherited

$Keys_{[i]}$

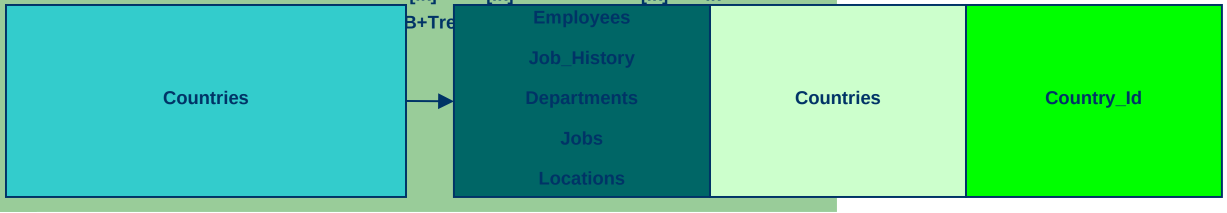
```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey(T ,Keys ,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found =
```

B+Tree(T_5)

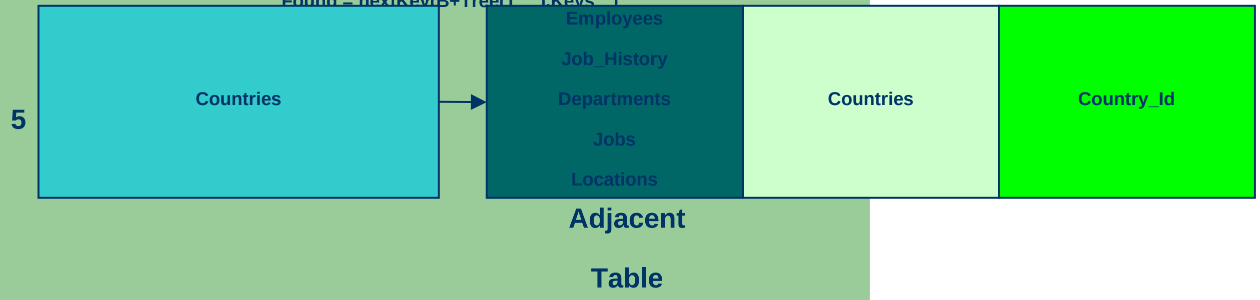


```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            B+Tre
```

5

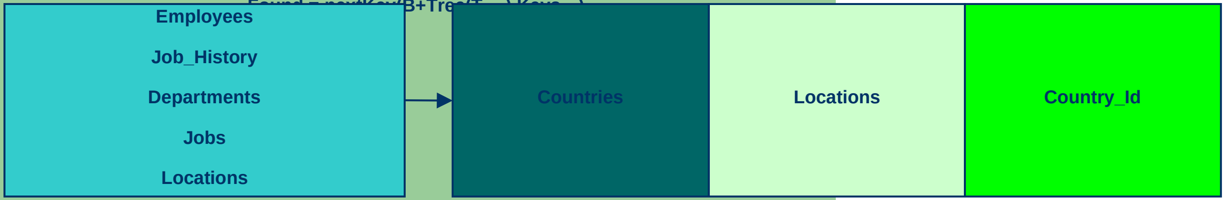


```
addJoinKey(in Virtual Table $T_{[i]}$, Keys$_{[i]}$, InheritedKeys$_{[i]}$, [DP$_i$])
    call addKey(B+Tree($T_{[i]}$),Keys$_{[i]}$,InheritedKeys$_{[i]}$,[DP$_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,Keys$_{[k]}$,InheritedKeys$_{[k]}$,[DP$_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,Keys$_{[ik]}$,InheritedKeys$_{[ik]}$,[DP$_{ik}$])
            Found = nextKey(B+Tree(T ),Keys )
```



```
addJoinKey(in Virtual Table $T_{[i]}$, Keys$_{[i]}$, InheritedKeys$_{[i]}$, [DP$_i$])
    call addKey(B+Tree($T_{[i]}$),Keys$_{[i]}$,InheritedKeys$_{[i]}$,[DP$_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,Keys$_{[k]}$,InheritedKeys$_{[k]}$,[DP$_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,Keys$_{[ik]}$,InheritedKeys$_{[ik]}$,[DP$_{ik}$])
            Found = nextKey(B+Tree(T   ),Keys   )
```

9


Employees

Job_History

Departments

Jobs

Locations

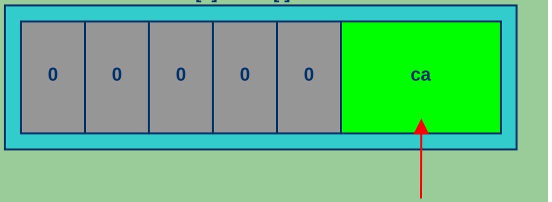
Countries

Locations

Country_Id

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

B+Tree($T_ø$)



Found: TRUE

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

Found: TRUE

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

COUNTRY_ ID
ca

$Keys_{[k]}$

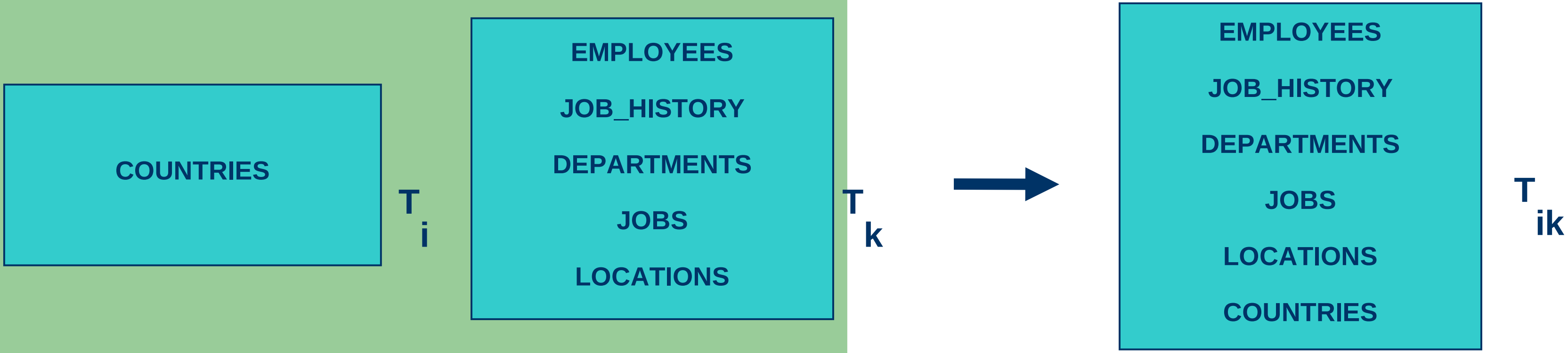
0	0	0	0	0

$[DP_k]$

Inherited

$Keys_{[i]}$

```
addJoinKey(in Virtual Table $T_{[i]}$, Keys$_{[i]}$, InheritedKeys$_{[i]}$, [DP$_i$])
    call addKey(B+Tree($T_{[i]}$),Keys$_{[i]}$,InheritedKeys$_{[i]}$,[DP$_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,Keys$_{[k]}$,InheritedKeys$_{[k]}$,[DP$_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,Keys$_{[ik]}$,InheritedKeys$_{[ik]}$,[DP$_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
```



COUNTRIES

T_i

EMPLOYEES

JOB_HISTORY

DEPARTMENTS

JOBS

LOCATIONS

T_k

EMPLOYEES

JOB_HISTORY

DEPARTMENTS

JOBS

LOCATIONS

COUNTRIES

T_{ik}

10

Employees Job_History

Departments Jobs

Locations Countries

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

10

Employees Job_History

Departments Jobs

Locations Countries

Keys

Inherited Keys

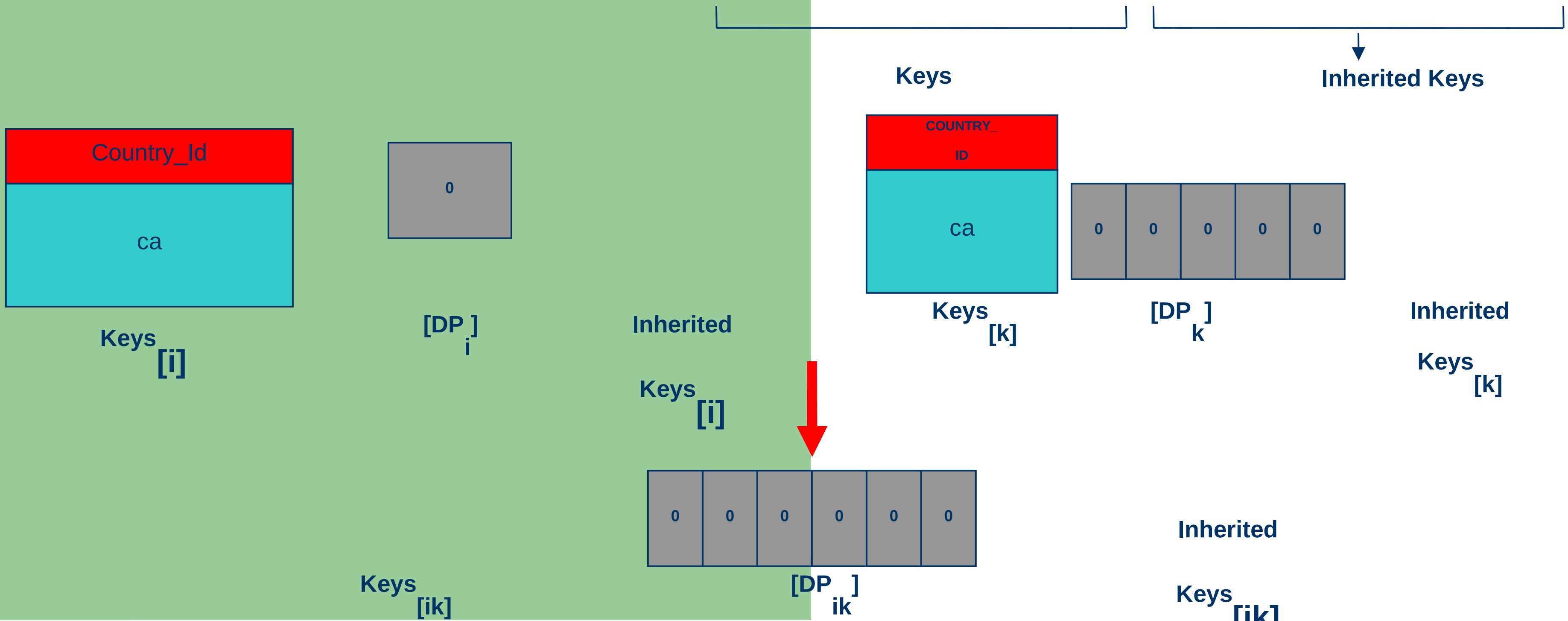
```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call AddJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

10

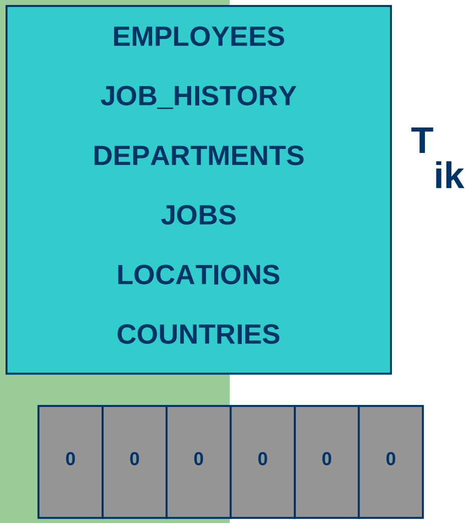
Employees Job_History

Departments Jobs

Locations Countries



```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_{i}]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_{i}]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_{k}]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```



$Keys_{[ik]}$

$[DP_{ik}]$

Inherited

$Keys_{[ik]}$

```
→ addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

EMPLOYEES

JOB_HISTORY

DEPARTMENTS

JOBS

LOCATIONS

COUNTRIES

$T_{[i]}$

0	0	0	0	0	0

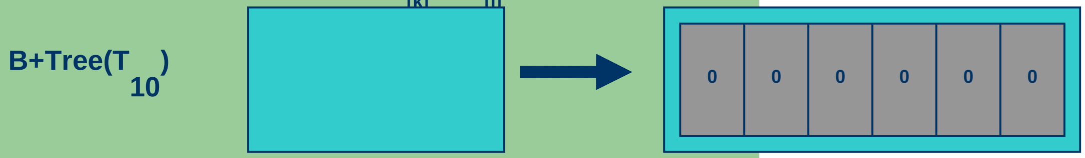
$Keys_{[i]}$

$[DP_i]$

Inherited

$Keys_{[i]}$

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
→       call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
        locate the entry of $T_{[i]}$ in JoinPathList
        from its adjacentList, locate the table $T_{[k]}$ adjacent to it
            locate the entry of $T_{[k]}$ in JoinPathList
            Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
            while found do
                returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
                locate the entry of $T_{[ik]}$ in JoinPathList
                from its adjacentList, locate the definition of Keys & Inherited Keys
                from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
                call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
                Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```



```
addJoinKey(in Virtual Table $T_{[i]}$ , $Keys_{[i]}$ , $InheritedKeys_{[i]}$ , $[DP_{i}]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_{i}]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_{k}]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from $keys_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

10

Employees Job_History

Departments Jobs

Locations Countries

```
addJoinKey(in Virtual Table $T_{[i]}$ , Keys$_{[i]}$ , InheritedKeys$_{[i]}$ , [DP$_i$])
    call addKey(B+Tree($T_{[i]}$),Keys$_{[i]}$,InheritedKeys$_{[i]}$,[DP$_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,Keys$_{[k]}$,InheritedKeys$_{[k]}$,[DP$_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,Keys$_{[ik]}$,InheritedKeys$_{[ik]}$,[DP$_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),Keys$_{[i]}$)
```

10

Employees Job_History

Departments Jobs

Locations Countries

Adjacent

Table

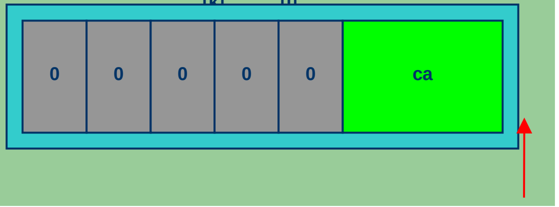
```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_{i}$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_{i}$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
→       locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_{k}$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)          Found: FALSE
```

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_{i}$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_{i}$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
→       Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_{k}$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)          Found: FALSE
```

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
→           call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, $[DP_i]$)
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,$[DP_i]$)
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,$[DP_k]$)
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, InheritedKeys$_{[i]}$, Keys$_{[k]}$ & InheritedKeys$_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
→           call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,$[DP_{ik}]$)
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

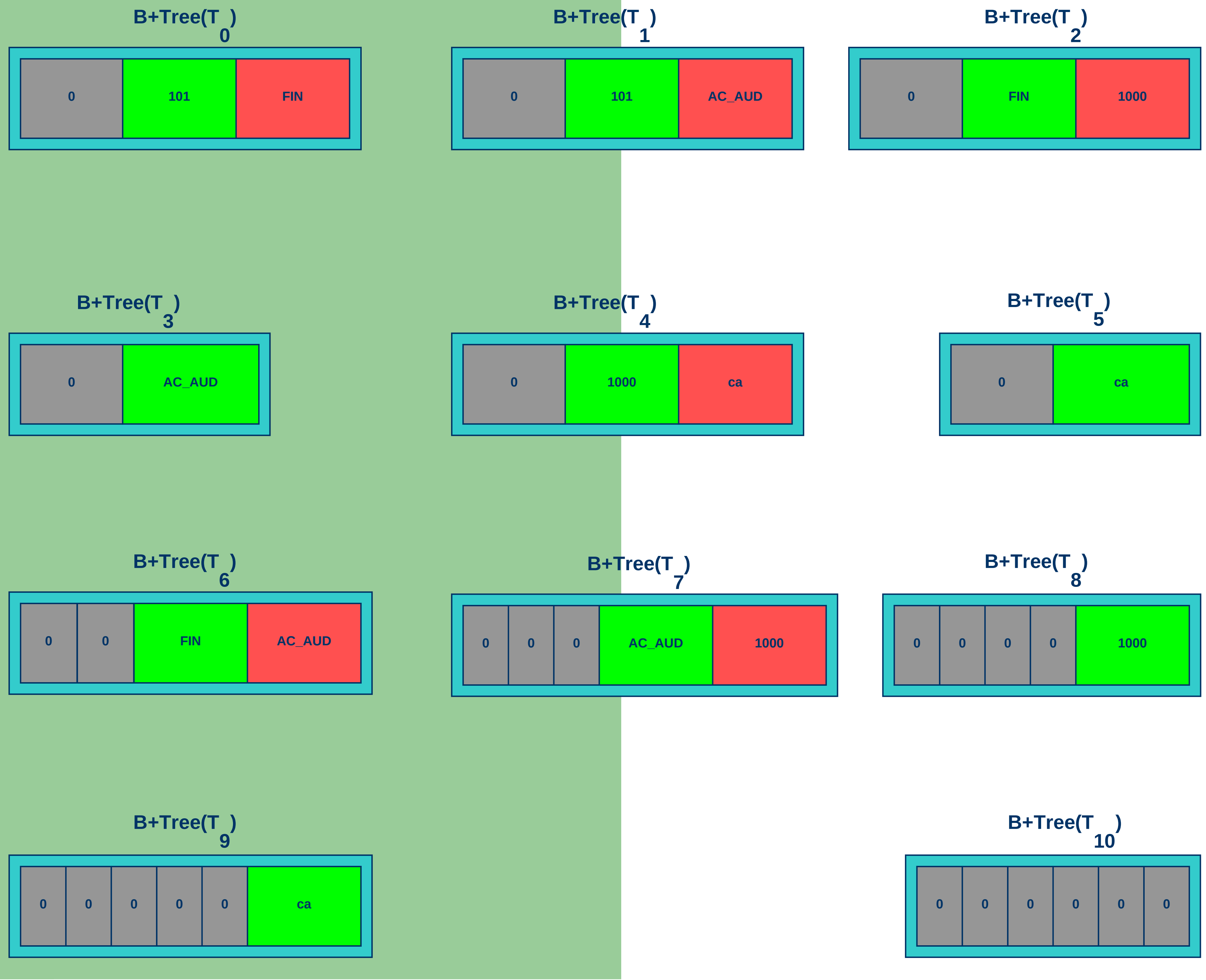
B+Tree(T_9)



Found: FALSE

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
→       while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
            call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```

```
addJoinKey(in Virtual Table $T_{[i]}$, $Keys_{[i]}$, $InheritedKeys_{[i]}$, [$DP_i$])
    call addKey(B+Tree($T_{[i]}$),$Keys_{[i]}$,$InheritedKeys_{[i]}$,[$DP_i$])
    locate the entry of $T_{[i]}$ in JoinPathList
    from its adjacentList, locate the table $T_{[k]}$ adjacent to it
        locate the entry of $T_{[k]}$ in JoinPathList
        Found = findKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
        while found do
            returnKeys(B+Tree($T_{[k]}$,$Keys_{[k]}$,$InheritedKeys_{[k]}$,[$DP_k$])
            locate the entry of $T_{[ik]}$ in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys$_{[i]}$, $InheritedKeys_{[i]}$, $Keys_{[k]}$ & $InheritedKeys_{[k]}$ get the keys & Inherited keys of $T_{[ik]}$
→           call addJoinKey($T_{[ik]}$,$Keys_{[ik]}$,$InheritedKeys_{[ik]}$,[$DP_{ik}$])
            Found = nextKey(B+Tree($T_{[k]}$),$Keys_{[i]}$)
```



B+Tree(T_0)

0	101	FIN

B+Tree(T_1)

0	101	AC_AUD

B+Tree(T_2)

0	FIN	1000

B+Tree(T_3)

0	AC_AUD

B+Tree(T_4)

0	1000	ca

B+Tree(T_5)

0	ca

B+Tree(T_6)

0	0	FIN	AC_AUD

B+Tree(T_7)

0	0	0	AC_AUD	1000

B+Tree(T_8)

0	0	0	0	1000

B+Tree(T_9)

0	0	0	0	0	ca

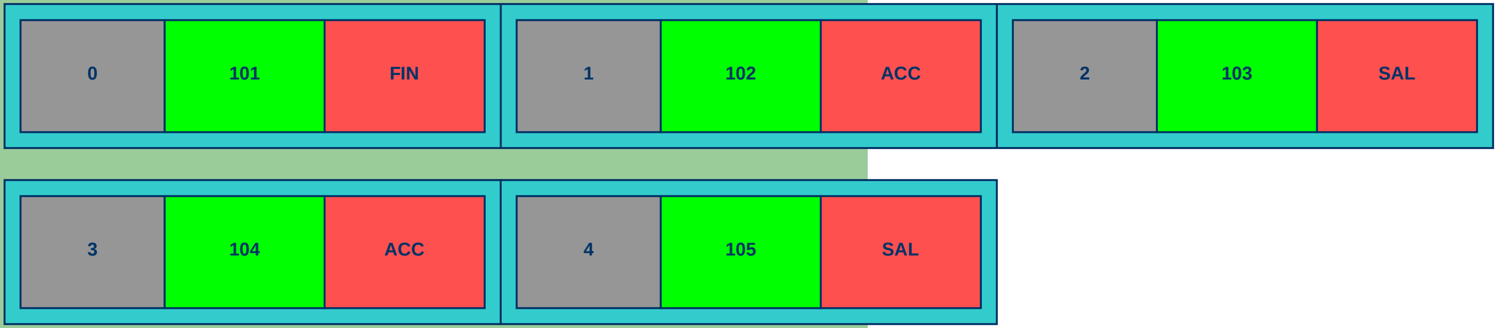
B+Tree(T_{10})

0	0	0	0	0	0

As we can notice from the last index we have an element with 6 data pointers respectively pointing to the 6 base tables forming the virtual table T_{10}, with all values equal to the first row on each table, those rows are in join together.

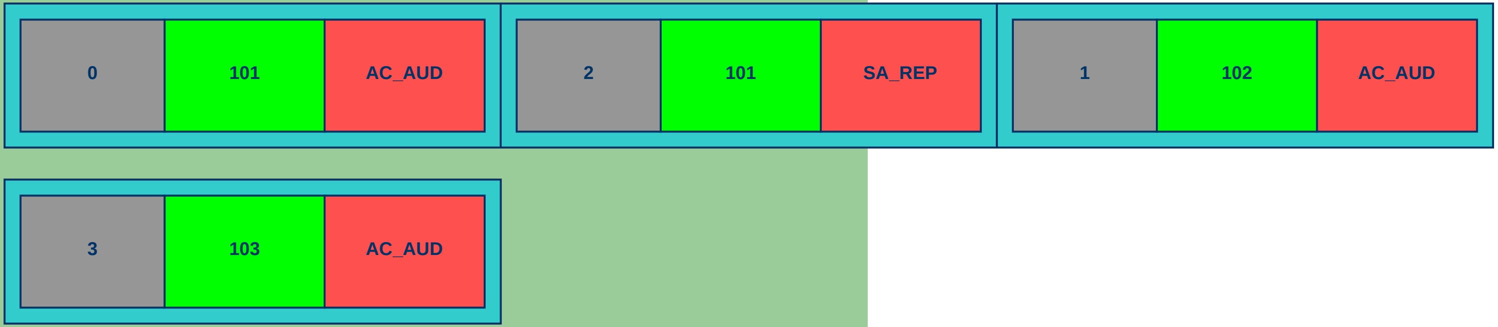
Inserting all the remaining rows from the tables we obtain the following indexes where the last index shows the join between the rows from the tables.

B+Tree(T_0)



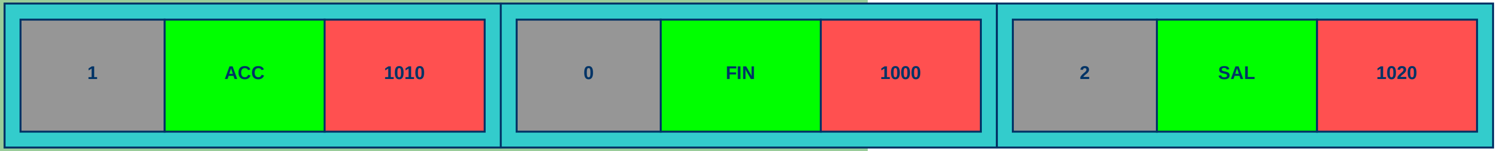
0	101	FIN
1	102	ACC
2	103	SAL
3	104	ACC
4	105	SAL

B+Tree(T_1)



0	101	AC_AUD
2	101	SA_REP
1	102	AC_AUD
3	103	AC_AUD

B+Tree(T_2)



1	ACC	1010
0	FIN	1000
2	SAL	1020

B+Tree(T_3)

0	AC_AUD
1	AC_MGR
2	FI_MGR
3	SA_MGR
4	SA_REP

B+Tree(T_4)

0	1000	ca
1	1010	me
2	1020	fr

B+Tree(T_5)

0	ca
1	fr
2	me

B+Tree(T_6)

1	1	ACC	AC_AUD
0	0	FIN	AC_AUD
0	2	FIN	SA_REP
2	3	SAL	AC_AUD

B+Tree(T_7)

0	0	0	AC_AUD	1000
2	3	2	AC_AUD	1020
1	1	1	AC_AUD	1010
0	2	0	SA_REP	1000

B+Tree(T_8)

0	0	0	0	1000
0	2	0	4	1000
1	1	1	0	1010
2	3	2	0	1020

B+Tree(T_9)

0	0	0	0	0	ca
0	2	0	4	0	ca
2	3	2	0	2	fr
1	1	1	0	1	me

B+Tree(T_{10})

0	0	0	0	0	0
0	2	0	4	0	0
1	1	1	0	1	2
2	3	2	0	2	1

Delete routine

When a row R_m from table T_i get deleted 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 DelJoinKey (T_i, $Keys_i$, $InheritedKeys_i$, DP_i) where DP_i is the row id of row R_m

  Notice that $Keys_i$, $InheritedKeys_i$ and DP_i are relative to the row R_m from table T_i

DelJoinKey (T$_{[i]}$, Keys$_{[i]}$, InheritedKeys$_{[i]}$,[DP$_{i}$])

- Call delKey (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[$_{ik]}$

    DelJoinKey (T$_{[ik]}$, Keys$_{[ik]}$, InheritedKeys$_{[ik]}$, [DP$_{ik}$])

    NextKey(B^{+}Tree(T$_{[k]}$), Keys$_{[i]}$)

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 arays 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:

$T_{[buf]}$ + = T_i

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

where T_i 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 mechanism to drive us in the insert for the join.

Suppose that the order of the B^{+}Trees is m and the number of elements for every B^{+}Tree with i as index from the (2*n-1) B^{+}Trees is $p_i * l_i$ where 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:

$Ord(n * \log_m(l_i * p_i))$

Complexity of the algorithm for the other operations.

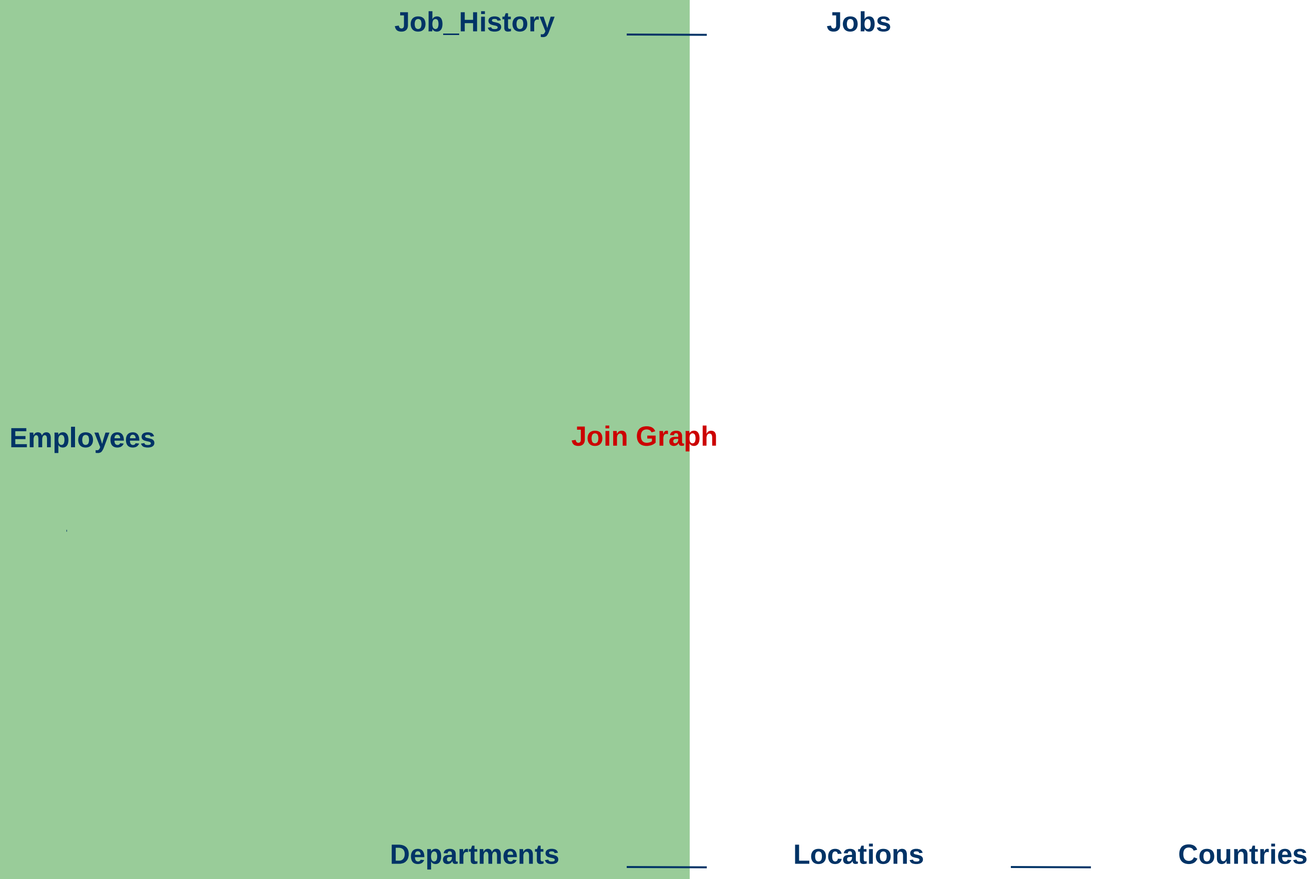
The only B$^+$Tree of our interest for the scan is the one with the latest index that have the join of the tables inside it.

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.

Proof of correctness.

To prove the correctness of the algorithm let see how does the algorithm work for the example above and later generalize it.

The Join Graph could be calculated easily even manually when we know which Tables are in direct join with others.



Let define a path in the Join Graph, the same path generated by the algorithm:

generateJoinPathList

Numbers on the tables to indicate the path direction to follow, the same generated by the algorithm.

1

Job_History ______ **Jobs**

3

0 **Employees** Start Table

A mechanism to drive B+Trees to do join internally for the following tables:

End Table

Departments ______ **Locations** ______ **Countries**

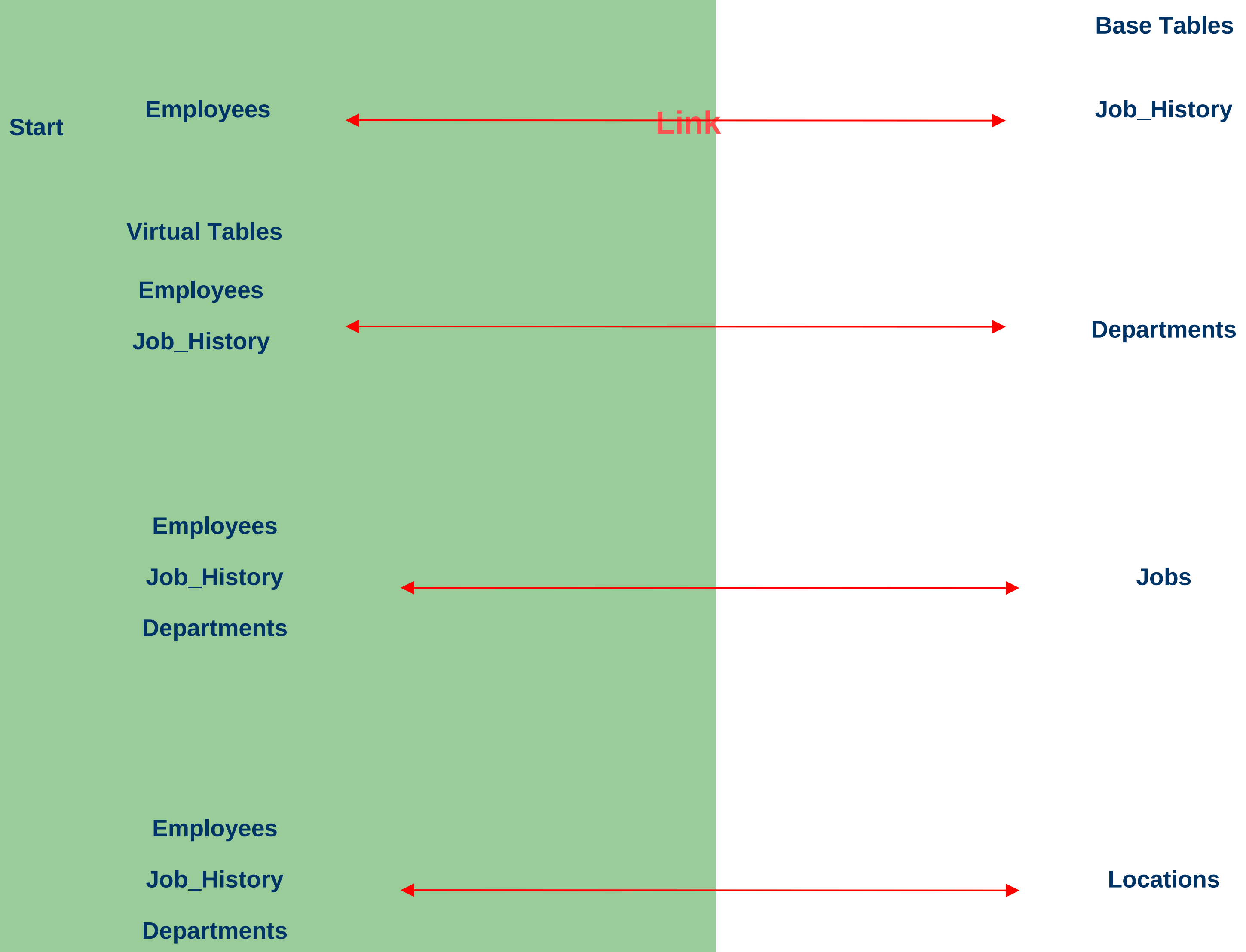
2 **4** **5**

Notice that in the path if we reach one table it is not necessary to continue from it.

This is very important because this makes the tables free from any order,

independent selection of the start and end tables.

By grouping comes out:`



Base Tables

Start

Employees

Link

Job_History

Virtual Tables

Employees

Job_History

Departments

Employees

Job_History

Departments

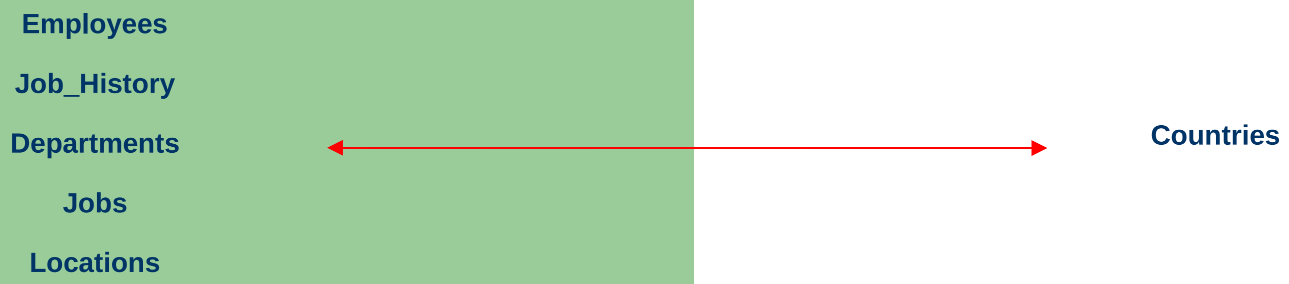
Jobs

Employees

Job_History

Departments

Locations



Employees

Job_History

Departments

Jobs

Locations

Countries

Employees

Job_History

Departments

Jobs

Locations

Countries

As we can see for every Virtual Table there is a Base Table in which there is a direct join between them and vice versa, in fact they belongs to the same Path in the Join Graph.

The idea consists in that every Virtual Table is constituted from Base Tables that are in join together. In fact the Base Tables constituting the Virtual Table appears by adding one at time that is in direct join with the one of the previous tables.

Now the join between tables should be calculated and stored to be found. For this reason B+Tree is declared for every Virtual Table that can hold references for rows from Base Tables constituting the Virtual Table in mode that concatenating them together bring out a joined Row.

Rows are inserted into a database as one row from a base table at a time, the system look for the link table, and check the B+Tree to see if there is any row that satisfy the join with the newly inserted; if this is the case combine each row satisfying with newly inserted by their references, and insert the combined row in the virtual table that has as base tables the base tables of the 2 previously tables.

So at any time when a row get inserted, the link table may eventually have the rows that satisfy the join with it, so they are combined and the process continue to the last virtual table or if they didn’t get inserted yet in the virtual table, later when they get inserted they are confronted with the one inserted yet and the process continue on the same way.

The last table will contain all base tables in join together.

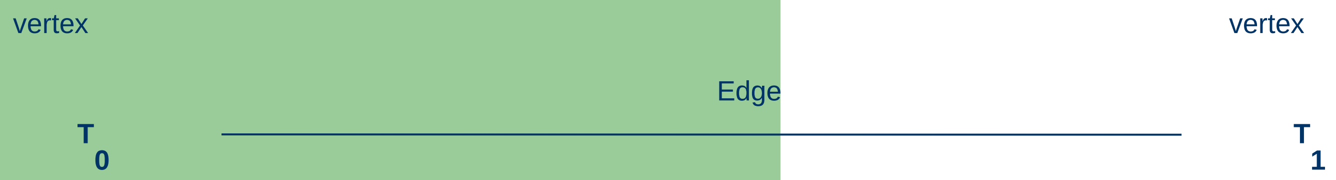
Proof of correctness.

Notice that what we show before is independent from the number of tables, so that the same reasoning apply to any number of tables and the proof of correctness could be easily proved by induction.

Let prove the correctness by induction.

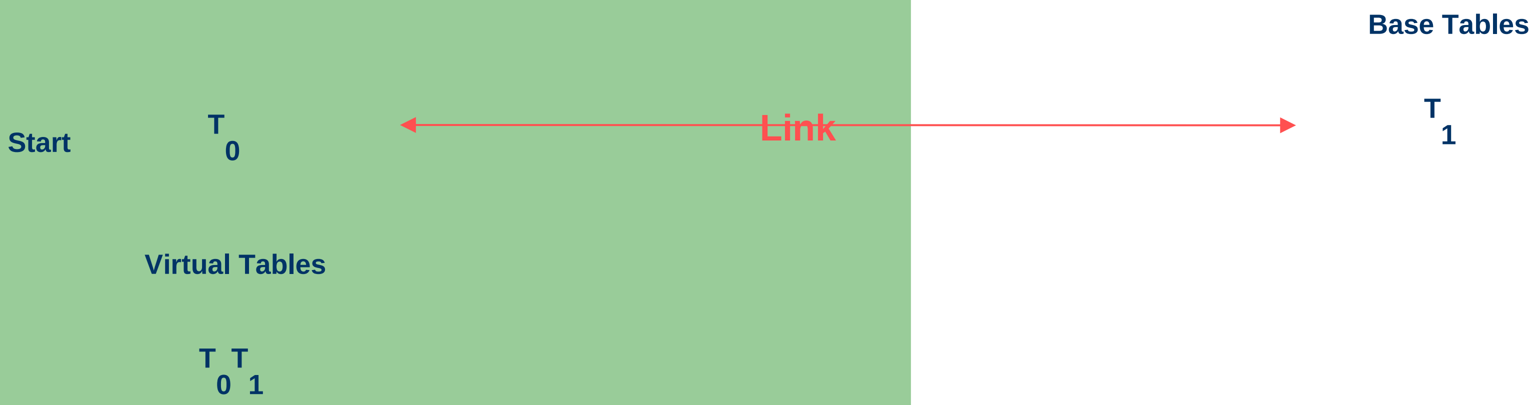
To do it, let see the correctness for 2 tables T_0 and T_1 in join together.

The join graph should be the following:

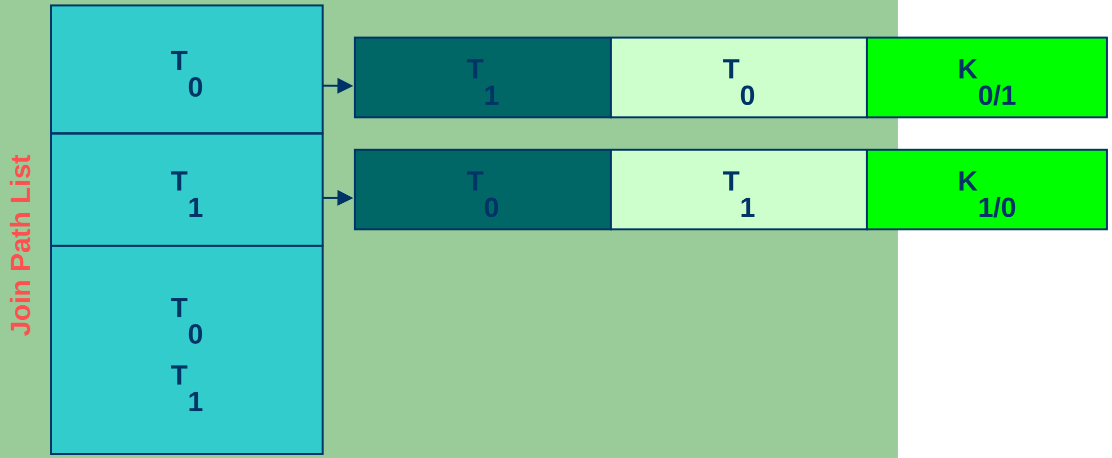


There are just 2 paths between the 2 tables: or from T_0 going toward T_1 or vice versa, let consider the former, the second case is symmetric and after all T_0 and T_1 are of arbitrarily choice.

By grouping comes out:



So, the JoinPathList should be the following:



Join Path List

T_0 → T_1 | T_0 | $K_{0/1}$

T_1 → T_0 | T_1 | $K_{1/0}$

T_0 T_1

If any key has been defined on the last virtual table and doesn’t exist as a key on the base tables then should be propagated as inherited key in the appropriate base table; but for the prove of correctness in case of 2 tables, it is not important.

To prove the correctness of the algorithm, we have to prove that the last virtual table contain data references to all the rows that combined form the join between the 2 base tables and only those in other sense it is equivalent to the result of the join between the 2 tables.

Let prove that the last virtual table contain data references to all the rows that combined form the join between the 2 base tables:

Suppose by absurd that there is a row $R_{m/0}$ from table T_0 and a row $R_{n/1}$ from table T_1 that are in join together and they don’t have references in the last virtual table.

If the 2 rows are in join together so their respective keys satisfy the join condition.

Suppose that $R_{m/0}$ comes first, so key($R_{m/0}$) is inserted in the B+Tree(T_0).

When $R_{n/1}$ get inserted later, the insert algorithm look in JoinPathList the adjacent table to T_1, it finds that T_0 is such table and look in B+Tree(T_0) all the keys that satisfy the join condition with the value of key($R_{n/1}$). It will get key($R_{m/0}$) because such key satisfy the join condition, it will combine the data references of the 2 Rows and insert in the virtual table such couple of references.

This is in contradiction on what we assume initially.

The case that $R_{n/1}$ comes first is symmetric.

Let prove that the only couples of data references in the last virtual table are those that combined make the join between the 2 base tables:

Suppose by absurd that there is a couple of references DP_0 and DP_1 that are data pointers to rows from table T_0 and table T_1 respectively in the virtual table and that the combined row doesn’t belong to a join between the 2 base tables.

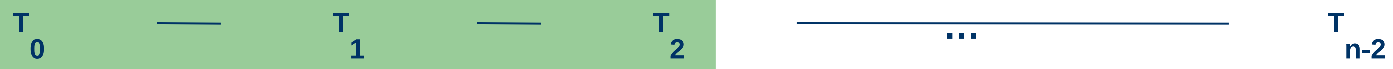
If such a couple of data pointers exist, it comes out because there is 2 keys belonging to the rows pointed by the data pointers and such keys satisfy the join condition, this is in contradiction on what we assume initially.

The initial case when there is only 2 tables in join is proved to be correct. Now let suppose that the correctness is true for n-1 tables and let prove it when the number of tables is n tables.

The easiest way to prove it for n tables is to expand the virtual table with (n-1) base tables. This virtual table has a B+Tree that is constituted from set of elements in which every element has a common key value with the nth table and (n-1) data pointers that points to the (n-1) base tables. By expanding in the sense that from every element taking the (n-1) rows from the (n-1) tables and considering them as one row in a virtual table, we can look at the virtual table as a table populated with such rows.

Let see first the Join Graph for the (n-1) tables and how they went in group and later what happens when we consider the nth table.

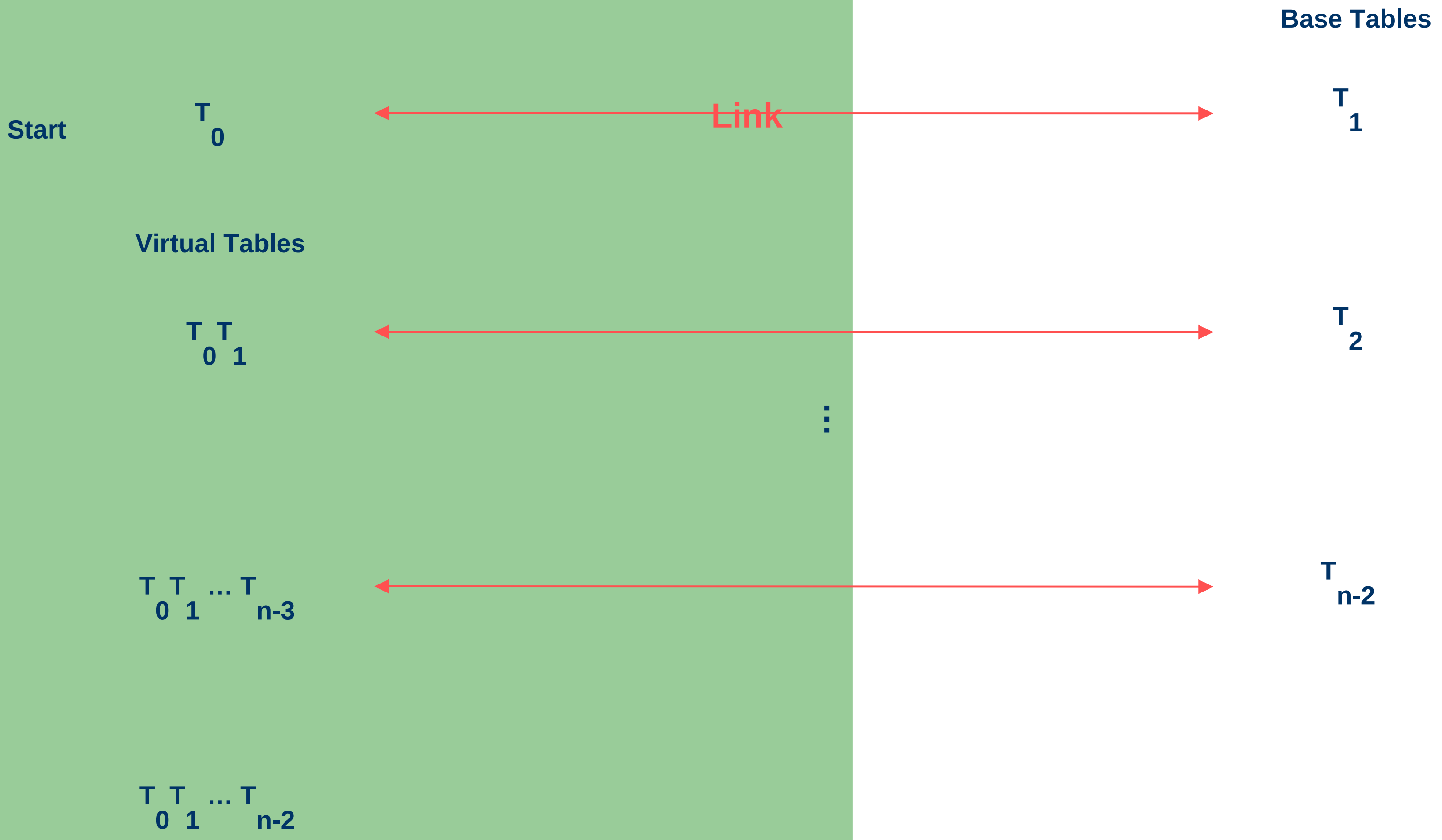
The join graph for the (n-1) tables should be the following:



T_0 — T_1 — T_2 … T_{n-2}

Suppose that the choice of T_0 … T_{n-2} are in the way that the path start from T_0, continue by T_1 … T_{n-3} till the end to arrive at T_{n-2}.

By grouping comes out:



Base Tables

Start T_0 ⟷ **Link** ⟷ T_1

Virtual Tables

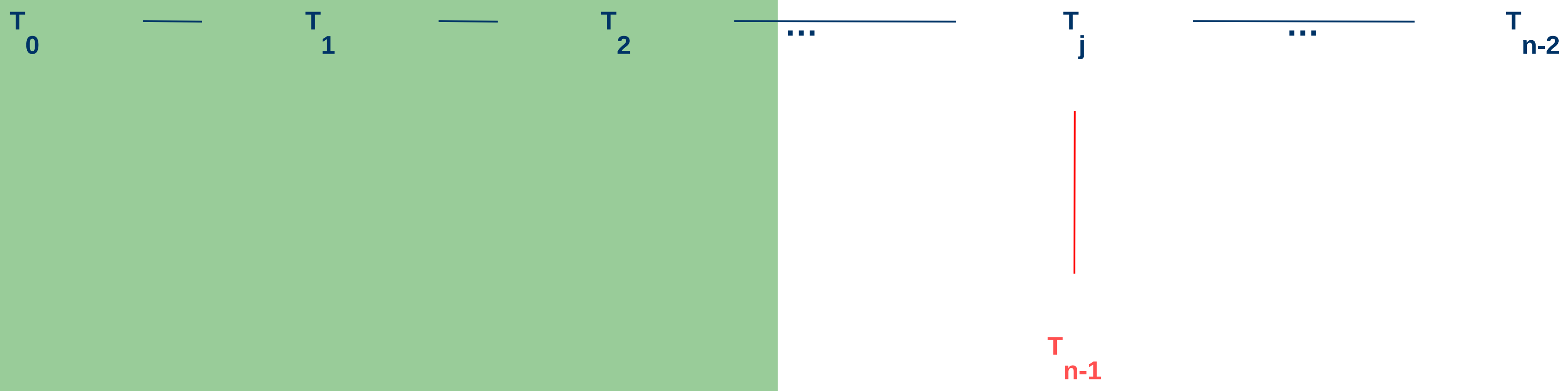
T_0T_1 ⟷ T_2

⋮

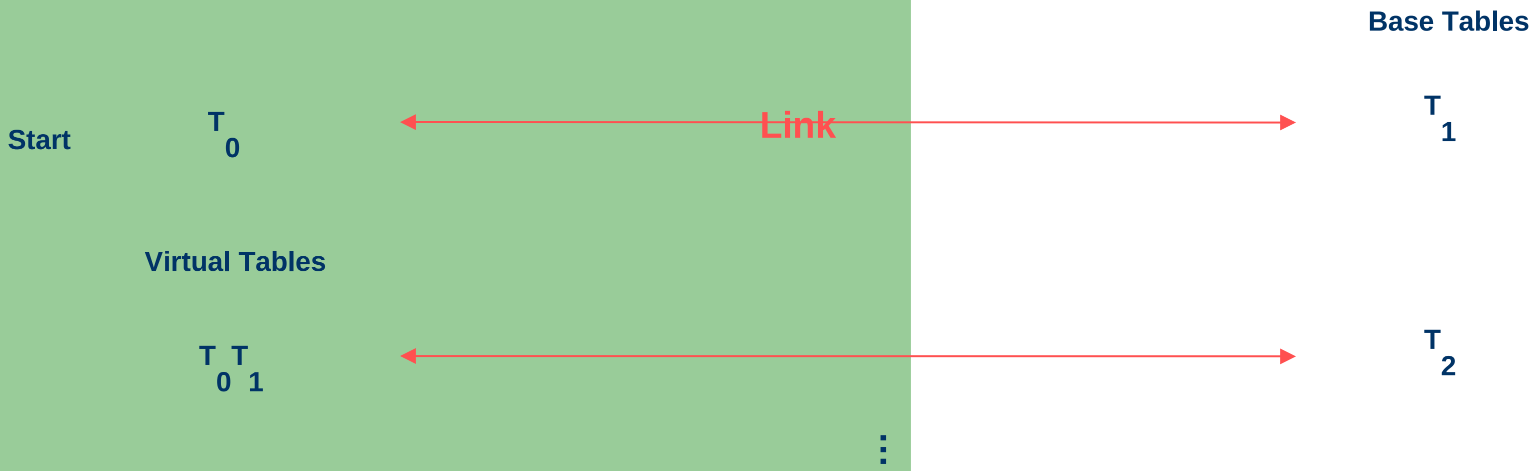
$T_0T_1 \dots T_{n-3}$ ⟷ T_{n-2}

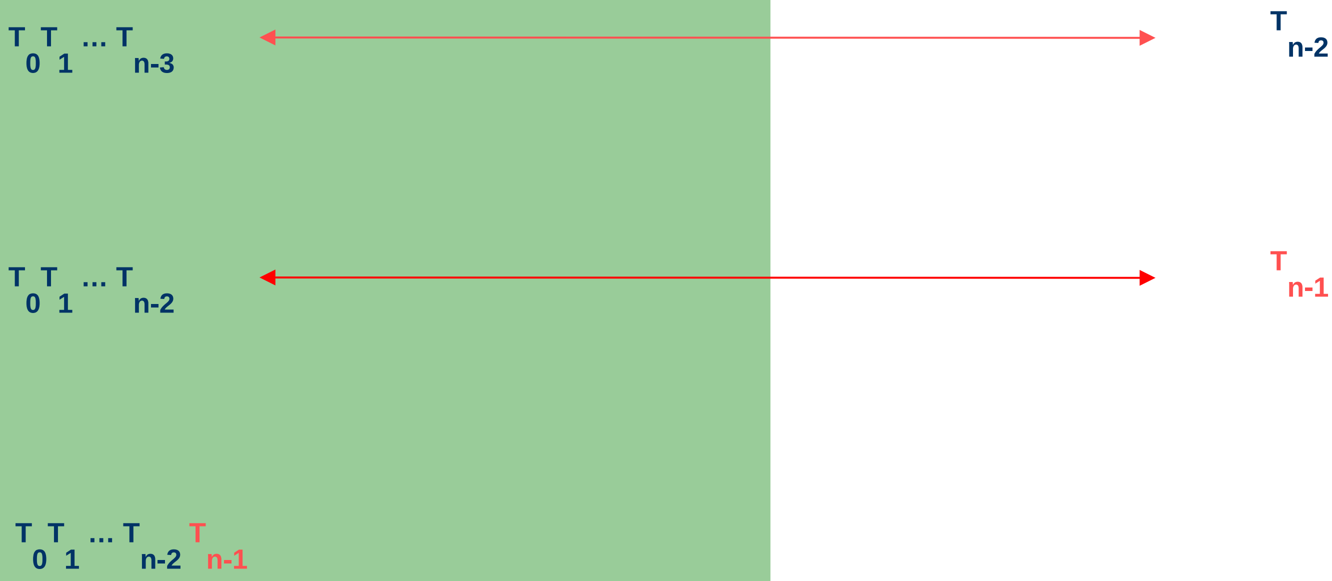
$T_0T_1 \dots T_{n-2}$

If there is one more table, the join graph would be the following:



The path should looks like this:





$T_0\ T_1 \dots T_{n-3}$ ↔ T_{n-2}

$T_0\ T_1 \dots T_{n-2}$ ↔ T_{n-1}

$T_0\ T_1 \dots T_{n-2}\ T_{n-1}$

So if we expand the virtual table $T_0 \dots T_{n-2}$ so the link would be just between it and the table T_{n-1}, where he common key should be from table T_j.

So we have the following situation:

Table $T_{0 \dots n-2}$ an expanded table from the virtual table $T_0 \dots T_{n-2}$ and by induction it is the same table obtained by the join of the (n-1) base tables.

Table T_{n-1}

So if we name $T_{0 \dots n-2}$ as T_0 and T_{n-1} as T_1, we return to the case already proved of 2 tables where the common key in $T_{0 \dots n-2}$ is calculated from the combined joined row in the place of the row pointed by DP_j.

The only thing remain to prove is the propagation of the key from $T_0 \ldots T_{n-2}$ to T_j and the eventual keys from $T_0 \ldots T_{n-1}$ to some base tables in the base tables $T_0 \ldots T_{n-1}$ but this is guaranteed in the third phase of the algorithm generateJoinPathList because it goes backward and insert eventual inherited keys.

Self Join

If the table is in join with itself, consider the table twice, every one with the necessary index.

Let see an example of self join.

Suppose that we add a column named SUPERVISOR_ID in the table EMPLOYEES, it has the id of the supervisor for a given employee.

Suppose that we have the following query:

```
SELECT A.EMPLOYEE_NAME, B.EMPLOYEE_NAME
FROM EMPLOYEES AS A, EMPLOYEES AS B
WHERE A.EMPLOYEE_ID = B.SUPERVISOR_ID
```

The table EMPLOYEES with the new column SUPERVISOR_ID is shown in slide 575.

```
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/Employee_Id	Employees/Supervisor_Id
0	1

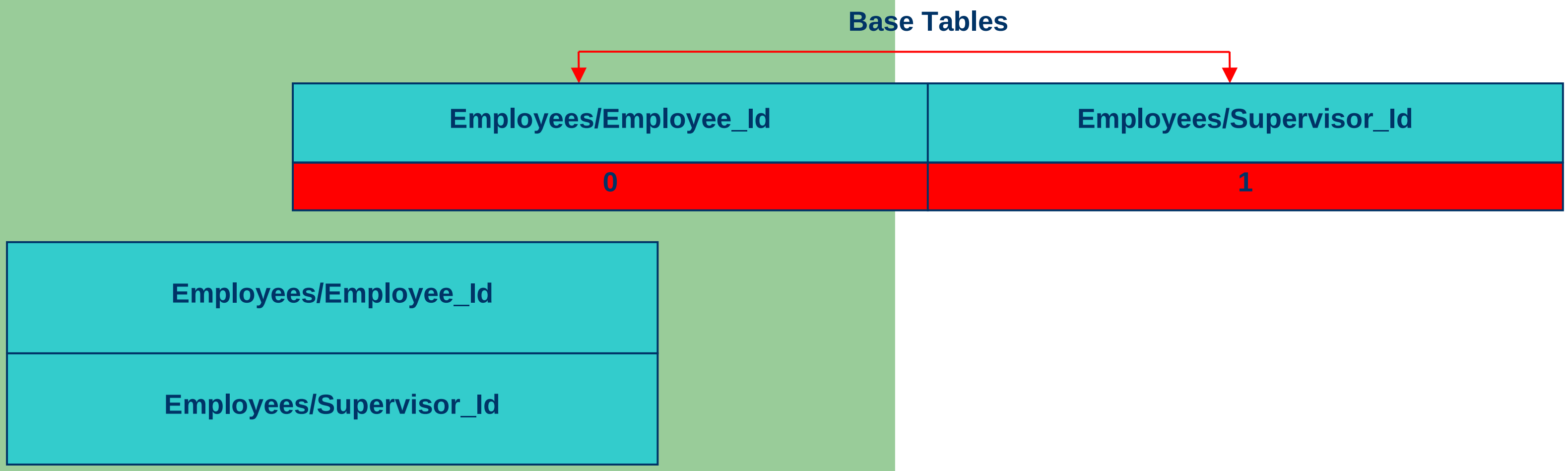
```
generateJoinGraph (in BaseTables; out JoinGraph)
→ insert the base tables as vertexes of the graph
for every direct join between 2 tables of the form $T_i$ and $T_k$ where $T_i$ is the table of order i and $T_k$ is the table of order k as defined by the DBA do
    AdjacentList[$T_i$] += $T_k$ follow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

Base Tables

Employees/Employee_Id	Employees/Supervisor_Id
0	1

Employees/Employee_Id
Employees/Supervisor_Id

```
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
```

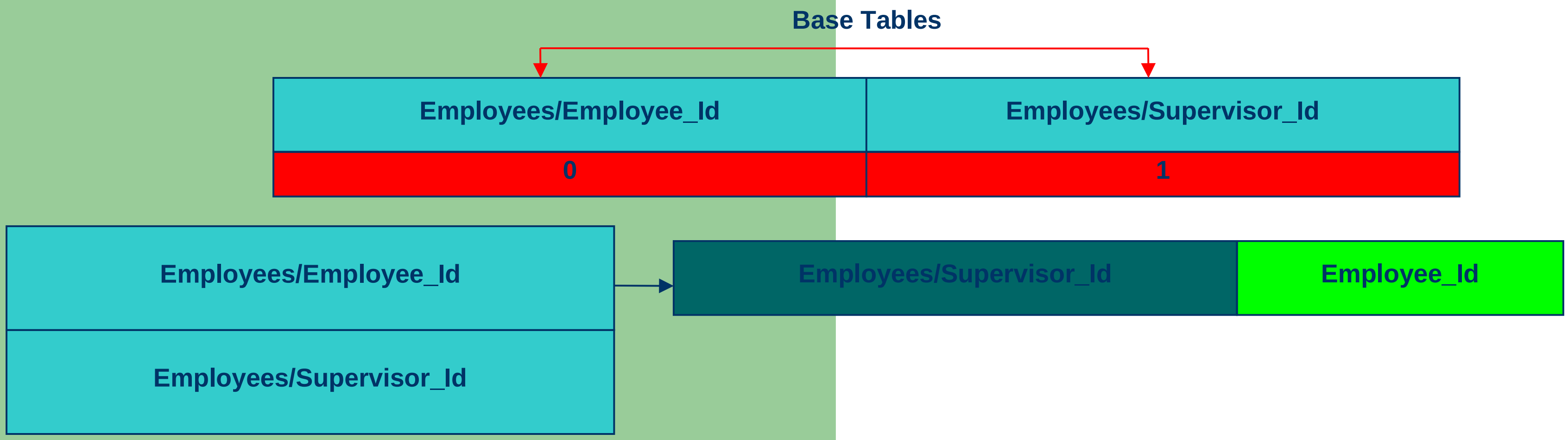


Base Tables

Employees/Employee_Id	Employees/Supervisor_Id
0	1

Employees/Employee_Id
Employees/Supervisor_Id

```
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/Employee_Id	Employees/Supervisor_Id
0	1

Employees/Employee_Id

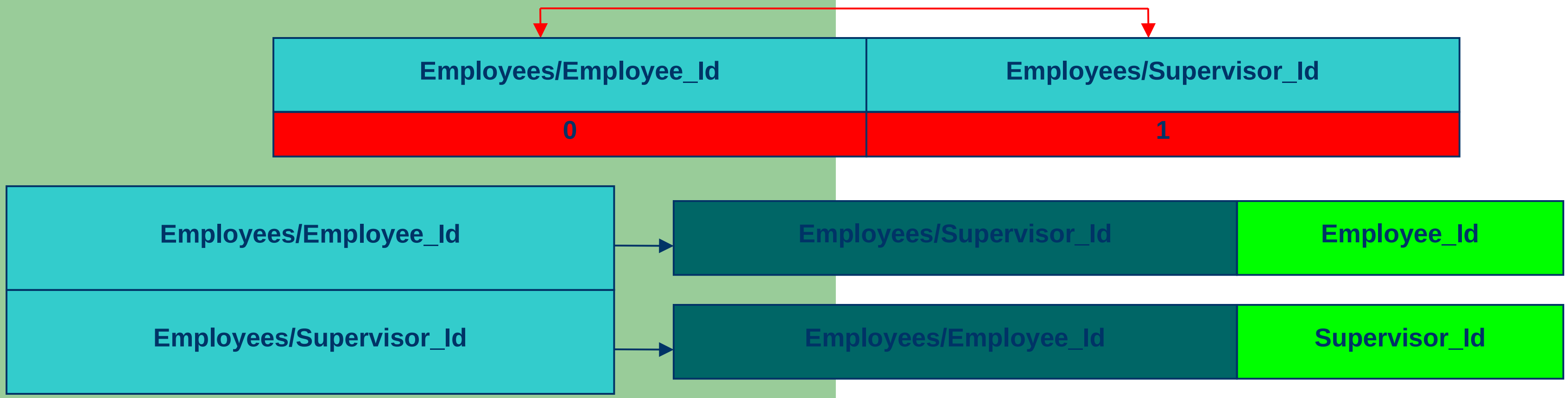
Employees/Supervisor_Id

Employees/Supervisor_Id

Employee_Id

```
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

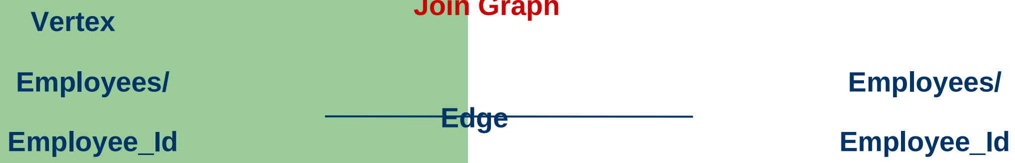


Employees/Employee_Id	Employees/Supervisor_Id
0	1

Employees/Employee_Id → Employees/Supervisor_Id | Employee_Id

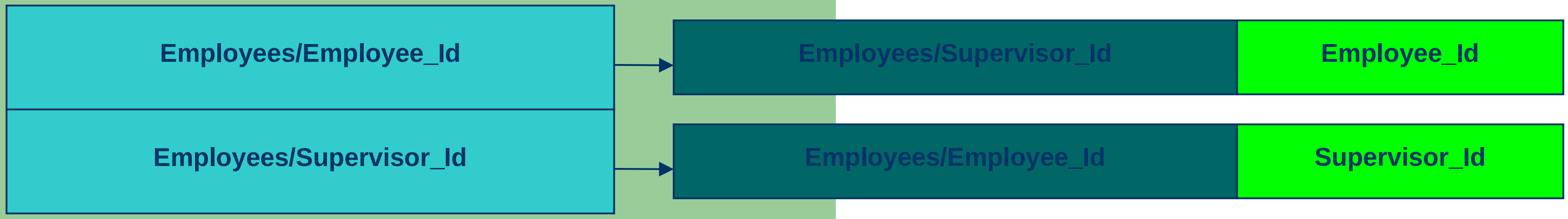
Employees/Supervisor_Id → Employees/Employee_Id | Supervisor_Id

Join Graph



Linked List representation of the Join Graph

Adjacent List



Node

Tables

Adjacent

Tables

Common keys between

Node & Adjacent Tables

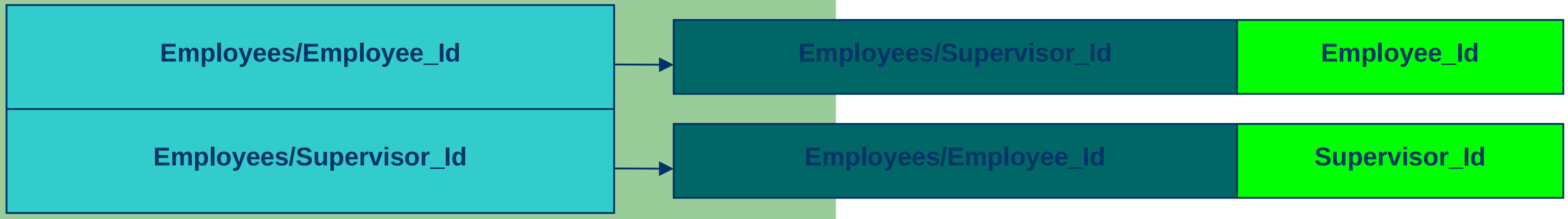
belongs to Node Tables

```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0$ …$T_m$ 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/Employee_Id	Employees/Supervisor_Id

Join Graph



Employees/Employee_Id

Employees/Supervisor_Id

Employees/Supervisor_Id

Employee_Id

Employees/Employee_Id

Supervisor_Id

```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
→ let $T_0$ …$T_m$ 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

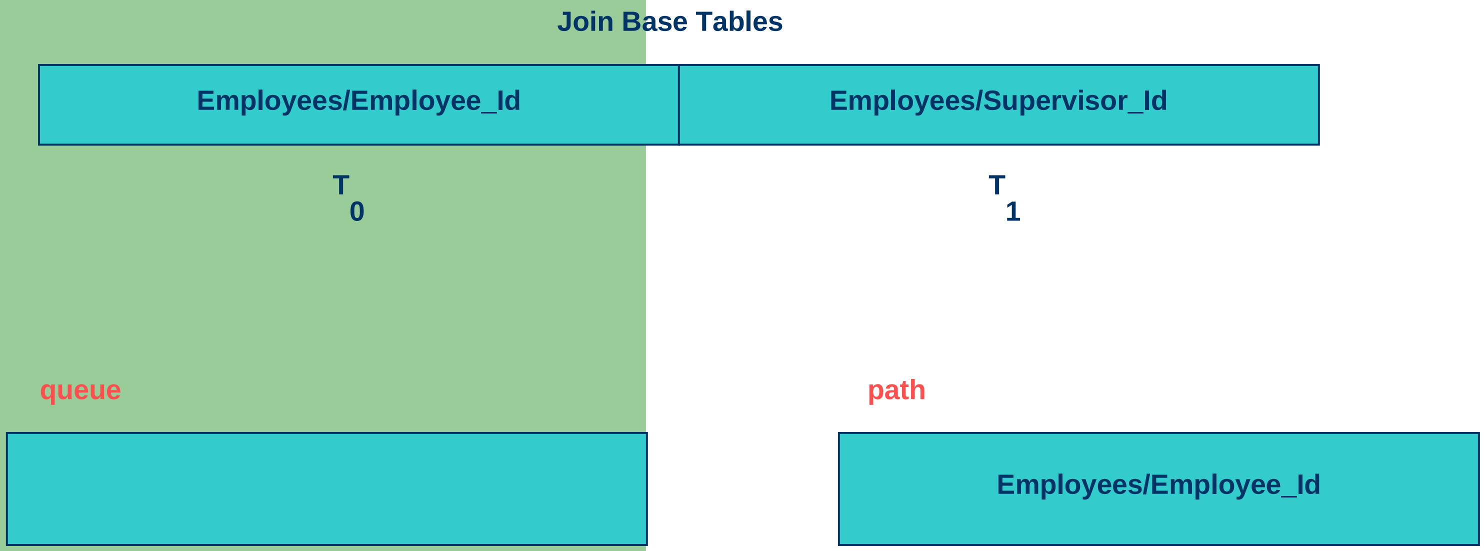
Employees/Employee_Id	Employees/Supervisor_Id
T_0	T_1

```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0$ …$T_m$ 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 $T_0$ …$T_m$ 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/Employee_Id	Employees/Supervisor_Id
T_0	T_1

queue

path

Employees/Employee_Id

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

let $T_0$ …$T_m$ 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/Employee_Id	Employees/Supervisor_Id
T_0	T_1

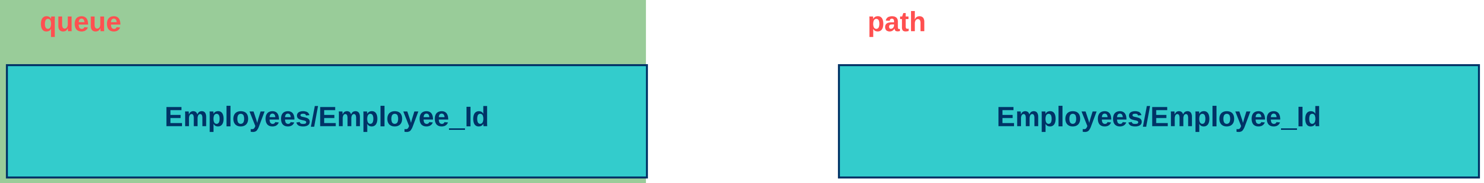
queue

Employees/Employee_Id

path

Employees/Employee_Id

```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0$ …$T_m$ 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
```



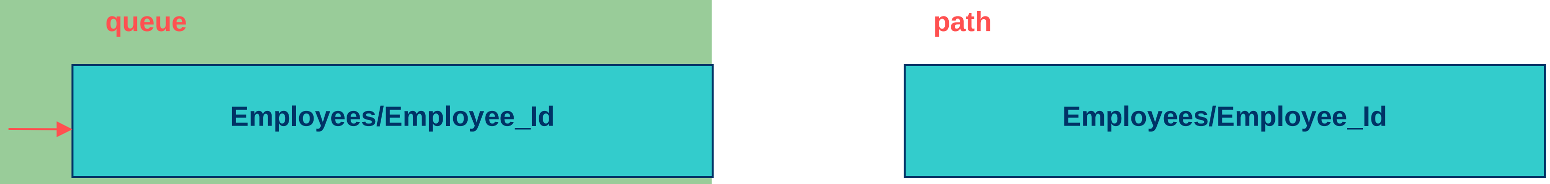
queue

Employees/Employee_Id

path

Employees/Employee_Id

```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0$ …$T_m$ 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
```

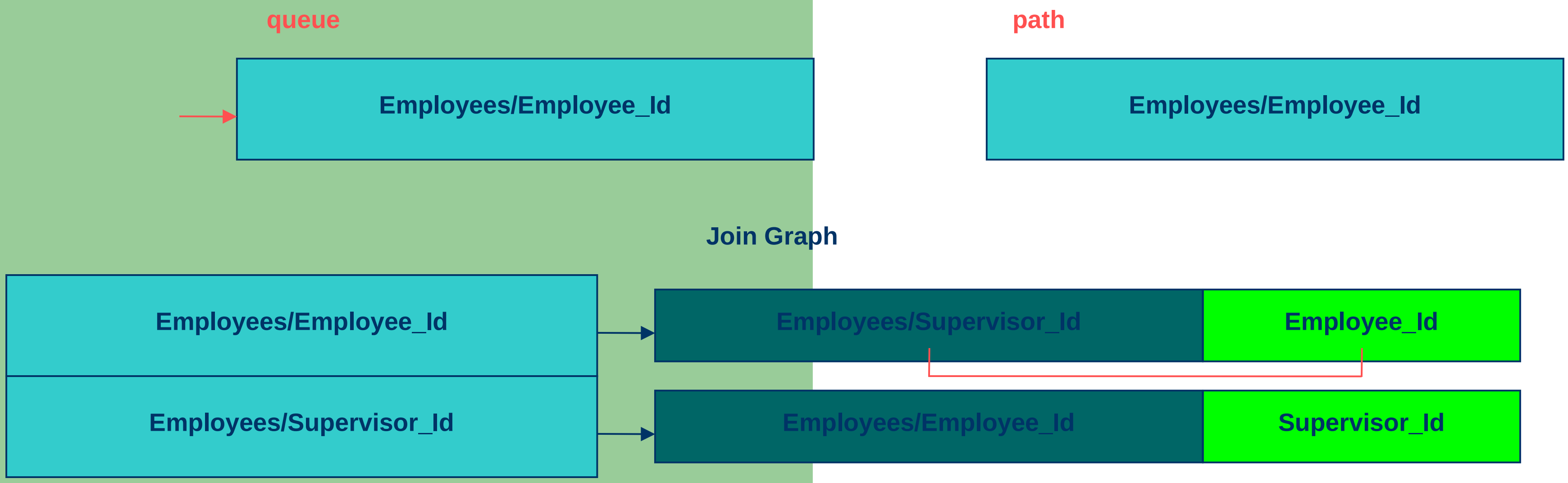

queue

Employees/Employee_Id

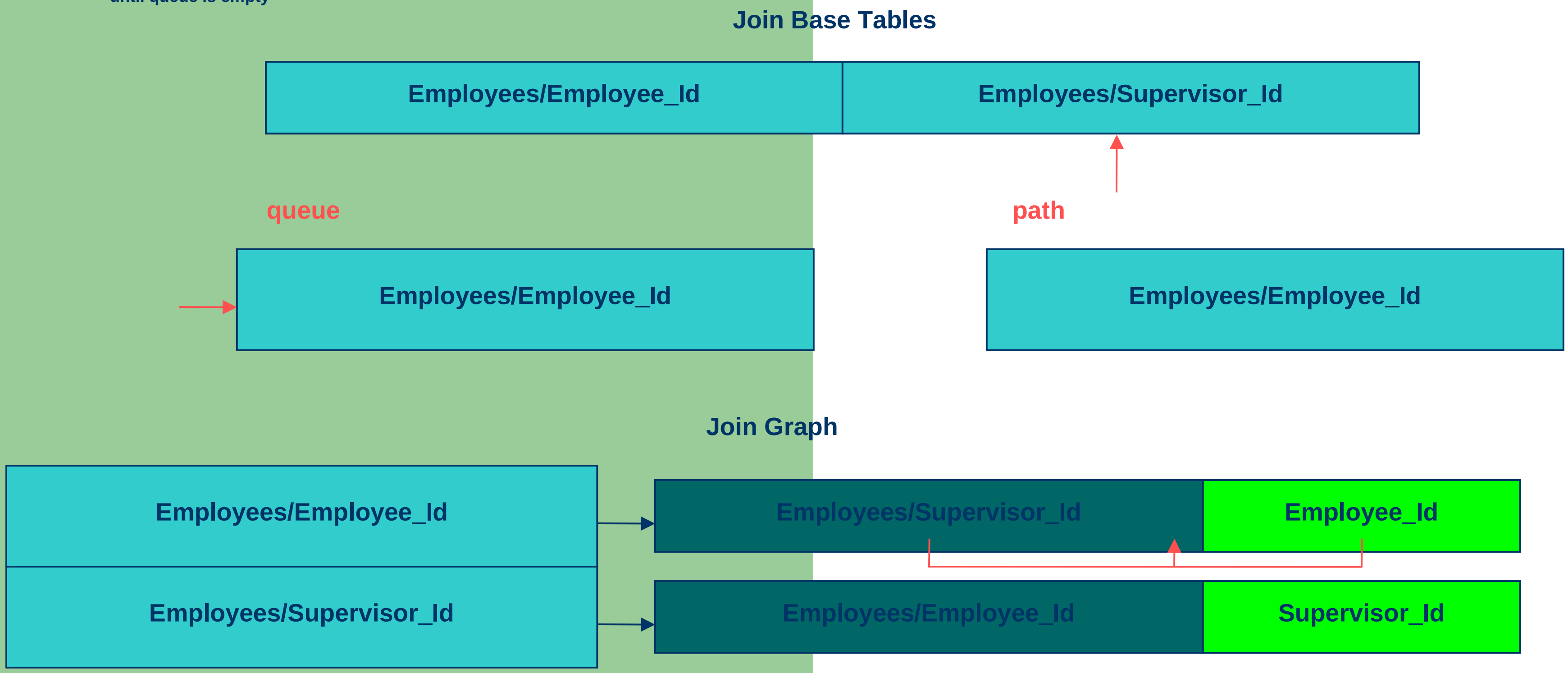
path

Employees/Employee_Id

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

let $T_0$ …$T_m$ 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 $T_0$ …$T_m$ 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/Employee_Id	Employees/Supervisor_Id

queue

Employees/Employee_Id

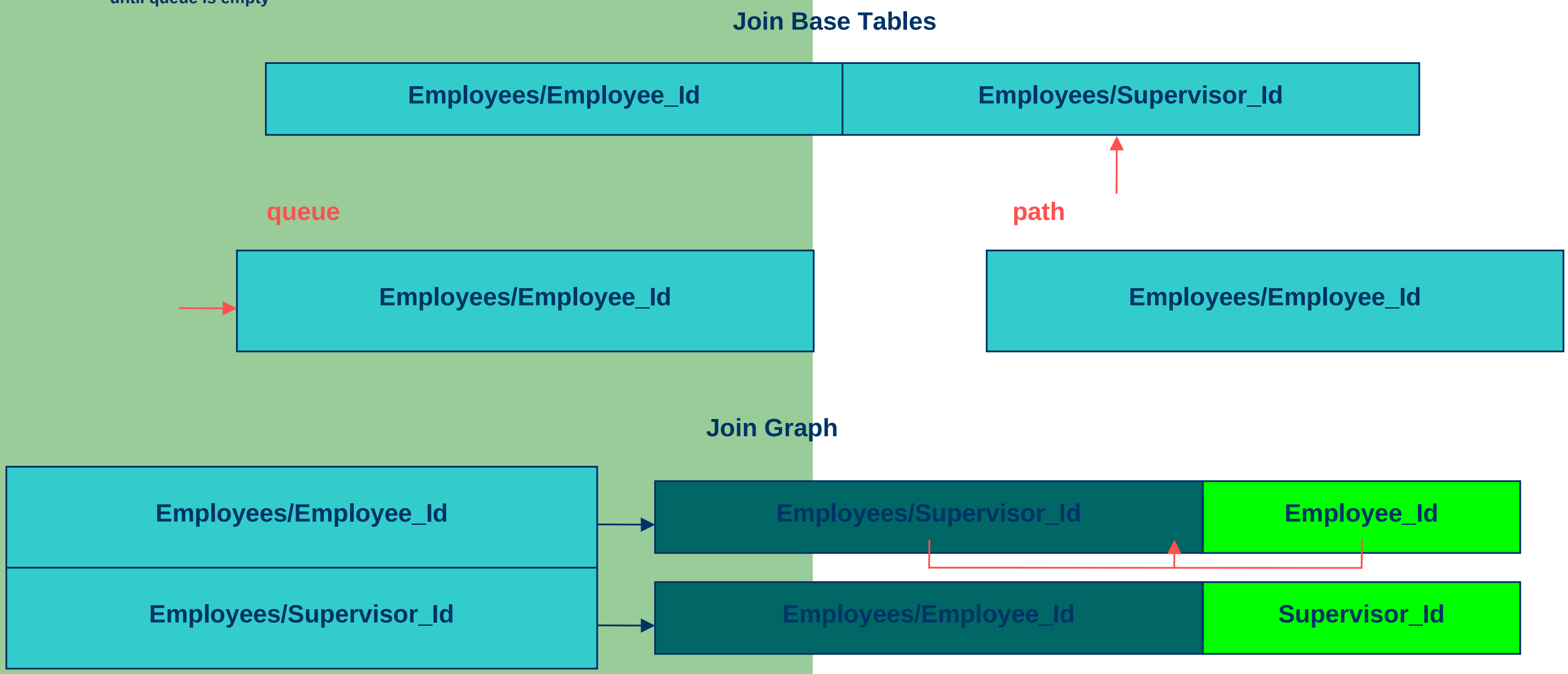
path

Employees/Employee_Id

Join Graph

Employees/Employee_Id	Employees/Supervisor_Id	Employee_Id
Employees/Supervisor_Id	Employees/Employee_Id	Supervisor_Id

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

let $T_0 \dots T_m$ 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/Employee_Id	Employees/Supervisor_Id

queue

Employees/Employee_Id

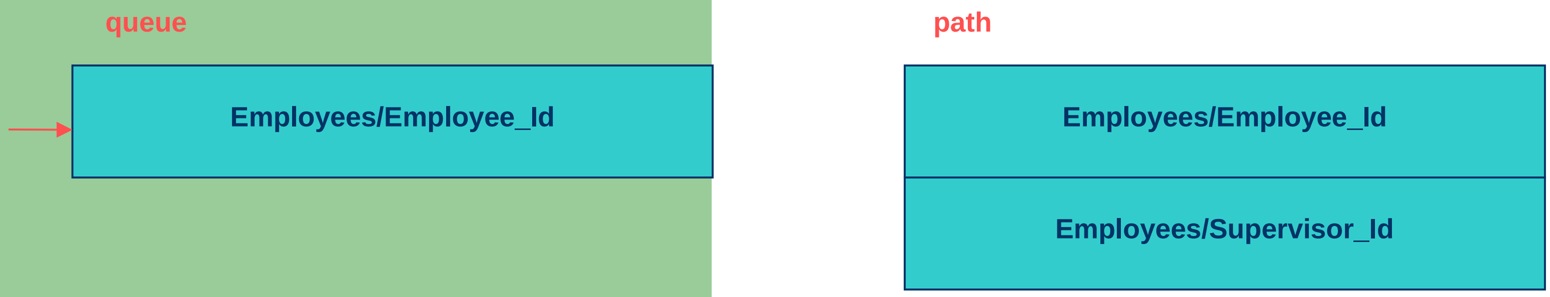
path

Employees/Employee_Id

Join Graph

Employees/Employee_Id	Employees/Supervisor_Id	Employee_Id
Employees/Supervisor_Id	Employees/Employee_Id	Supervisor_Id

```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0$ …$T_m$ 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

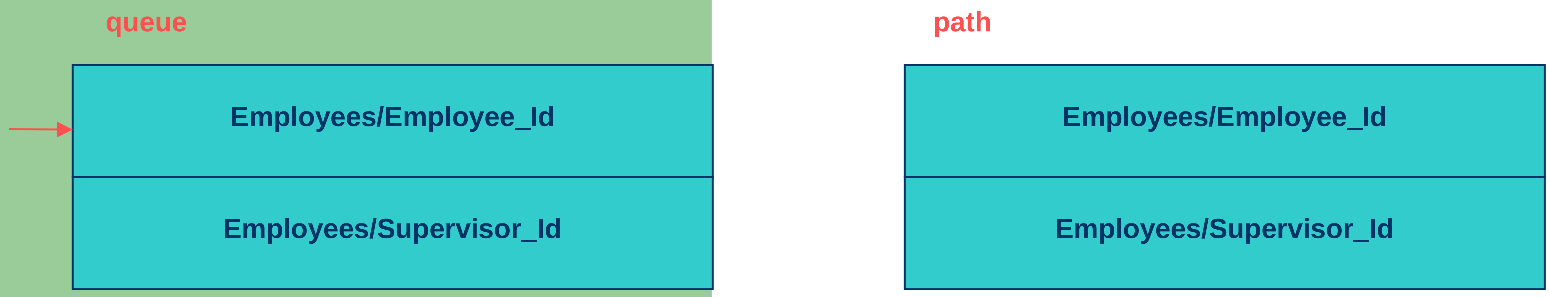
Employees/Employee_Id

path

Employees/Employee_Id

Employees/Supervisor_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
```



queue

Employees/Employee_Id

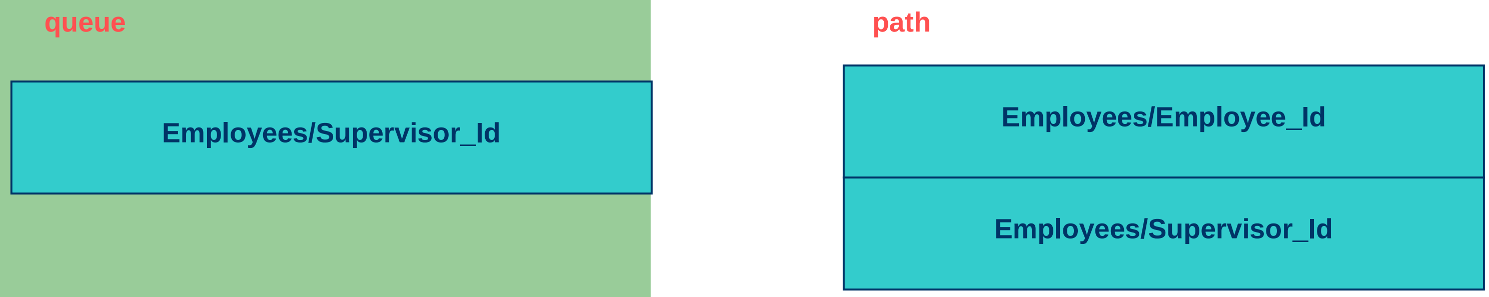
Employees/Supervisor_Id

path

Employees/Employee_Id

Employees/Supervisor_Id

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

let $T_0$ …$T_m$ 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

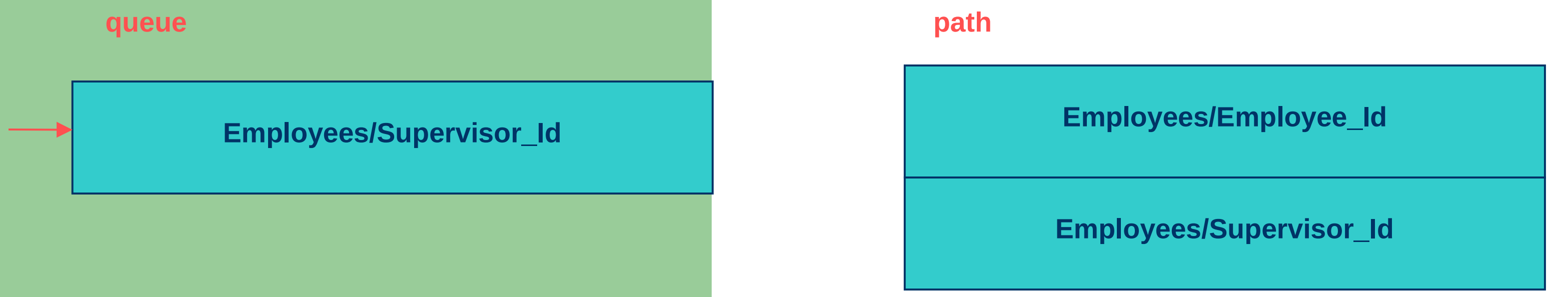
Employees/Supervisor_Id

path

Employees/Employee_Id

Employees/Supervisor_Id

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

let $T_0$ …$T_m$ 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

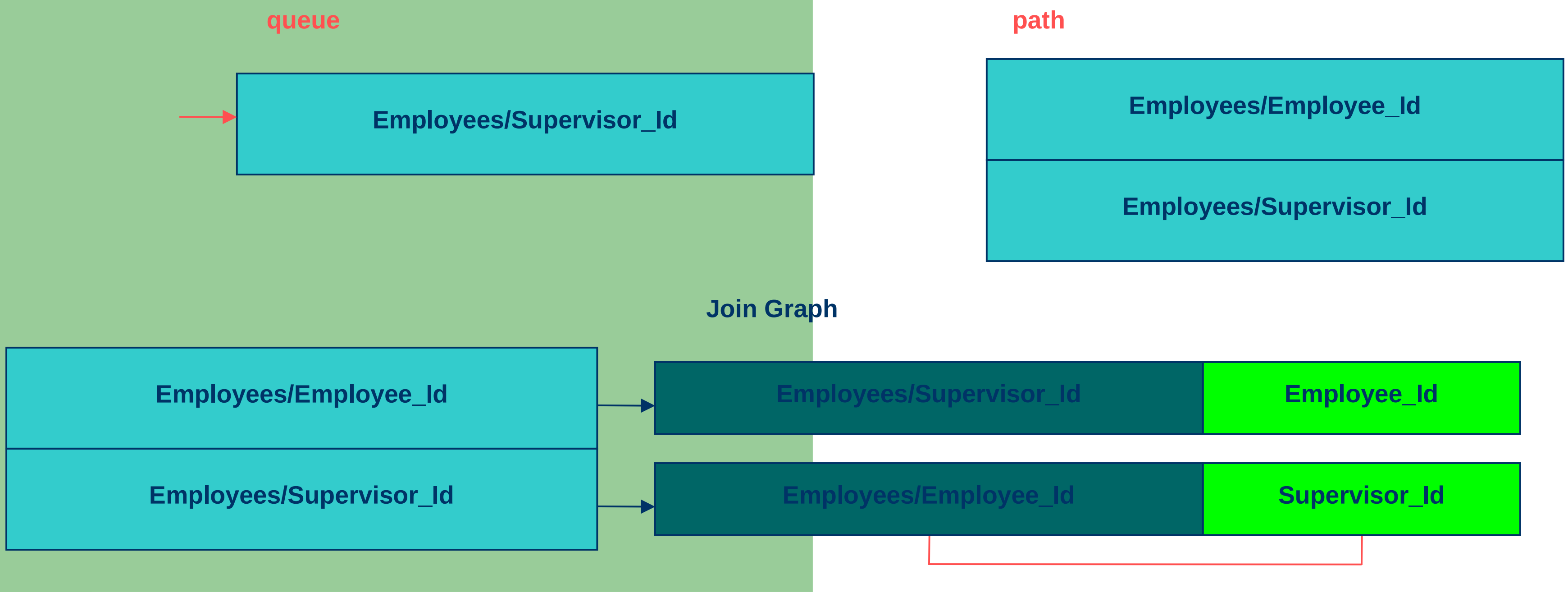
Employees/Supervisor_Id

path

Employees/Employee_Id

Employees/Supervisor_Id

```
generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)
let $T_0$ …$T_m$ 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/Supervisor_Id

path

Employees/Employee_Id

Employees/Supervisor_Id

Join Graph

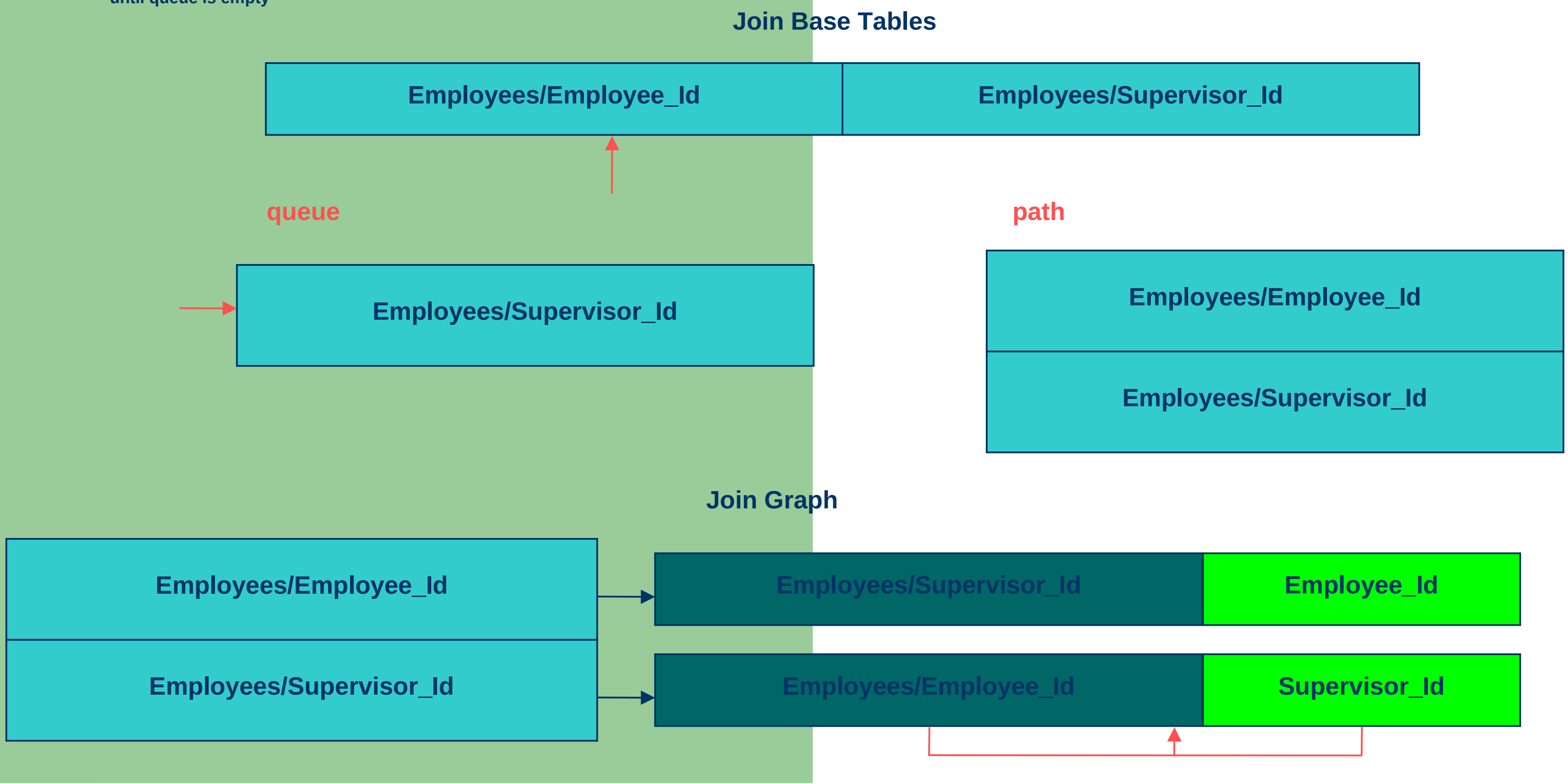
Employees/Employee_Id

Employees/Supervisor_Id

Employees/Supervisor_Id | Employee_Id

Employees/Employee_Id | Supervisor_Id

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

let $T_0$ …$T_m$ 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/Employee_Id	Employees/Supervisor_Id

queue

Employees/Supervisor_Id

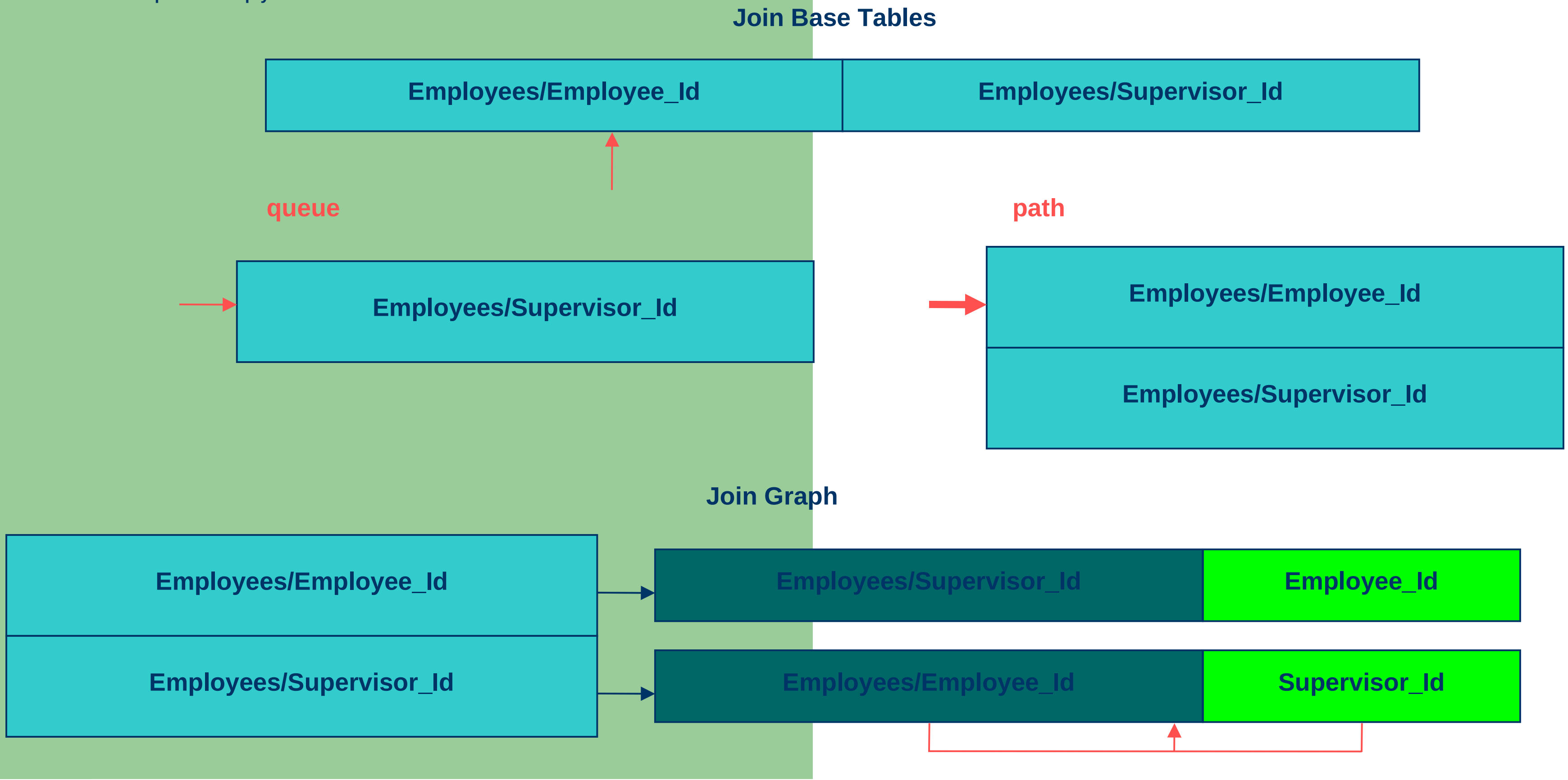
path

Employees/Employee_Id
Employees/Supervisor_Id

Join Graph

Employees/Employee_Id	Employees/Supervisor_Id	Employee_Id
Employees/Supervisor_Id	Employees/Employee_Id	Supervisor_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
```



Join Base Tables

Employees/Employee_Id

Employees/Supervisor_Id

queue

Employees/Supervisor_Id

path

Employees/Employee_Id

Employees/Supervisor_Id

Join Graph

Employees/Employee_Id

Employees/Supervisor_Id

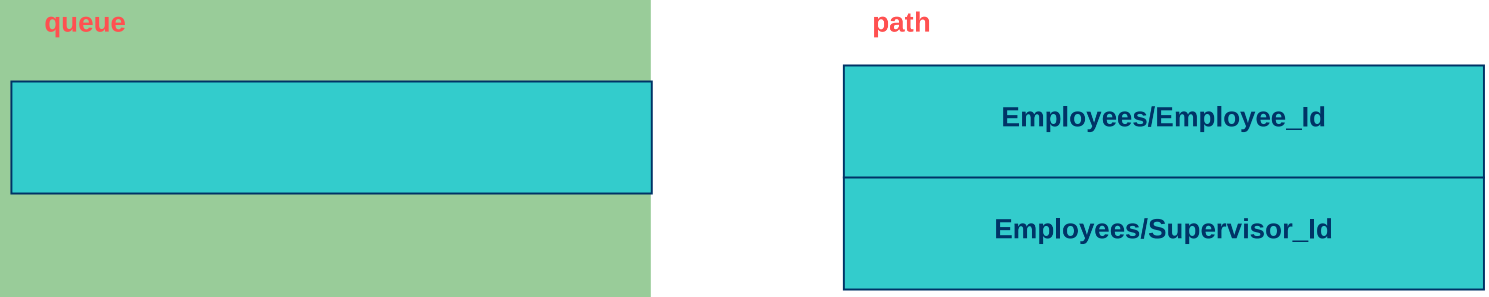
Employees/Supervisor_Id

Employee_Id

Employees/Employee_Id

Supervisor_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
```



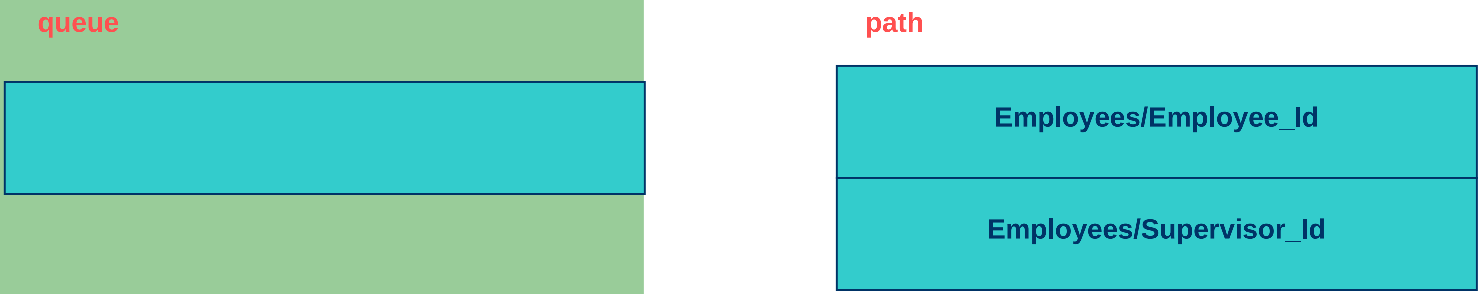
queue

path

Employees/Employee_Id

Employees/Supervisor_Id

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

let $T_0$ …$T_m$ 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[$T_{[buf]}$] = $T_{[buf]}$
```

Nodes

Employees/Employee_Id
Employees/Supervisor_Id

path

Employees/Employee_Id
Employees/Supervisor_Id

```
  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[$T_{[buf]}$] = $T_{[buf]}$
```

Nodes

Employees/Employee_Id

Employees/Supervisor_Id

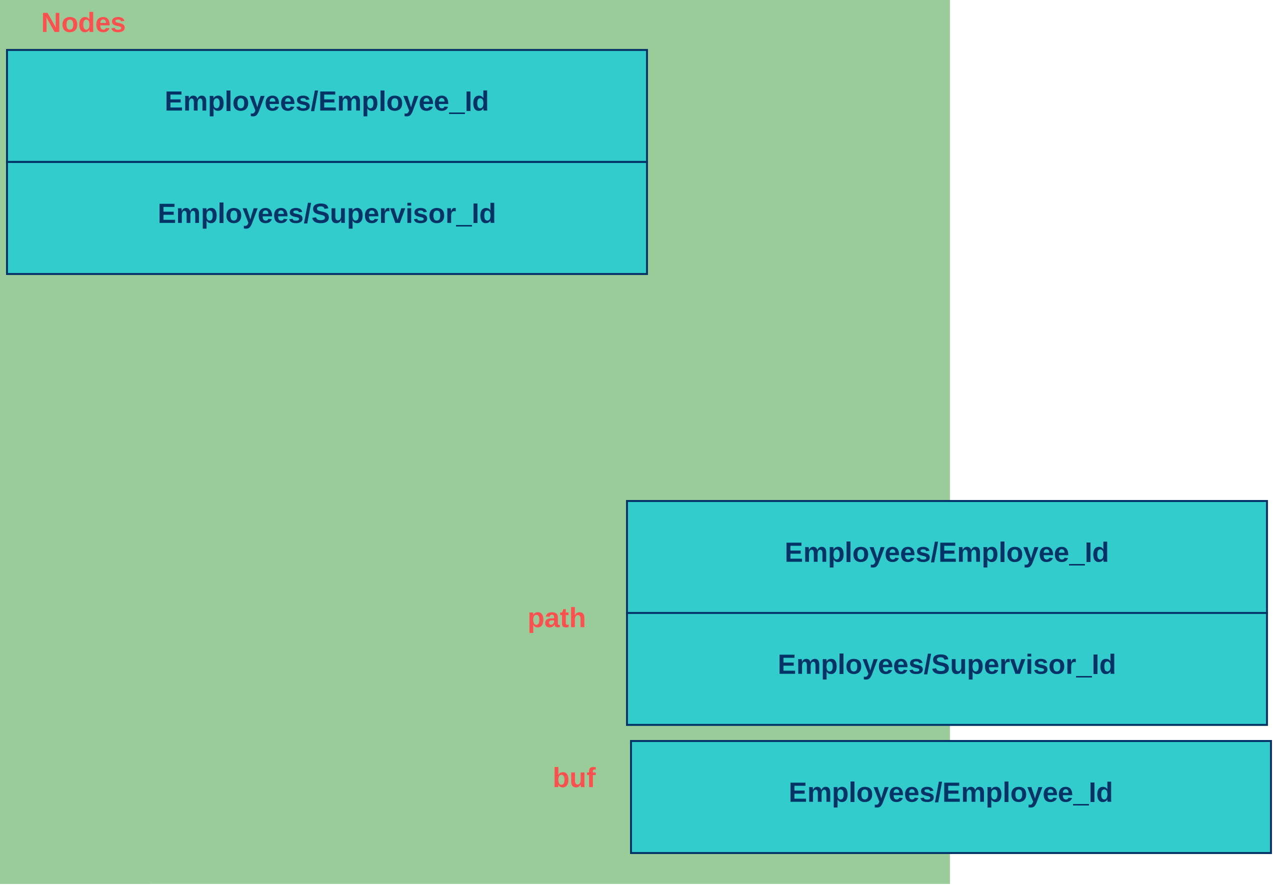
path

Employees/Employee_Id

Employees/Supervisor_Id

buf

```
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[$T_{[buf]}$] = $T_{[buf]}$
```



Nodes

Employees/Employee_Id

Employees/Supervisor_Id

path

Employees/Employee_Id

Employees/Supervisor_Id

buf

Employees/Employee_Id

```
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[$T_{[buf]}$ ] = $T_{[buf]}$
```

Nodes

Employees/Employee_Id
Employees/Supervisor_Id

path

Employees/Employee_Id
Employees/Supervisor_Id

buf

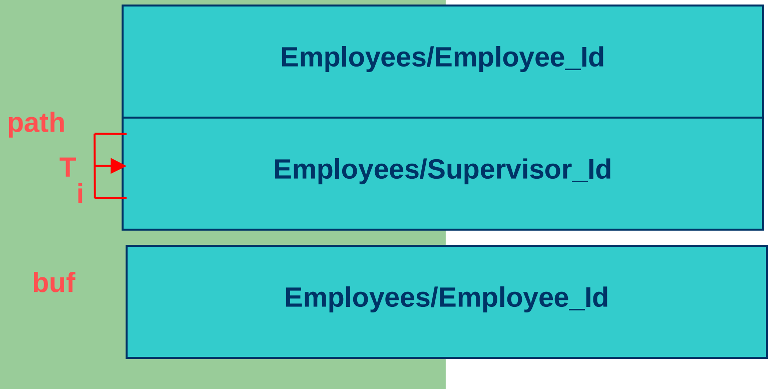
Employees/Employee_Id

```
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[$T_{[buf]}$] = $T_{[buf]}$
```

Nodes

Employees/Employee_Id

Employees/Supervisor_Id



path

Employees/Employee_Id

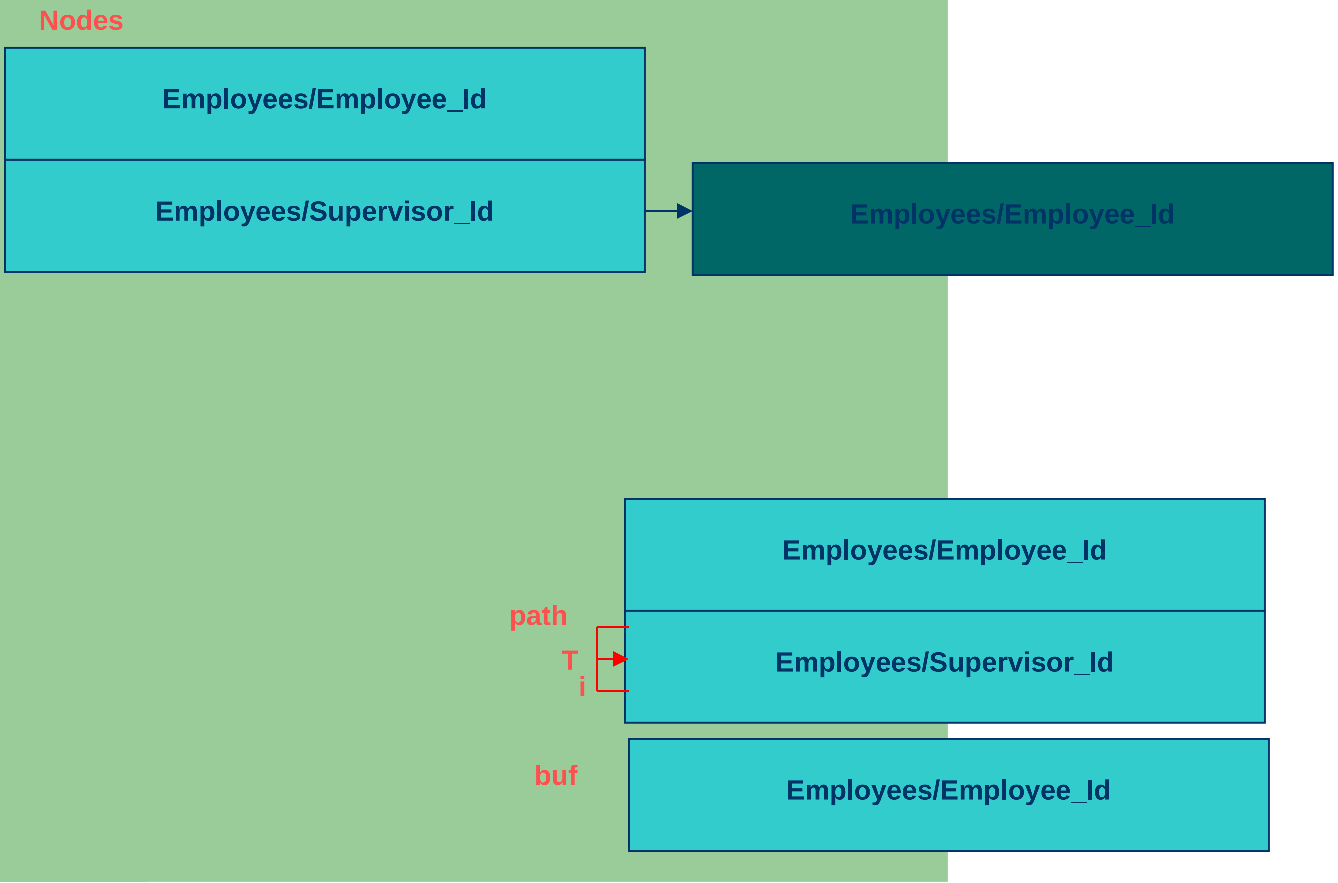
Employees/Supervisor_Id

T_i

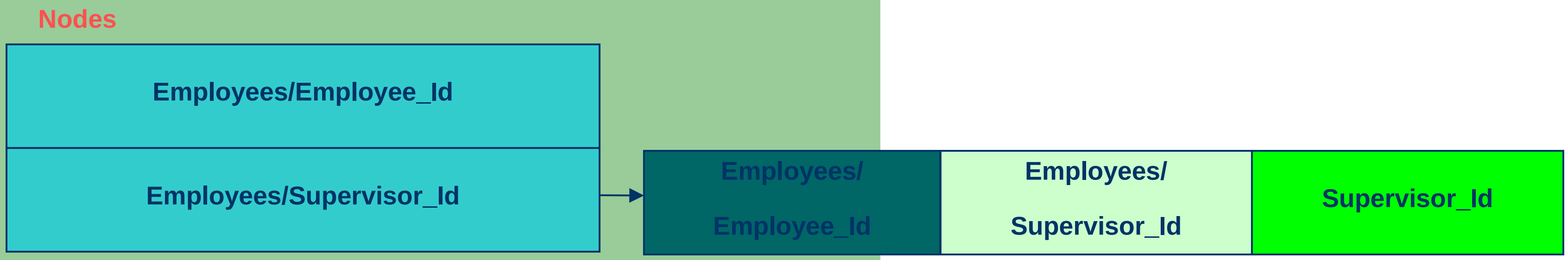
buf

Employees/Employee_Id

```
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[$T_{[buf]}$] = $T_{[buf]}$
```



```
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[$T_{[buf]}$] = $T_{[buf]}$
```



Nodes

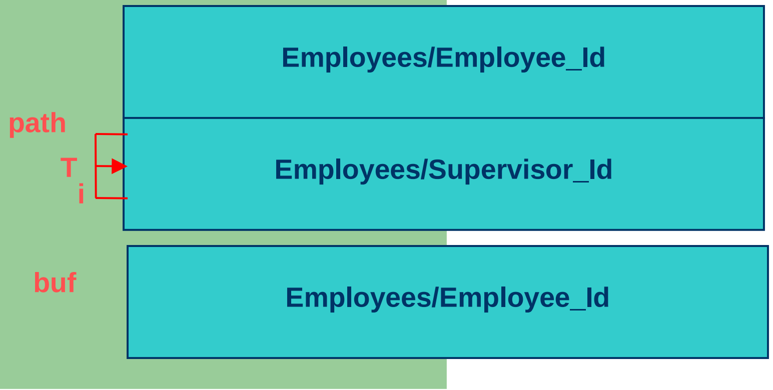
Employees/Employee_Id

Employees/Supervisor_Id

Employees/ Employee_Id

Employees/ Supervisor_Id

Supervisor_Id



path

T_i

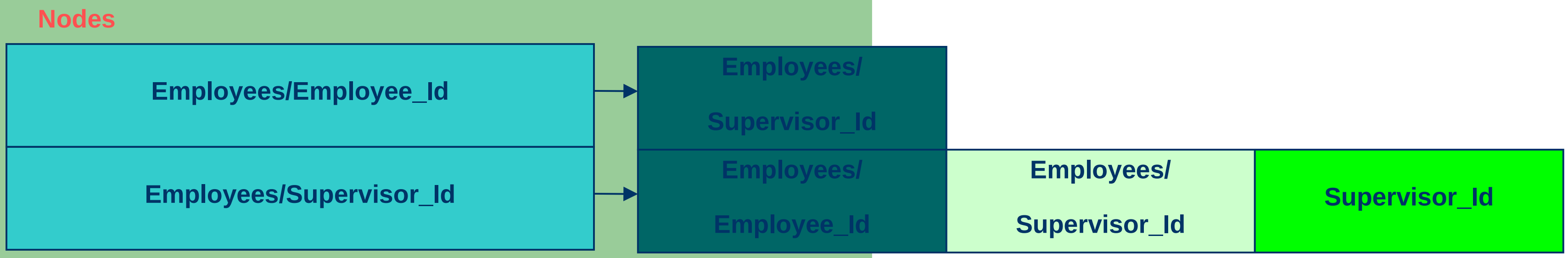
Employees/Employee_Id

Employees/Supervisor_Id

buf

Employees/Employee_Id

```
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[$T_{[buf]}$] = $T_{[buf]}$
```



Nodes

Employees/Employee_Id

Employees/Supervisor_Id

Employees/

Supervisor_Id

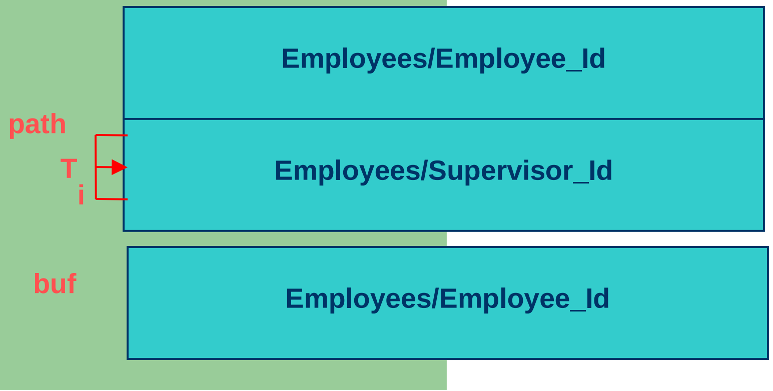
Employees/

Employee_Id

Employees/

Supervisor_Id

Supervisor_Id



path

T_i

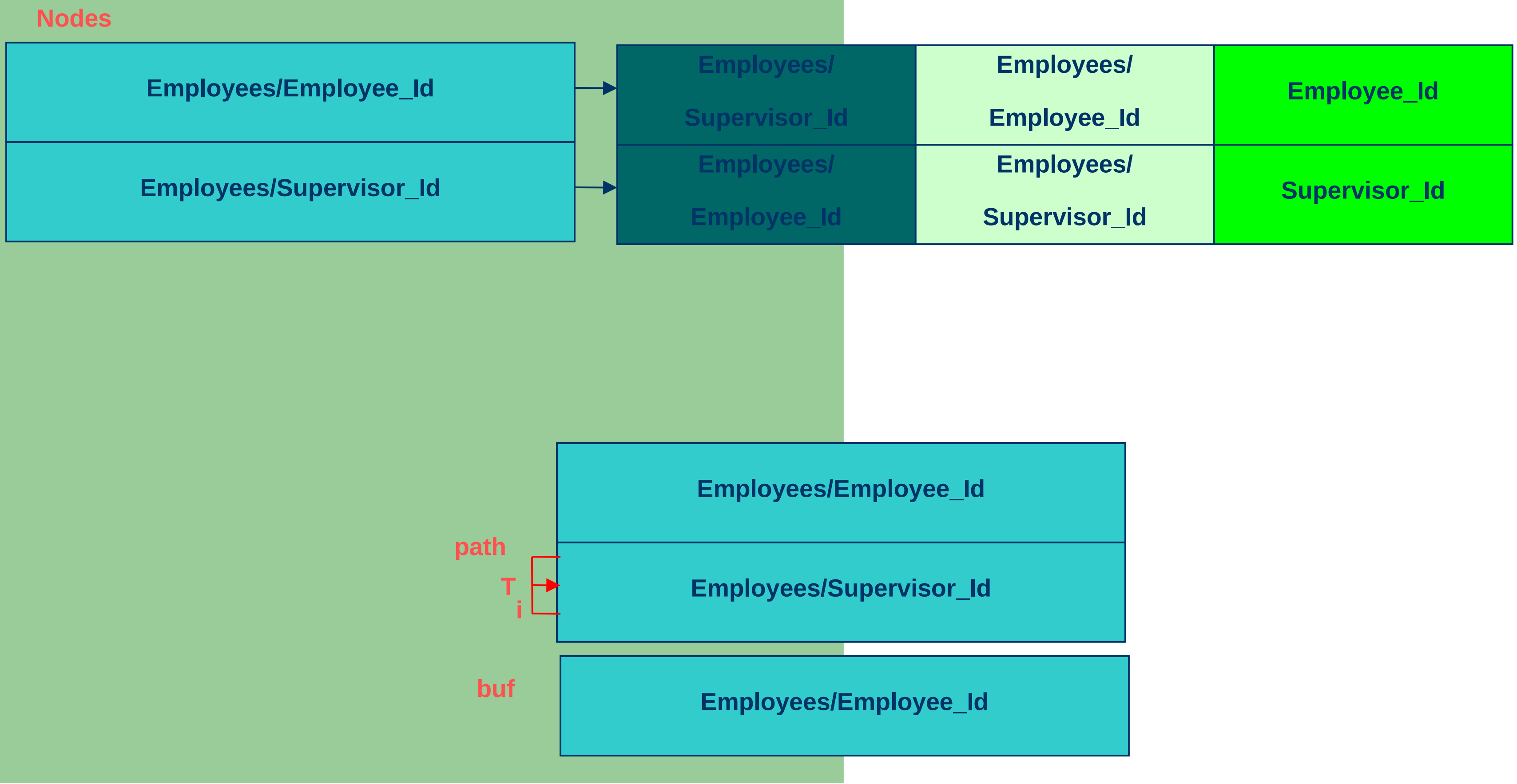
Employees/Employee_Id

Employees/Supervisor_Id

buf

Employees/Employee_Id

```
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[$T_{[buf]}$] = $T_{[buf]}$
```



Nodes

Employees/Employee_Id	Employees/ Supervisor_Id	Employees/ Employee_Id	Employee_Id
Employees/Supervisor_Id	Employees/ Employee_Id	Employees/ Supervisor_Id	Supervisor_Id

path

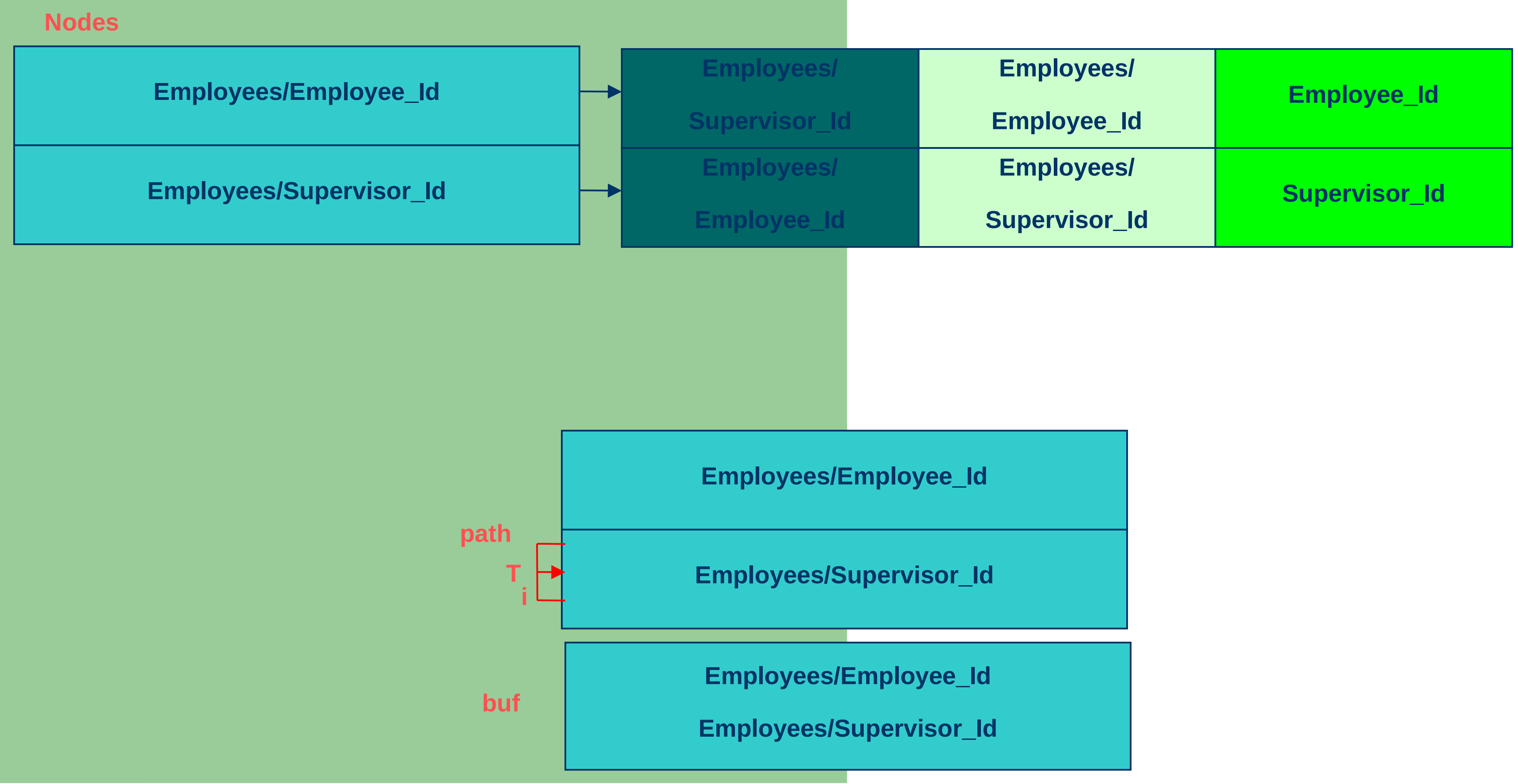
Employees/Employee_Id
Employees/Supervisor_Id

T_i

buf

Employees/Employee_Id

```
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[$T_{[buf]}$] = $T_{[buf]}$
```



Nodes

Employees/Employee_Id	Employees/ Supervisor_Id	Employees/ Employee_Id	Employee_Id
Employees/Supervisor_Id	Employees/ Employee_Id	Employees/ Supervisor_Id	Supervisor_Id

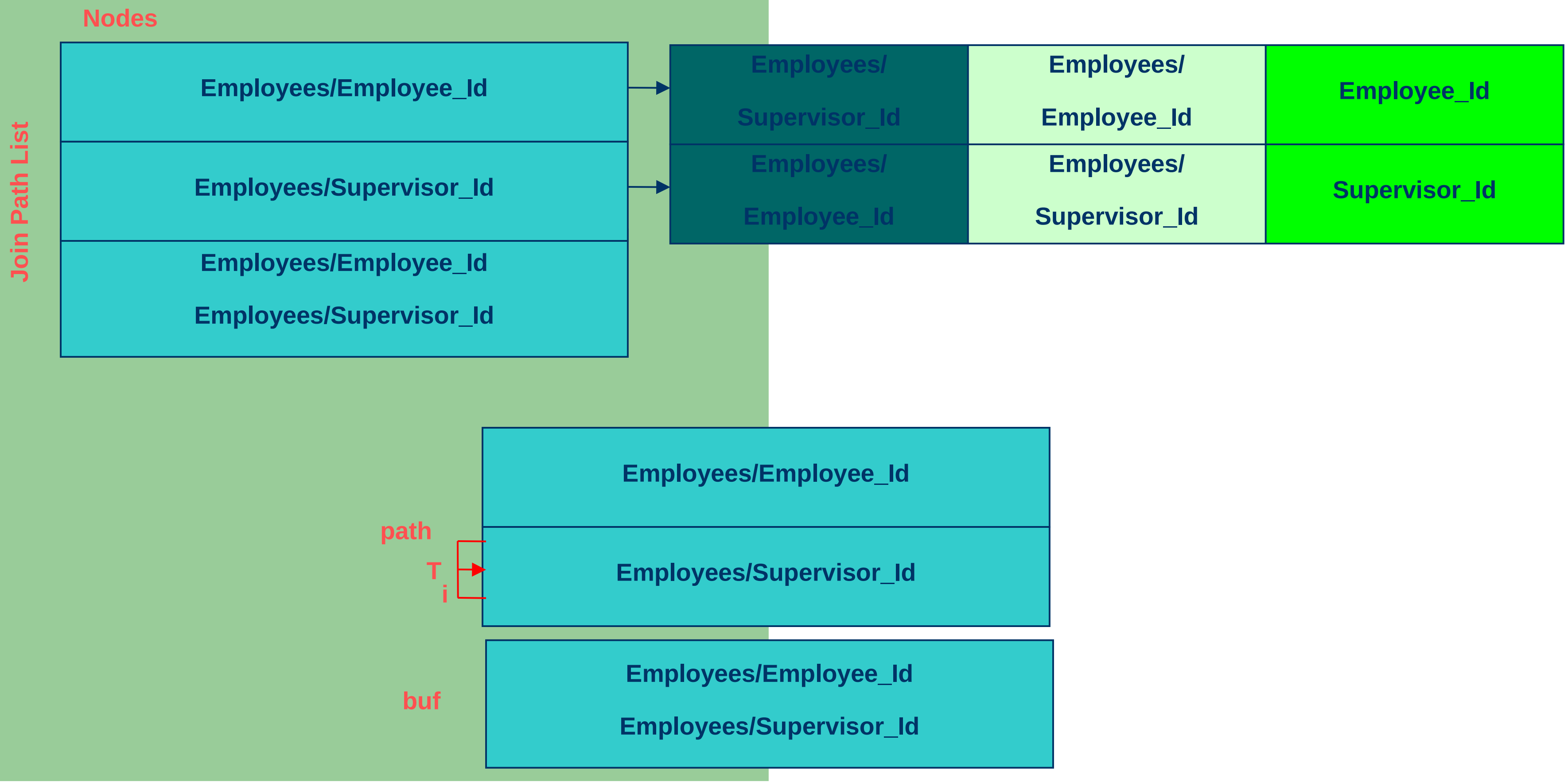
path

Employees/Employee_Id
Employees/Supervisor_Id

buf

Employees/Employee_Id Employees/Supervisor_Id

```
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[$T_{[buf]}$] = $T_{[buf]}$
```



Nodes

Join Path List

Employees/Employee_Id	Employees/ Supervisor_Id	Employees/ Employee_Id	Employee_Id
Employees/Supervisor_Id	Employees/ Employee_Id	Employees/ Supervisor_Id	Supervisor_Id
Employees/Employee_Id Employees/Supervisor_Id			

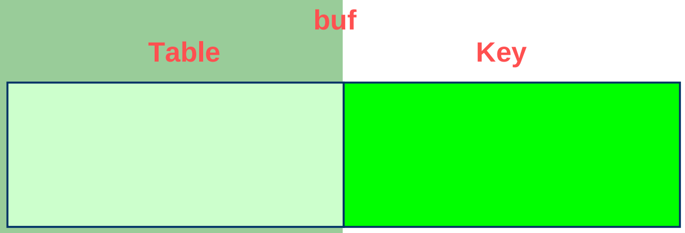
path

Employees/Employee_Id
T_i → Employees/Supervisor_Id

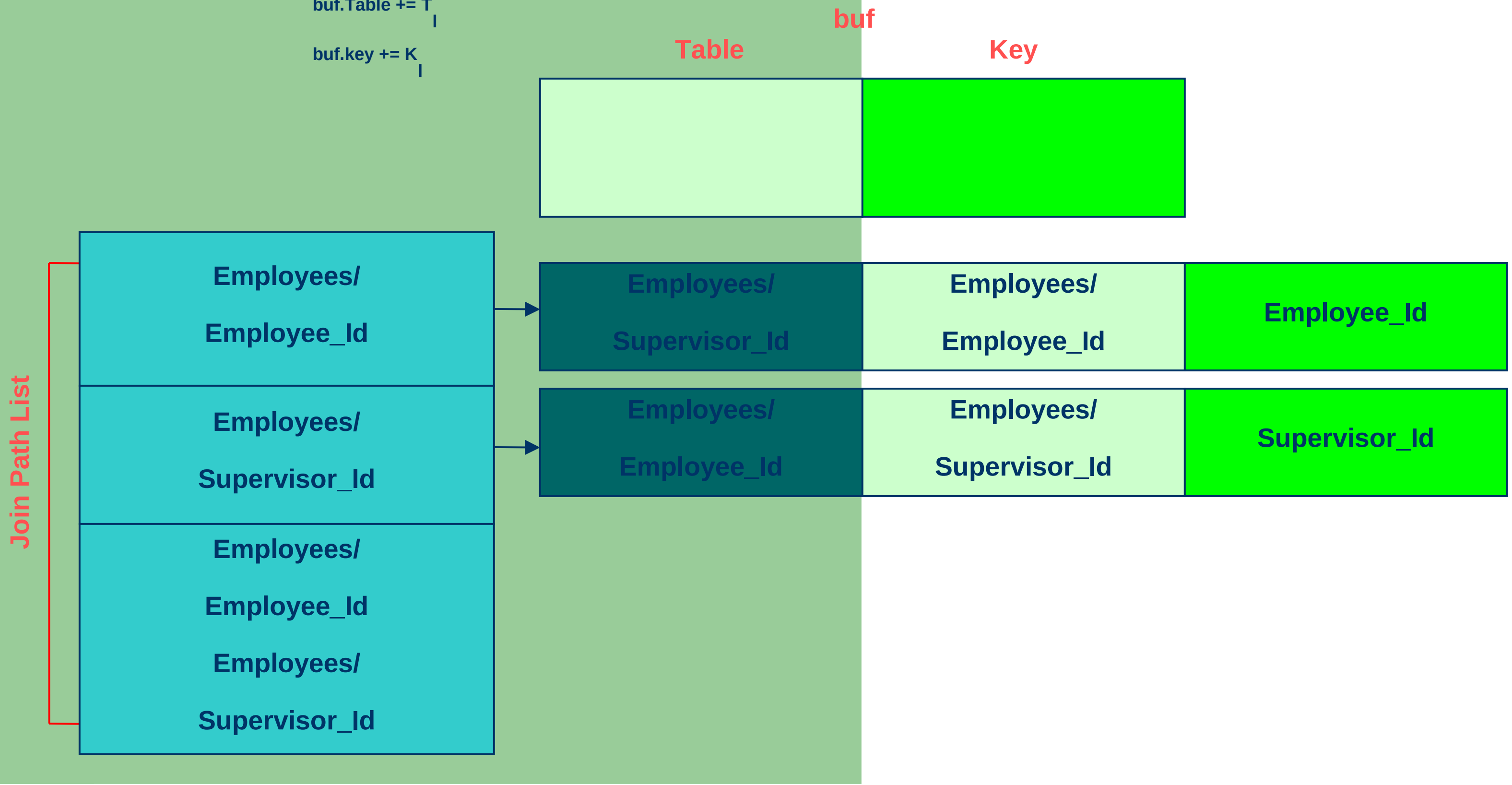
buf

Employees/Employee_Id Employees/Supervisor_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 all the couples of <Table,key>  in buf do
            take one couple $buf_c$ at a time
            if ($buf_c$.Table = $T_k$) then
                if ($buf_c$ != Key($T_{[i]}$) ) and ($buf_c$ not in InheritedKey($T_{[i]}$)) then
                    InheritedKey($T_{[i]}$)  += $buf_c$
    if $T_{[i]}$ is not a base table then
        if $T_l$ is the table from which comes Key($T_{[i]}$) and $k_l$ is the respective key then
            buf.Table += $T_l$
            buf.key += $K_l$
```



```
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 all the couples of <Table,key> in buf do
            take one couple $buf_c$ at a time
            if ($buf_c$.Table = $T_k$) then
                if ($buf_c$ != Key($T_{[i]}$) ) and ($buf_c$ not in InheritedKey($T_{[i]}$)) then
                    InheritedKey($T_{[i]}$) += $buf_c$
    if $T_{[i]}$ is not a base table then
        if $T_l$ is the table from which comes Key($T_{[i]}$) and $k_l$ is the respective key then
            buf.Table += $T_l$
            buf.key += $K_l$
```



buf

Table

Key

Join Path List

Employees/

Employee_Id

Employees/

Supervisor_Id

Employees/

Employee_Id

Employees/

Supervisor_Id

Employees/

Supervisor_Id

Employees/

Employee_Id

Employee_Id

Employees/

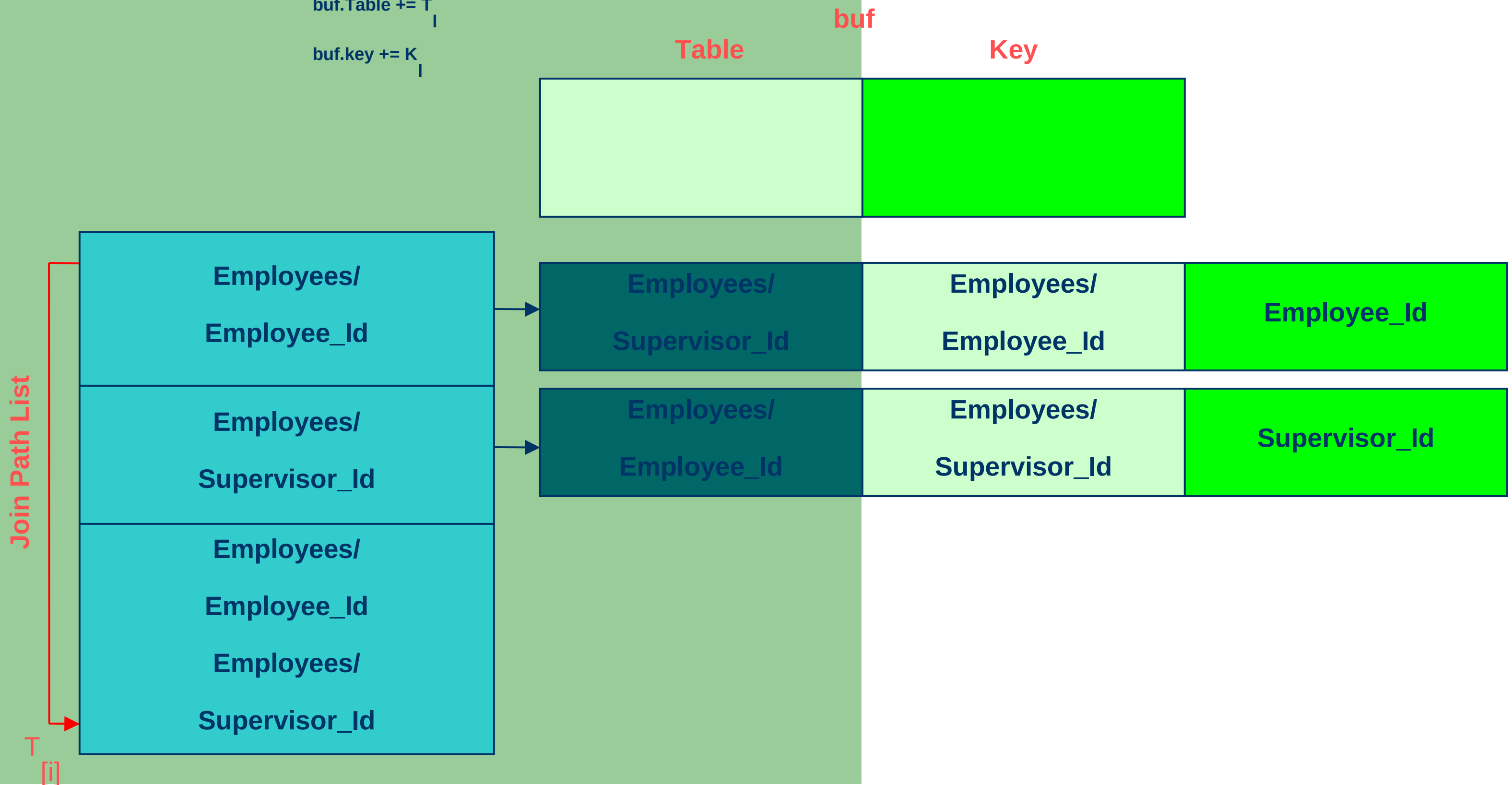
Employee_Id

Employees/

Supervisor_Id

Supervisor_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 all the couples of <Table,key>  in buf do
            take one couple $buf_c$ at a time
            if ($buf_c$.Table = $T_k$) then
                if ($buf_c$ != Key($T_{[i]}$) ) and ($buf_c$ not in InheritedKey($T_{[i]}$)) then
                    InheritedKey($T_{[i]}$)  += $buf_c$
    if $T_{[i]}$ is not a base table then
        if $T_l$ is the table from which comes Key($T_{[i]}$) and $k_l$ is the respective key then
            buf.Table += $T_l$
            buf.key += $K_l$
```



buf

Table

Key

Join Path List

Employees/

Employee_Id

Employees/

Supervisor_Id

Employees/

Employee_Id

Employees/

Supervisor_Id

Employees/

Supervisor_Id

Employees/

Employee_Id

Employee_Id

Employees/

Employee_Id

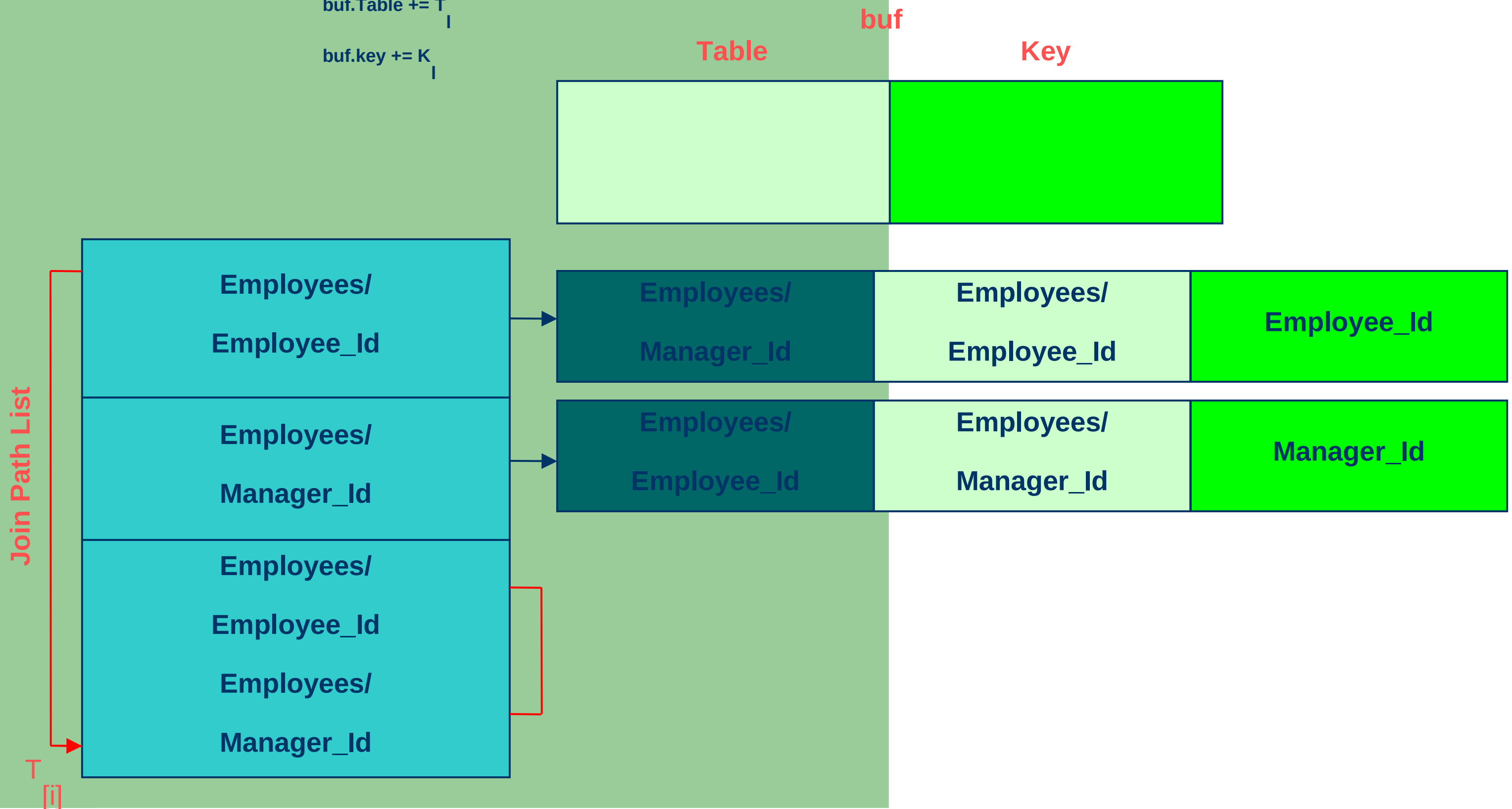
Employees/

Supervisor_Id

Supervisor_Id

$T_{[i]}$

```
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 all the couples of <Table,key>  in buf do
                        take one couple $buf_c$ at a time
                        if ($buf_c$.Table = $T_k$) then
                                if ($buf_c$ != Key($T_{[i]}$) ) and ($buf_c$ not in InheritedKey($T_{[i]}$)) then
                                    InheritedKey($T_{[i]}$)  += $buf_c$
        if $T_{[i]}$ is not a base table then
                if $T_l$ is the table from which comes Key($T_{[i]}$) and $k_l$ is the respective key then
                        buf.Table += $T_l$
                        buf.key += $K_l$
```



buf

Table

Key

Employees/

Employee_Id

Employees/

Manager_Id

Employees/

Employee_Id

Employees/

Manager_Id

Employees/

Manager_Id

Employees/

Employee_Id

Employee_Id

Employees/

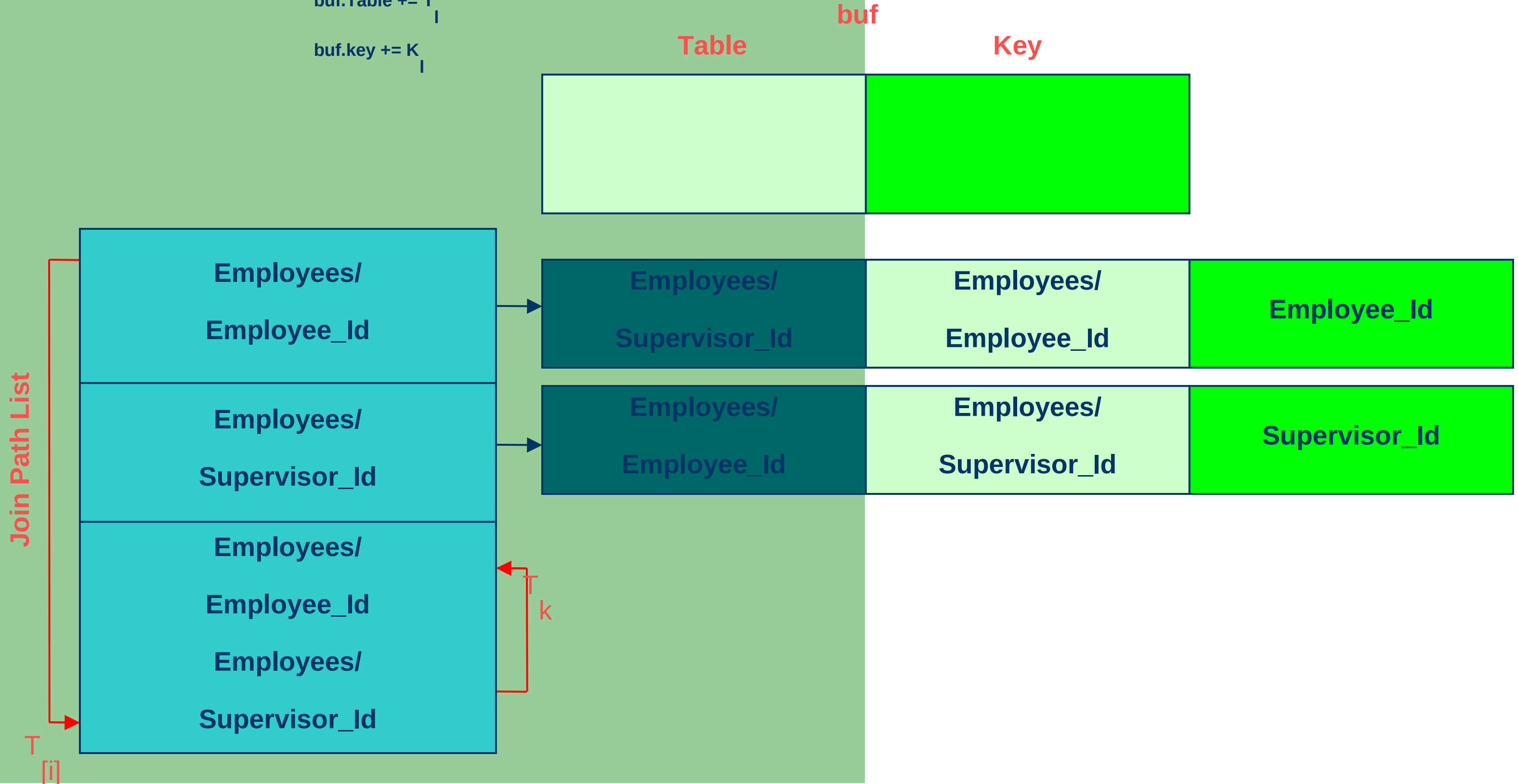
Employee_Id

Employees/

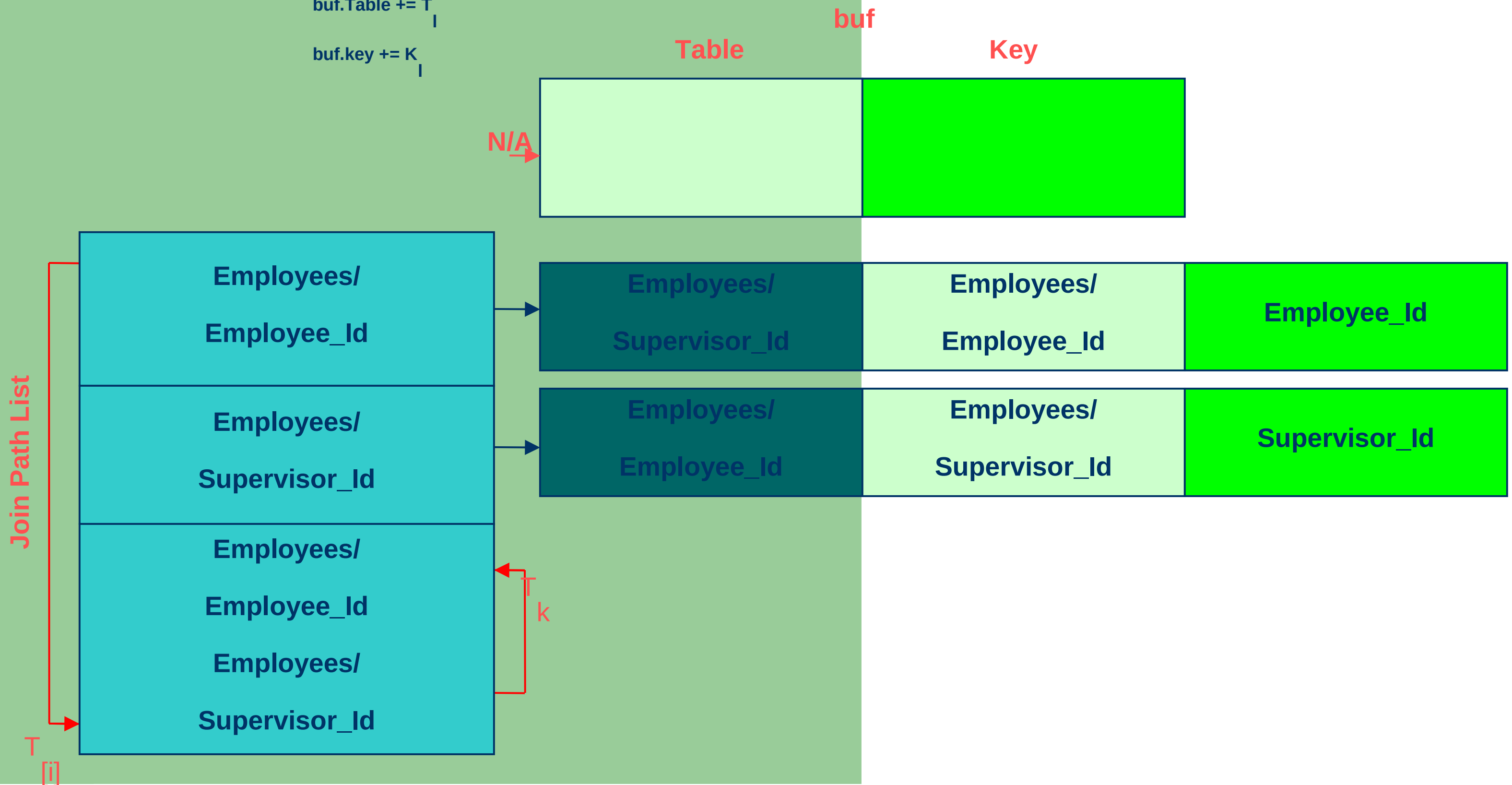
Manager_Id

Manager_Id

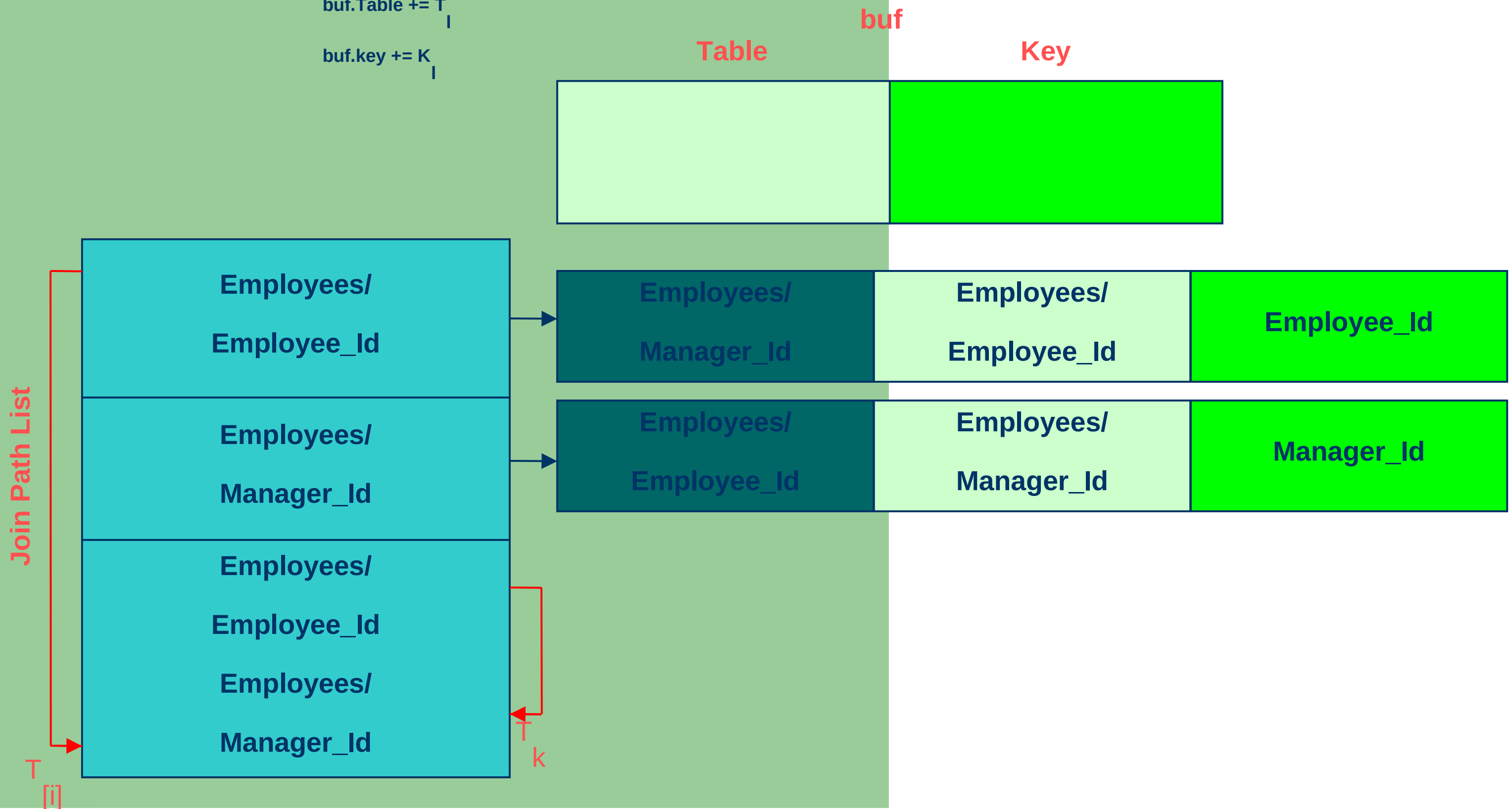
```
create a structure buf with 2 fields: Table and Key
for all the tables in JoinPathList going downward do
    take one $T_{[i]}$ at a time
    for all Base Tables in$T_{[i]}$ do
        take one $T_k$ at a time
        for all the couples of <Table,key>  in buf do
            take one couple $buf_c$ at a time
            if ($buf_c$.Table = $T_k$) then
                if ($buf_c$ != Key($T_{[i]}$) ) and ($buf_c$ not in InheritedKey($T_{[i]}$)) then
                    InheritedKey($T_{[i]}$)  += $buf_c$
    if $T_{[i]}$ is not a base table then
        if $T_l$ is the table from which comes Key($T_{[i]}$) and $k_l$ is the respective key then
            buf.Table += $T_l$
            buf.key += $K_l$
```



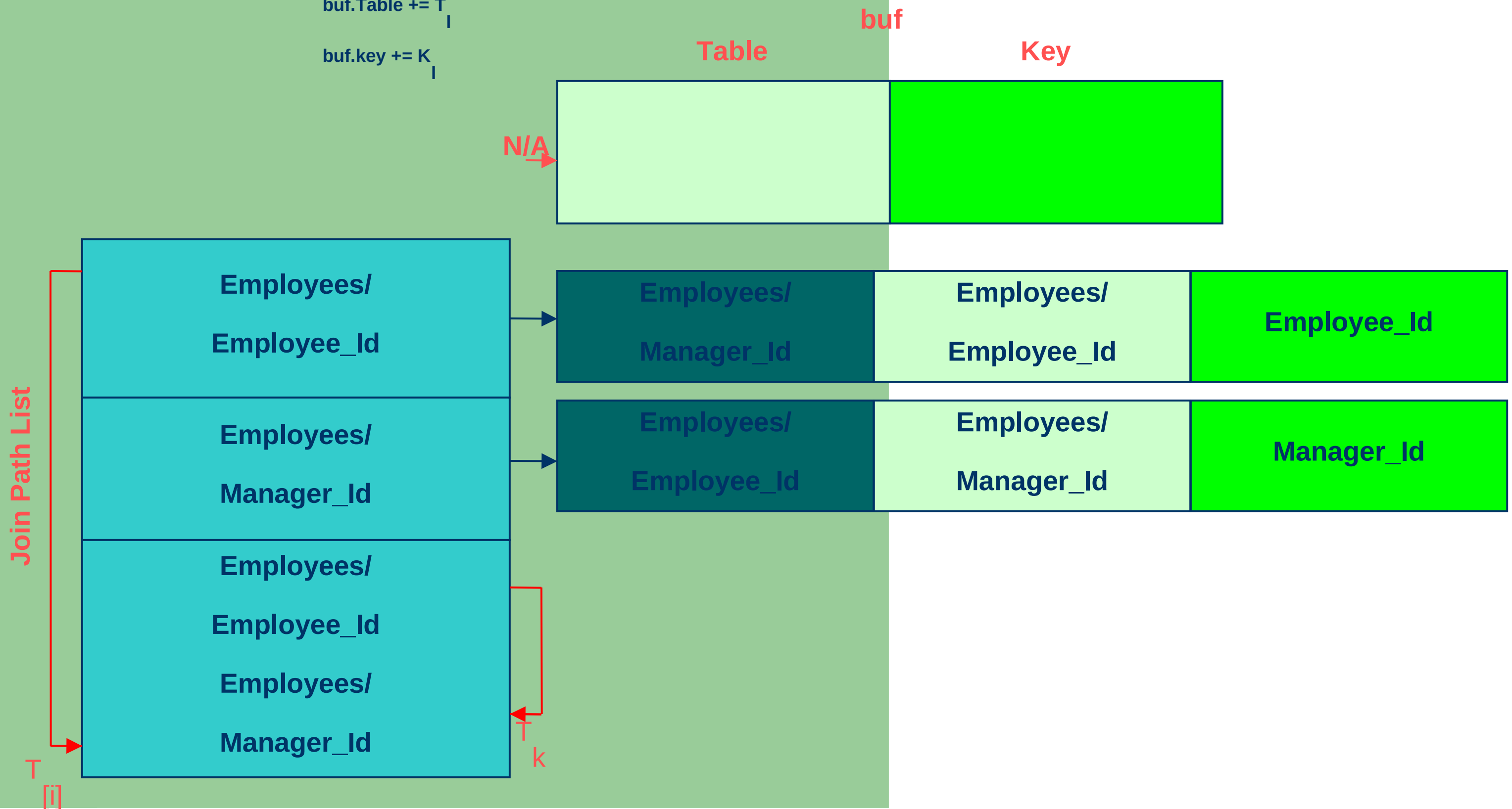
```
create a structure buf with 2 fields: Table and Key
for all the tables in JoinPathList going downward do
    take one $T_{[i]}$ at a time
    for all Base Tables in$T_{[i]}$ do
        take one $T_k$ at a time
        for all the couples of <Table,key>  in buf do
            take one couple $buf_c$ at a time
            if ($buf_c$.Table = $T_k$) then
                if ($buf_c$ != Key($T_{[i]}$) ) and ($buf_c$ not in InheritedKey($T_{[i]}$)) then
                    InheritedKey($T_{[i]}$)  += $buf_c$
    if $T_{[i]}$ is not a base table then
        if $T_l$ is the table from which comes Key($T_{[i]}$) and $k_l$ is the respective key then
            buf.Table += $T_l$
            buf.key += $K_l$
```



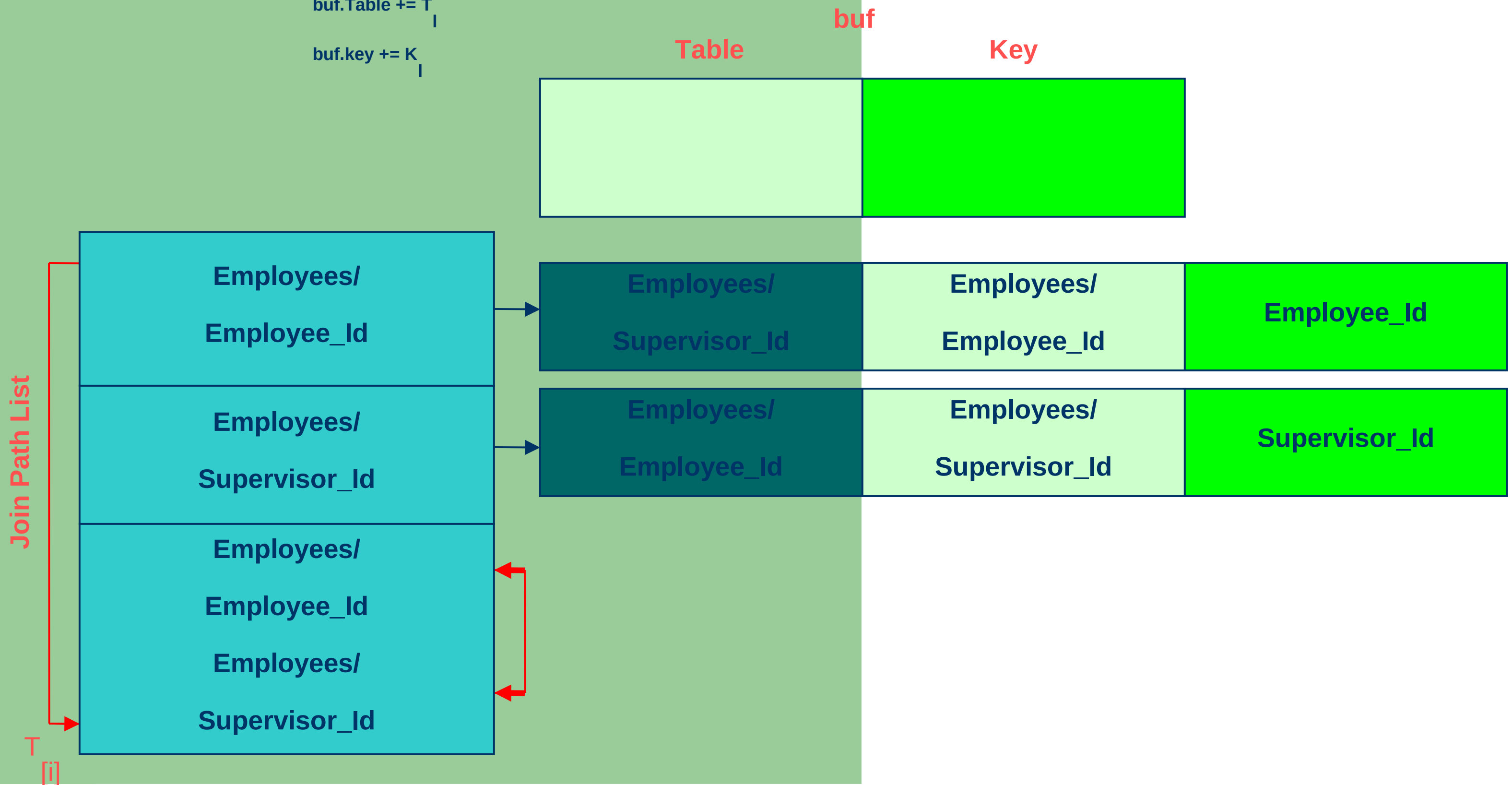
```
create a structure buf with 2 fields: Table and Key
for all the tables in JoinPathList going downward do
    take one $T_{[i]}$ at a time
    for all Base Tables in$T_{[i]}$ do
        take one $T_k$ at a time
        for all the couples of <Table,key>  in buf do
            take one couple $buf_c$ at a time
            if ($buf_c$.Table = $T_k$) then
                if ($buf_c$ != Key($T_{[i]}$) ) and ($buf_c$ not in InheritedKey($T_{[i]}$)) then
                    InheritedKey($T_{[i]}$)  += $buf_c$
    if $T_{[i]}$ is not a base table then
        if $T_l$ is the table from which comes Key($T_{[i]}$) and $k_l$ is the respective key then
            buf.Table += $T_l$
            buf.key += $K_l$
```



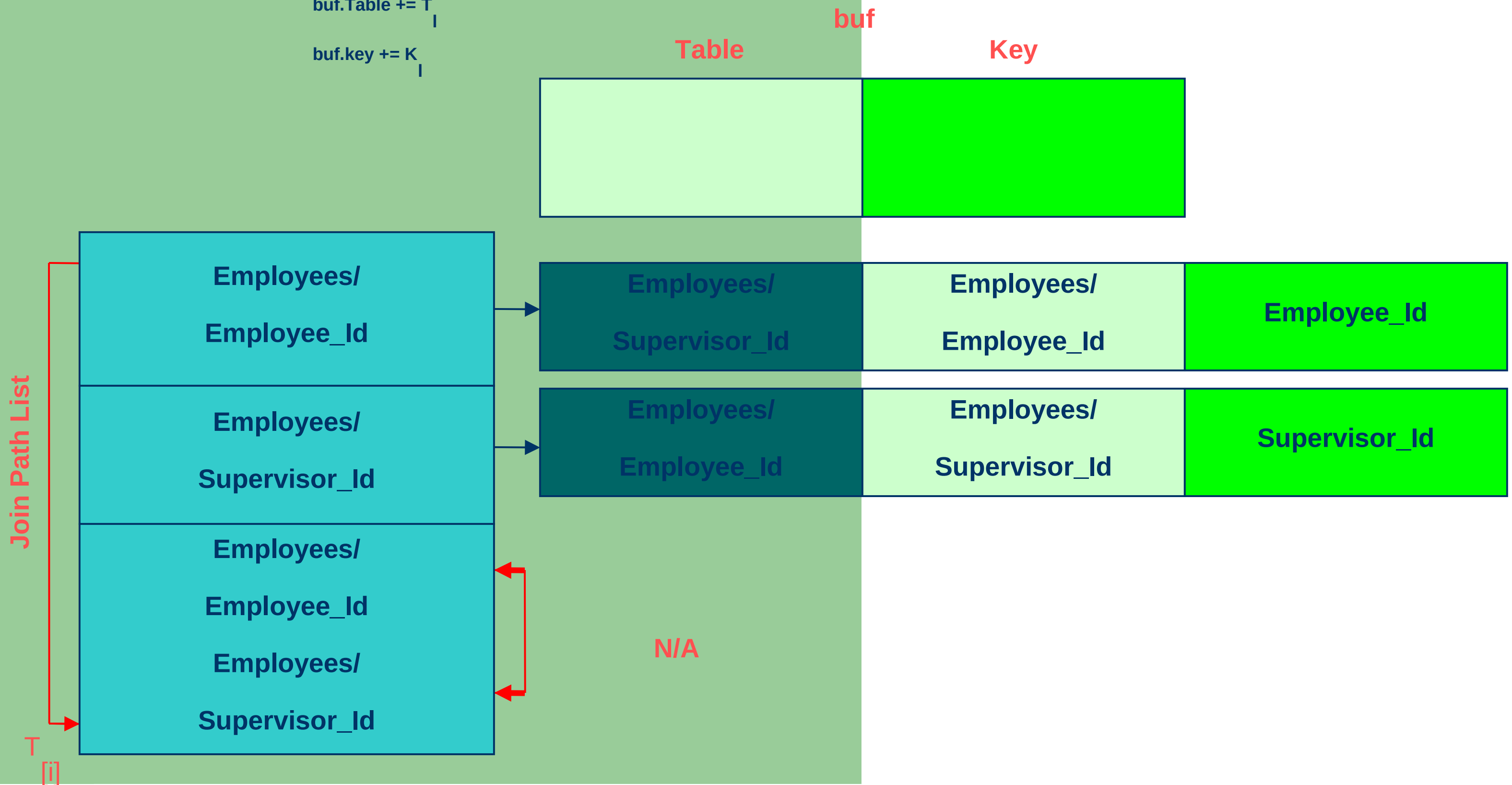
```
create a structure buf with 2 fields: Table and Key
for all the tables in JoinPathList going downward do
    take one $T_{[i]}$ at a time
    for all Base Tables in$T_{[i]}$ do
        take one $T_k$ at a time
        for all the couples of <Table,key>  in buf do
            take one couple $buf_c$ at a time
            if ($buf_c$.Table = $T_k$) then
                if ($buf_c$ != Key($T_{[i]}$) ) and ($buf_c$ not in InheritedKey($T_{[i]}$)) then
                    InheritedKey($T_{[i]}$)  += $buf_c$
    if $T_{[i]}$ is not a base table then
        if $T_l$ is the table from which comes Key($T_{[i]}$) and $k_l$ is the respective key then
            buf.Table += $T_l$
            buf.key += $K_l$
```



```
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 all the couples of <Table,key>  in buf do

            take one couple $buf_c$ at a time

            if ($buf_c$.Table = $T_k$) then

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

                    InheritedKey($T_{[i]}$)  += $buf_c$

    if $T_{[i]}$ is not a base table then

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

            buf.Table += $T_l$

            buf.key += $K_l$
```



```
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 all the couples of <Table,key> in buf do
            take one couple $buf_c$ at a time
            if ($buf_c$.Table = $T_k$) then
                if ($buf_c$ != Key($T_{[i]}$) ) and ($buf_c$ not in InheritedKey($T_{[i]}$)) then
                    InheritedKey($T_{[i]}$) += $buf_c$
    if $T_{[i]}$ is not a base table then
        if $T_l$ is the table from which comes Key($T_{[i]}$) and $k_l$ is the respective key then
            buf.Table += $T_l$
            buf.key += $K_l$
```



buf

Table

Key

Join Path List

Employees/
Employee_Id

Employees/
Supervisor_Id

Employees/
Employee_Id
Employees/
Supervisor_Id

$T_{[i]}$

Employees/
Supervisor_Id

Employees/
Employee_Id

Employee_Id

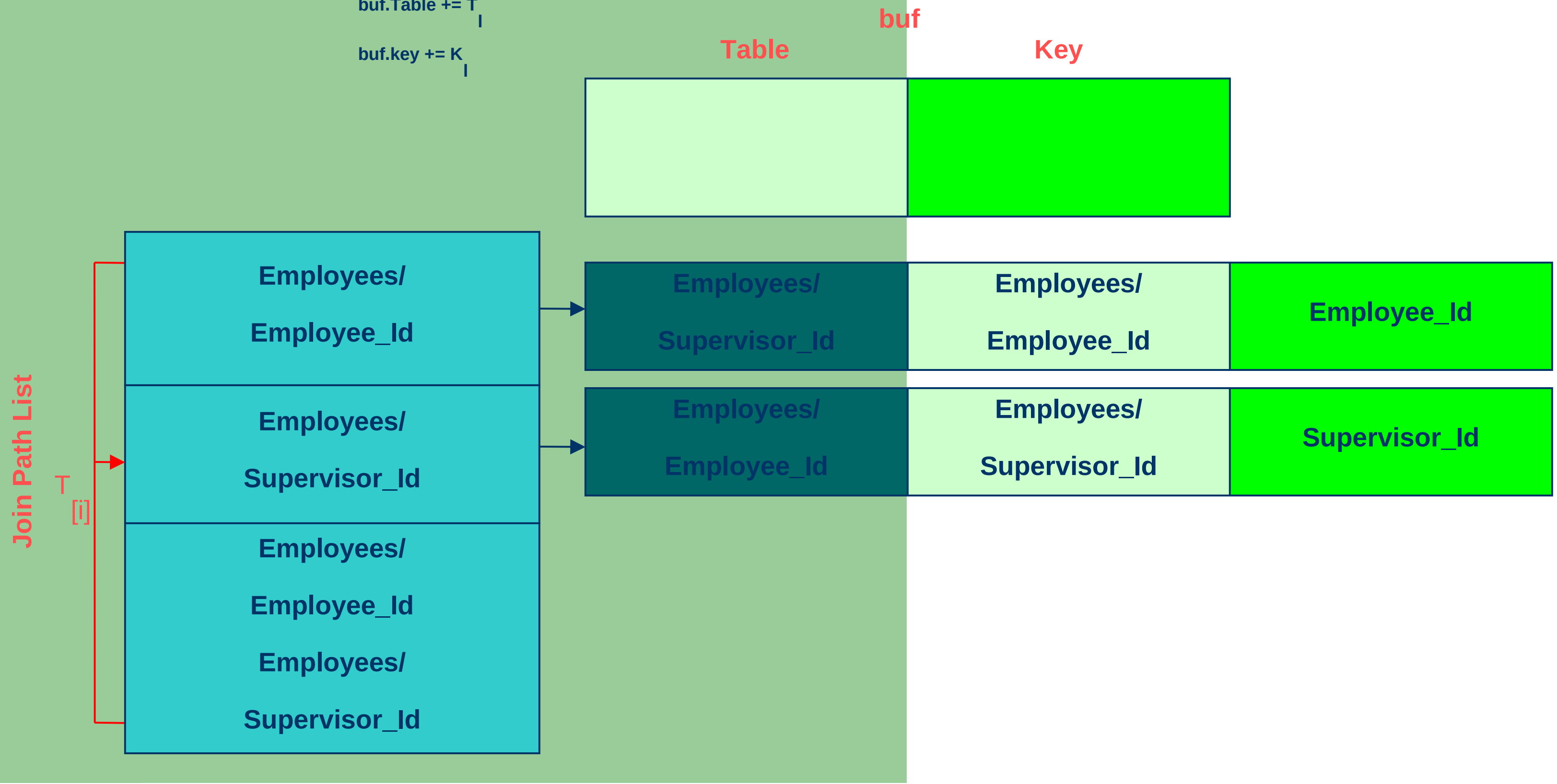
Employees/
Employee_Id

Employees/
Supervisor_Id

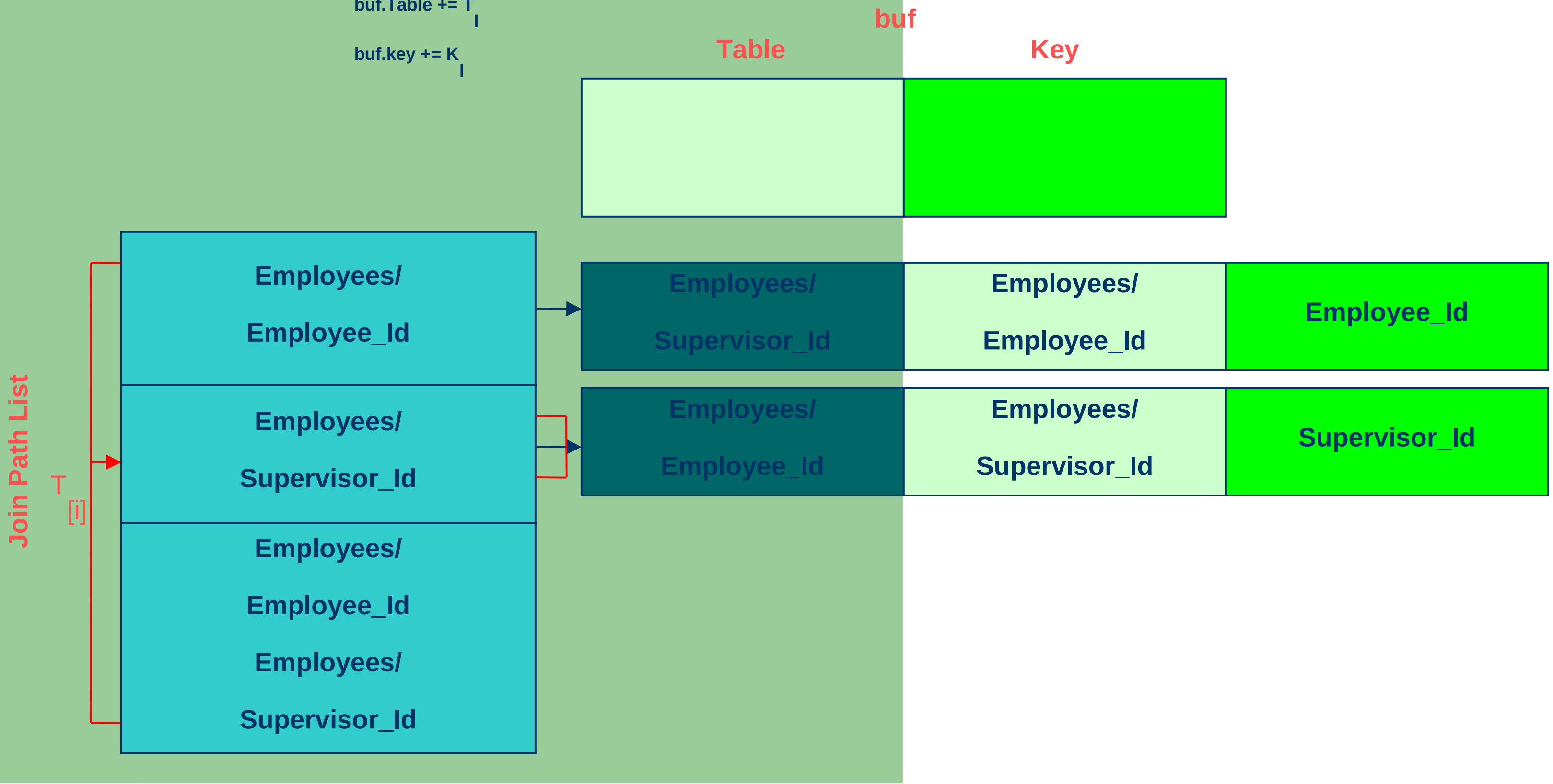
Supervisor_Id

N/A

```
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 all the couples of <Table,key>  in buf do

            take one couple $buf_c$ at a time

            if ($buf_c$.Table = $T_k$) then

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

                    InheritedKey($T_{[i]}$)  += $buf_c$

    if $T_{[i]}$ is not a base table then

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

            buf.Table += $T_l$

            buf.key += $K_l$
```



```
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 all the couples of <Table,key>  in buf do
            take one couple $buf_c$ at a time
            if ($buf_c$.Table = $T_k$) then
                if ($buf_c$ != Key($T_{[i]}$) ) and ($buf_c$ not in InheritedKey($T_{[i]}$)) then
                    InheritedKey($T_{[i]}$)  += $buf_c$
    if $T_{[i]}$ is not a base table then
        if $T_l$ is the table from which comes Key($T_{[i]}$) and $k_l$ is the respective key then
            buf.Table += $T_l$
            buf.key += $K_l$
```



buf

Table

Key

Employees/ Supervisor_Id

Employees/ Employee_Id

Employee_Id

Employees/ Employee_Id

Employees/ Supervisor_Id

Supervisor_Id

Join Path List

$T_{[i]}$

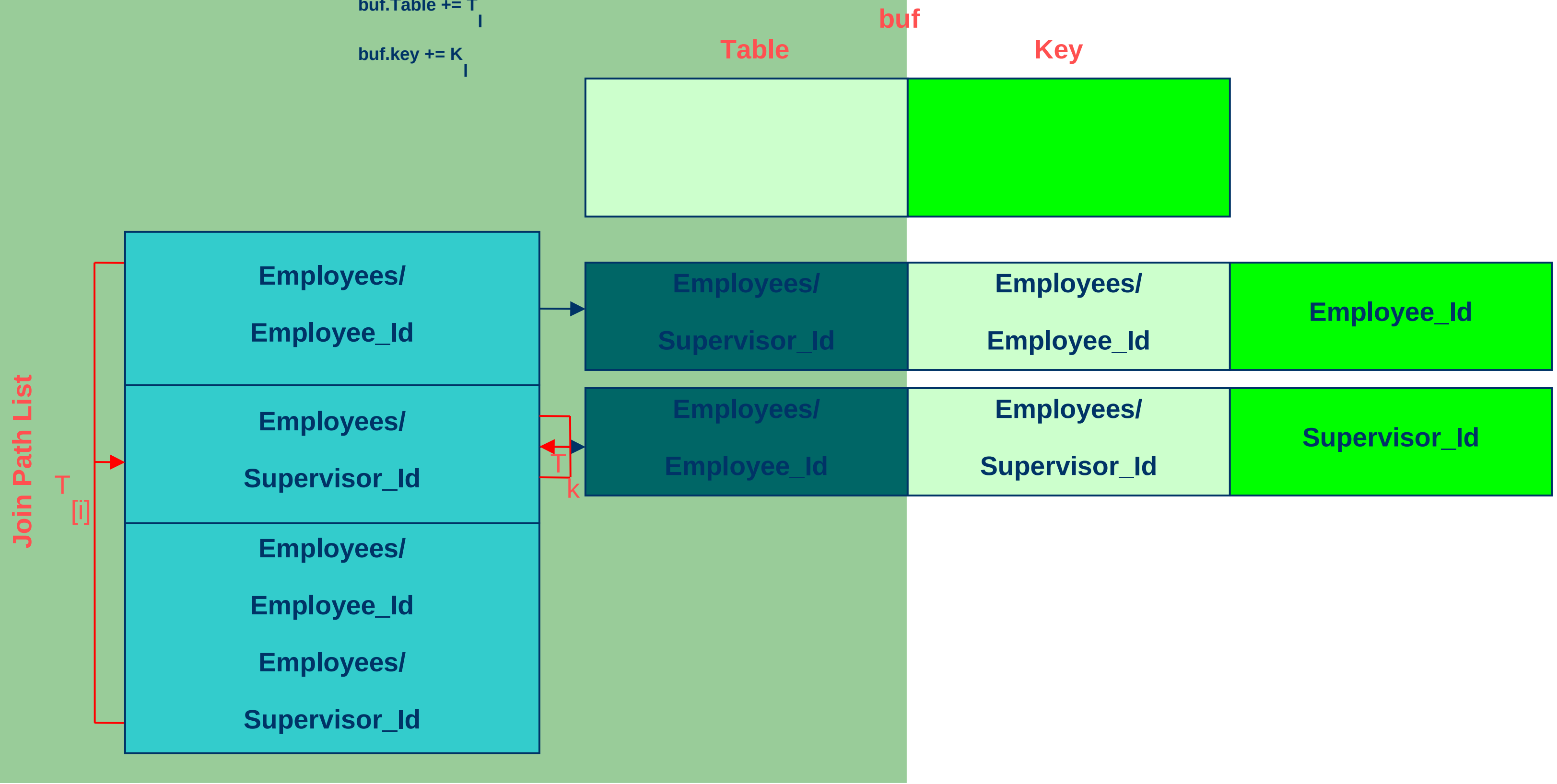
Employees/ Employee_Id

Employees/ Supervisor_Id

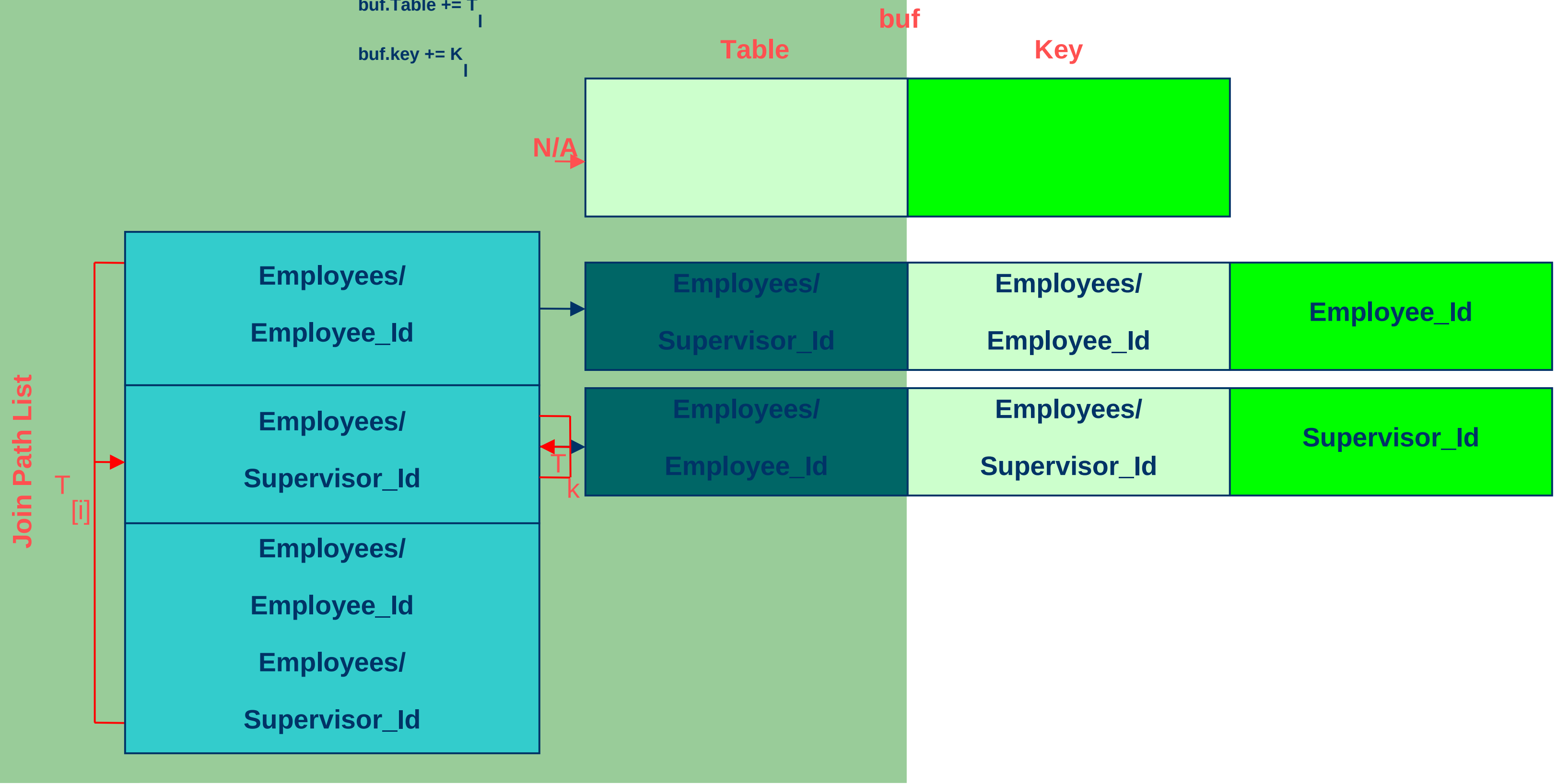
Employees/ Employee_Id

Employees/ Supervisor_Id

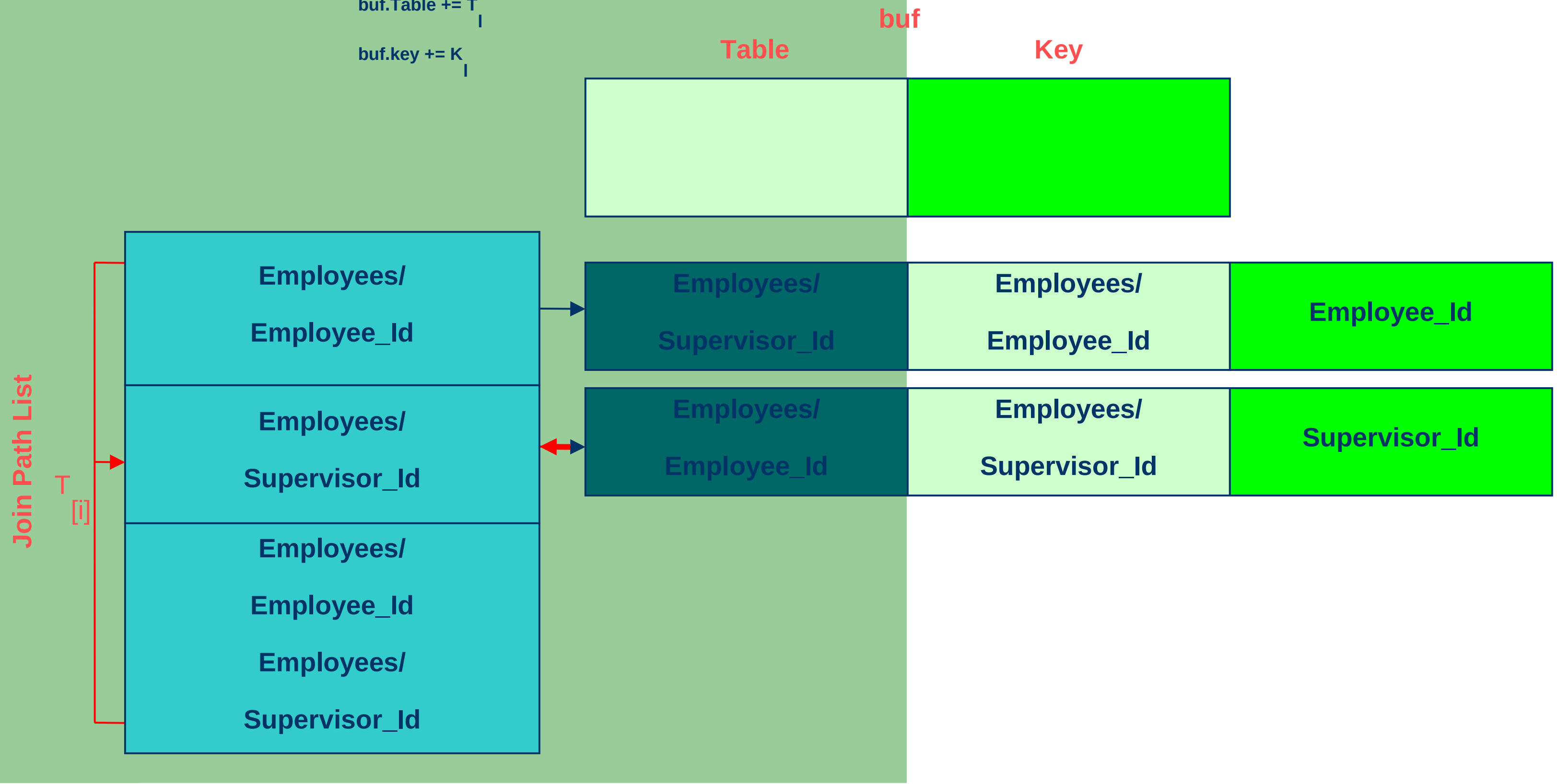
```
create a structure buf with 2 fields: Table and Key

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 all the couples of <Table,key>  in buf do

            take one couple $buf_c$ at a time

            if ($buf_c$.Table = $T_k$) then

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

                    InheritedKey($T_{[i]}$)  += $buf_c$

    if $T_{[i]}$ is not a base table then

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

            buf.Table += $T_l$

            buf.key += $K_l$
```



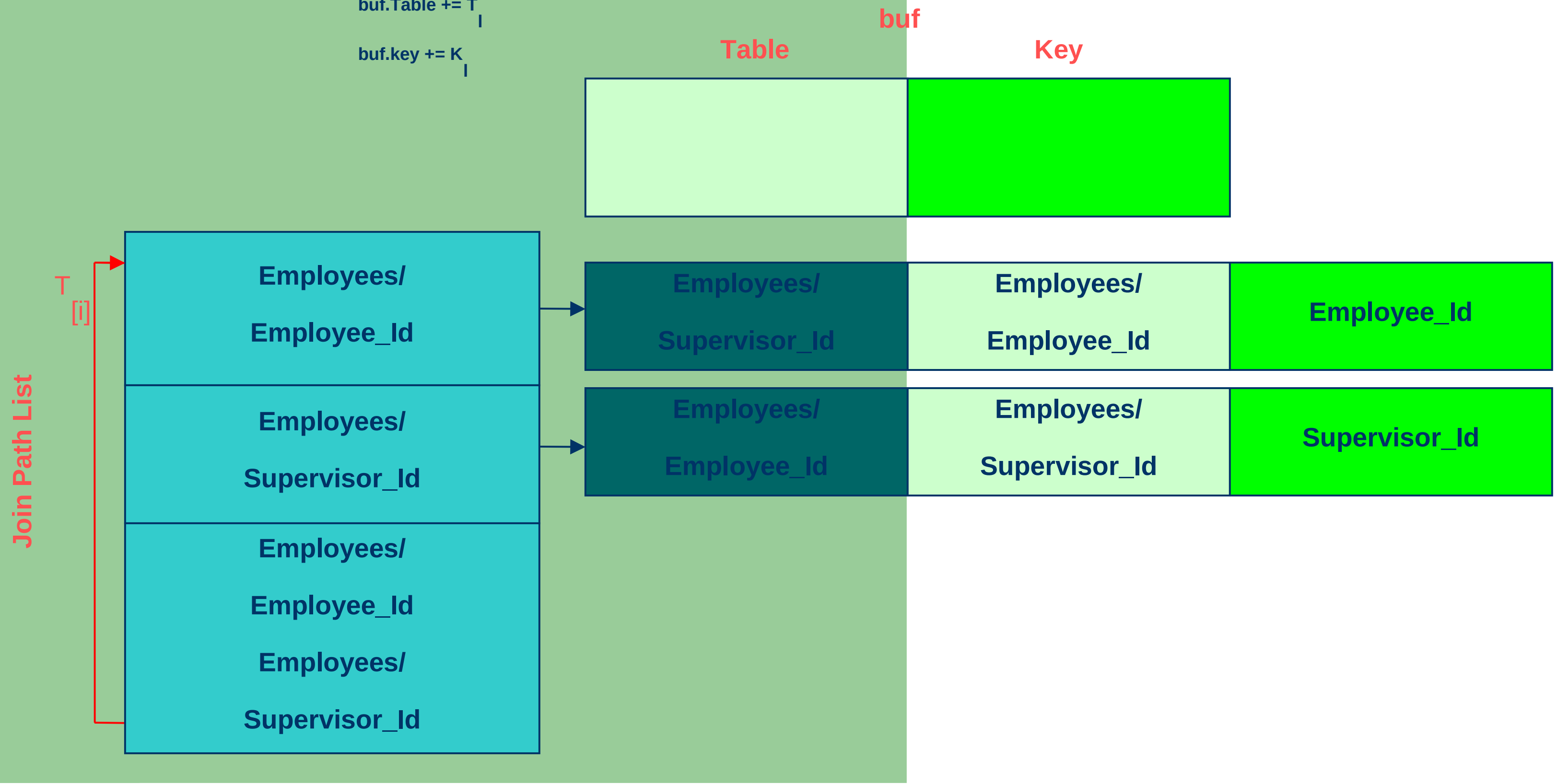
```
create a structure buf with 2 fields: Table and Key
for all the tables in JoinPathList going downward do
    take one $T_{[i]}$ at a time
    for all Base Tables in$T_{[i]}$ do
        take one $T_k$ at a time
        for all the couples of <Table,key> in buf do
            take one couple $buf_c$ at a time
            if ($buf_c$.Table = $T_k$) then
                if ($buf_c$ != Key($T_{[i]}$) ) and ($buf_c$ not in InheritedKey($T_{[i]}$)) then
                    InheritedKey($T_{[i]}$) += $buf_c$
    if $T_{[i]}$ is not a base table then
        if $T_l$ is the table from which comes Key($T_{[i]}$) and $k_l$ is the respective key then
            buf.Table += $T_l$
            buf.key += $K_l$
```



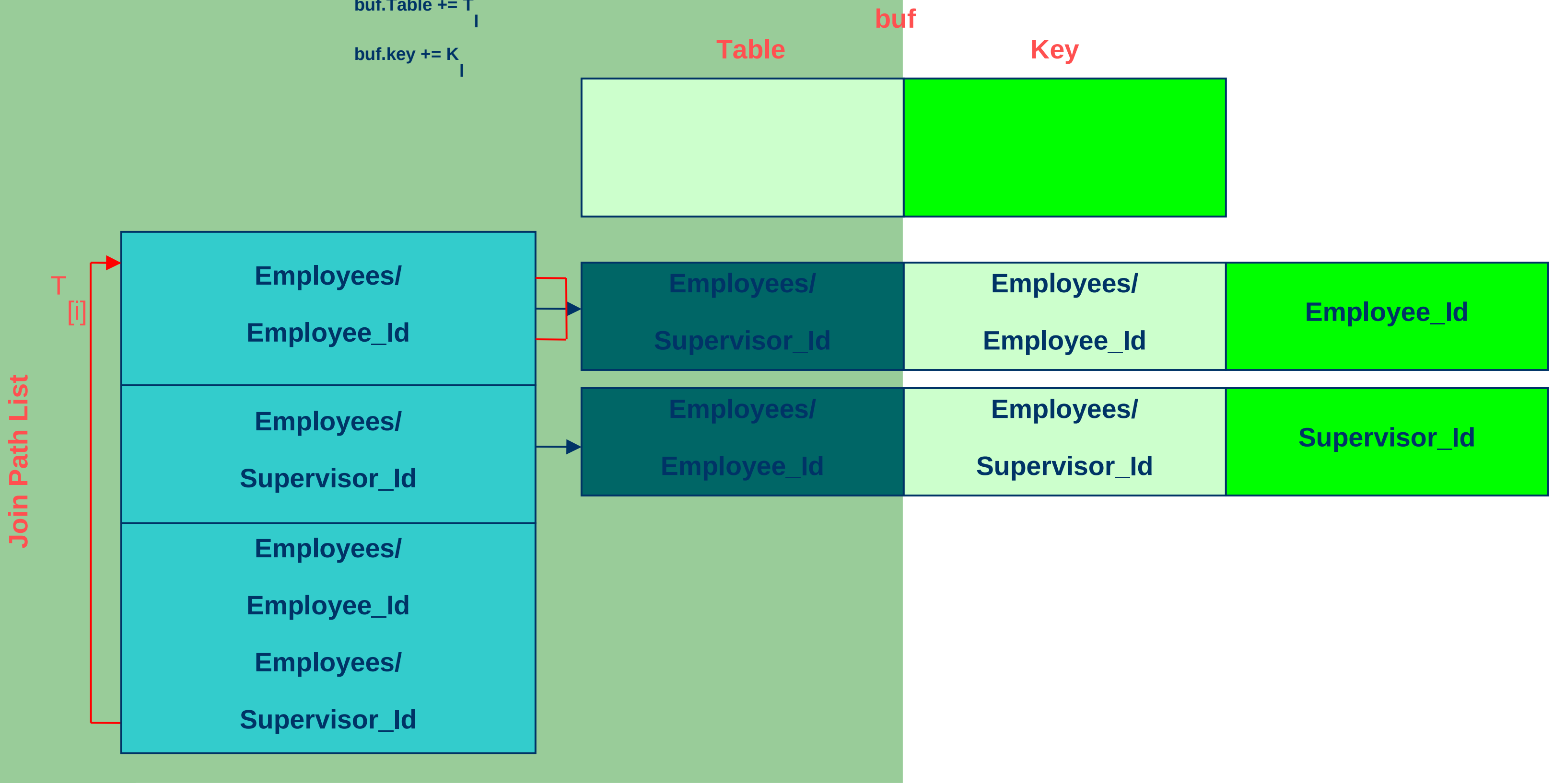
```
create a structure buf with 2 fields: Table and Key
for all the tables in JoinPathList going downward do
    take one $T_{[i]}$ at a time
    for all Base Tables in$T_{[i]}$ do
        take one $T_k$ at a time
        for all the couples of <Table,key> in buf do
            take one couple $buf_c$ at a time
            if ($buf_c$.Table = $T_k$) then
                if ($buf_c$ != Key($T_{[i]}$) ) and ($buf_c$ not in InheritedKey($T_{[i]}$)) then
                    InheritedKey($T_{[i]}$) += $buf_c$
→   if $T_{[i]}$ is not a base table then
        if $T_l$ is the table from which comes Key($T_{[i]}$) and $k_l$ is the respective key then
            buf.Table += $T_l$
            buf.key += $K_l$
```



```
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 all the couples of <Table,key>  in buf do
                        take one couple $buf_c$ at a time
                        if ($buf_c$.Table = $T_k$) then
                                if ($buf_c$ != Key($T_{[i]}$) ) and ($buf_c$ not in InheritedKey($T_{[i]}$)) then
                                    InheritedKey($T_{[i]}$)  += $buf_c$
        if $T_{[i]}$ is not a base table then
                if $T_l$ is the table from which comes Key($T_{[i]}$) and $k_l$ is the respective key then
                        buf.Table += $T_l$
                        buf.key += $K_l$
```



```
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 all the couples of <Table,key>  in buf do

            take one couple $buf_c$ at a time

            if ($buf_c$ .Table = $T_k$ ) then

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

                    InheritedKey($T_{[i]}$ )  += $buf_c$

    if $T_{[i]}$ is not a base table then

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

            buf.Table += $T_l$

            buf.key += $K_l$
```



buf

Table

Key

$T_{[i]}$

Join Path List

Employees/ Employee_Id

Employees/ Supervisor_Id

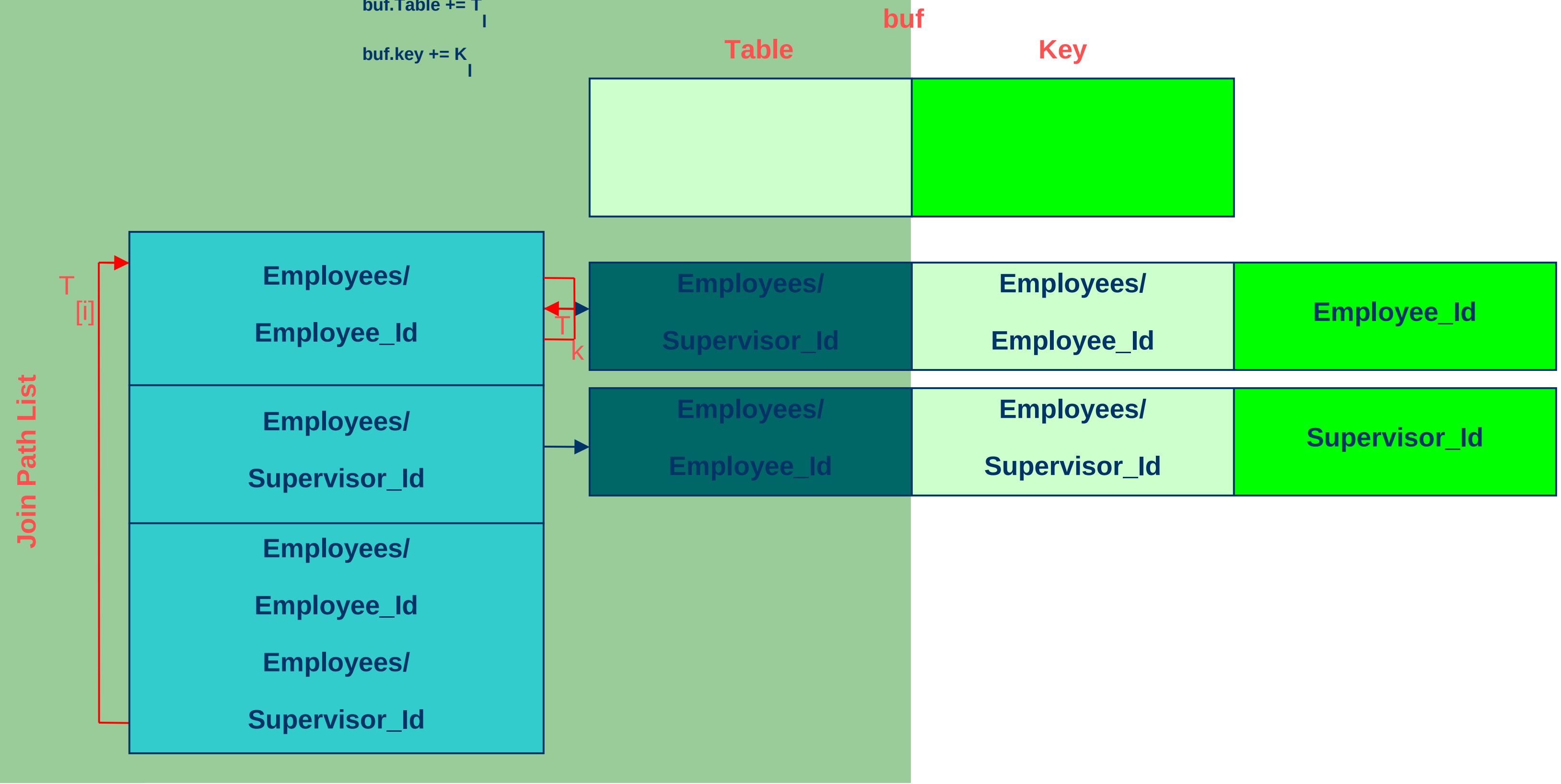
Employees/ Employee_Id

Employees/ Supervisor_Id

Employees/ Supervisor_Id | Employees/ Employee_Id | Employee_Id

Employees/ Employee_Id | Employees/ Supervisor_Id | Supervisor_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 all the couples of <Table,key> in buf do
            take one couple $buf_c$ at a time
            if ($buf_c$.Table = $T_k$) then
                if ($buf_c$ != Key($T_{[i]}$) ) and ($buf_c$ not in InheritedKey($T_{[i]}$)) then
                    InheritedKey($T_{[i]}$) += $buf_c$
    if $T_{[i]}$ is not a base table then
        if $T_l$ is the table from which comes Key($T_{[i]}$) and $k_l$ is the respective key then
            buf.Table += $T_l$
            buf.key += $K_l$
```



buf

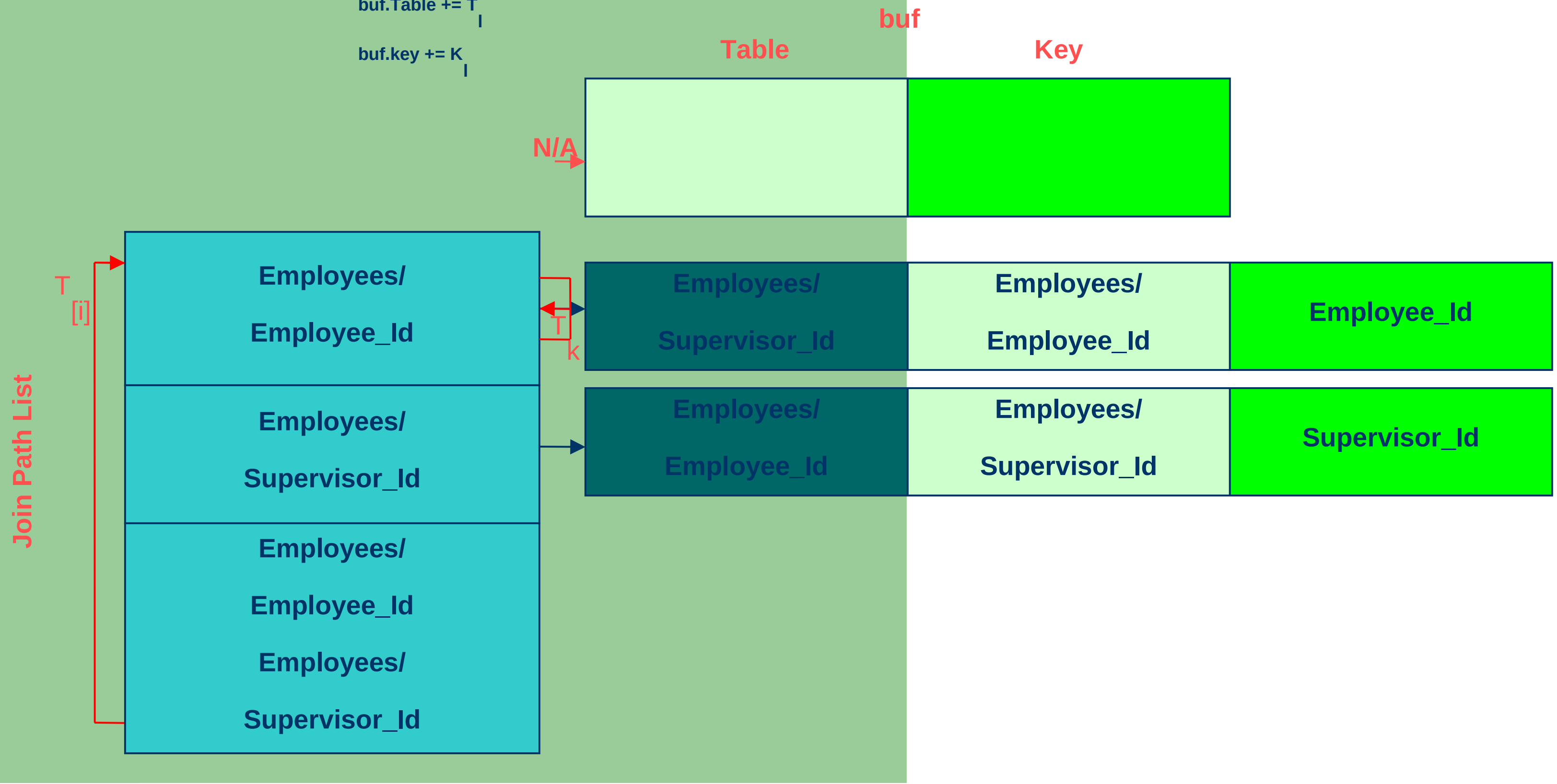
Table

Key

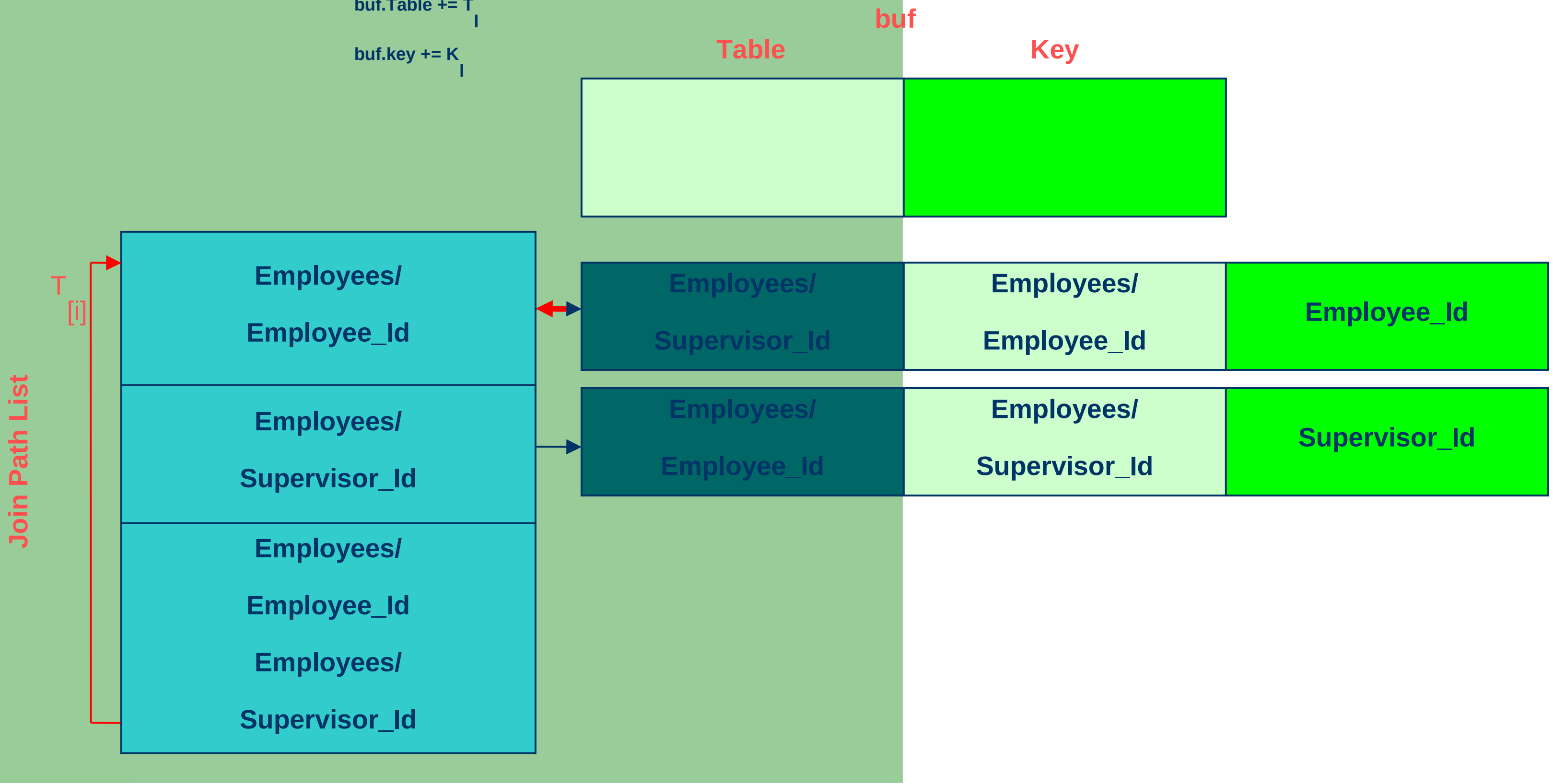
Employees/ Supervisor_Id | Employees/ Employee_Id | Employee_Id

Employees/ Employee_Id | Employees/ Supervisor_Id | Supervisor_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 all the couples of <Table,key> in buf do
            take one couple $buf_c$ at a time
            if ($buf_c$.Table = $T_k$) then
                if ($buf_c$ != Key($T_{[i]}$) ) and ($buf_c$ not in InheritedKey($T_{[i]}$)) then
                    InheritedKey($T_{[i]}$) += $buf_c$
    if $T_{[i]}$ is not a base table then
        if $T_l$ is the table from which comes Key($T_{[i]}$) and $k_l$ is the respective key then
            buf.Table += $T_l$
            buf.key += $K_l$
```



```
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 all the couples of <Table,key>  in buf do
            take one couple $buf_c$ at a time
            if ($buf_c$.Table = $T_k$) then
                if ($buf_c$ != Key($T_{[i]}$) ) and ($buf_c$ not in InheritedKey($T_{[i]}$)) then
                    InheritedKey($T_{[i]}$)  += $buf_c$
    if $T_{[i]}$ is not a base table then
        if $T_l$ is the table from which comes Key($T_{[i]}$) and $k_l$ is the respective key then
            buf.Table += $T_l$
            buf.key += $K_l$
```



buf

Table

Key

$T_{[i]}$

Join Path List

Employees/ Employee_Id

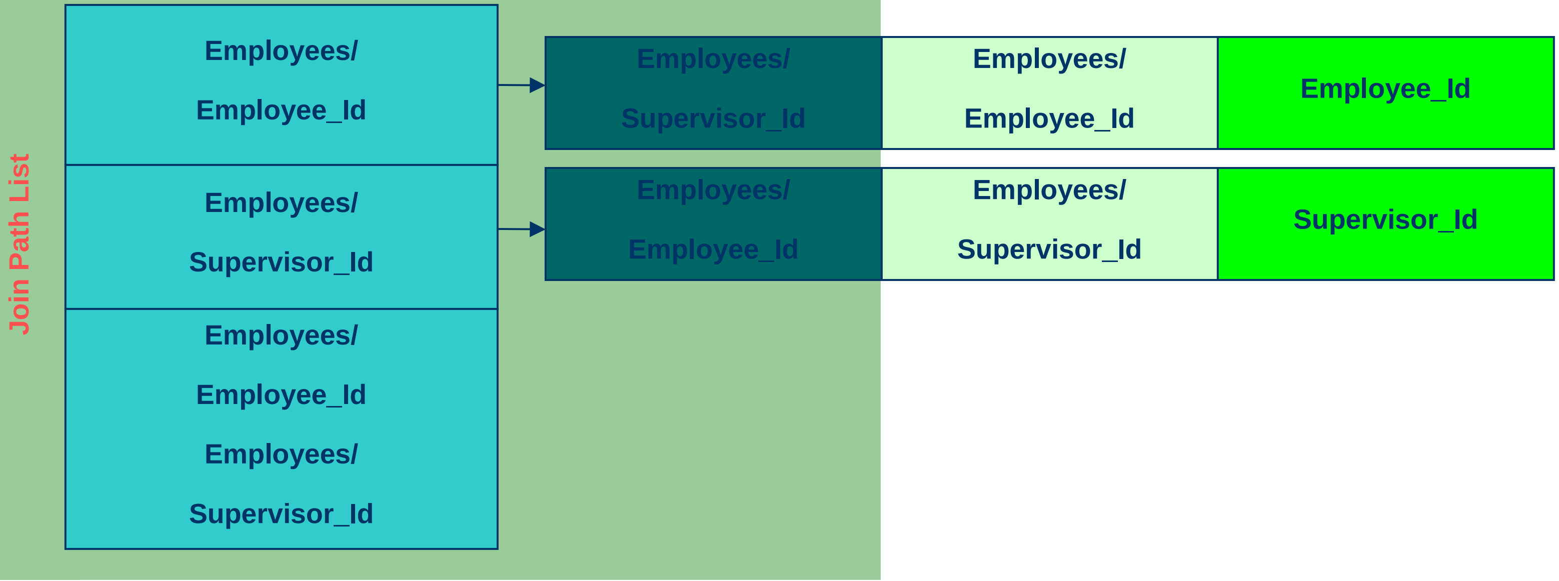
Employees/ Supervisor_Id

Employees/ Employee_Id

Employees/ Supervisor_Id

Employees/ Supervisor_Id	Employees/ Employee_Id	Employee_Id
Employees/ Employee_Id	Employees/ Supervisor_Id	Supervisor_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 all the couples of <Table,key>  in buf do
            take one couple $buf_c$ at a time
            if ($buf_c$.Table = $T_k$) then
                if ($buf_c$ != Key($T_{[i]}$) ) and ($buf_c$ not in InheritedKey($T_{[i]}$)) then
                    InheritedKey($T_{[i]}$)  += $buf_c$
    if $T_{[i]}$ is not a base table then
        if $T_l$ is the table from which comes Key($T_{[i]}$) and $k_l$ is the respective key then
            buf.Table += $T_l$
            buf.key += $K_l$
```



Join Path List

Employees/ Employee_Id

Employees/ Supervisor_Id

Employees/ Employee_Id
Employees/ Supervisor_Id

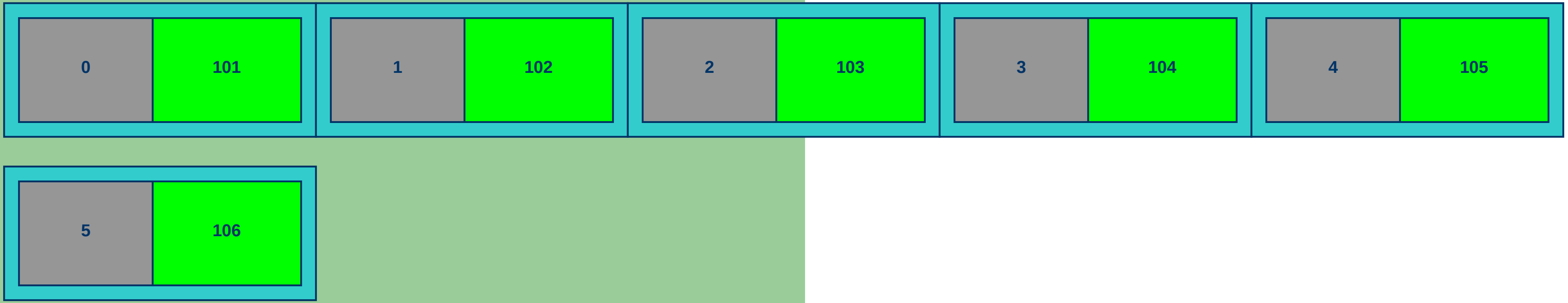
Employees/ Supervisor_Id	Employees/ Employee_Id	Employee_Id
Employees/ Employee_Id	Employees/ Supervisor_Id	Supervisor_Id

Employees table

	EMPLOYEE_ID	NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	SUPERVISOR_ID	DEPARTMENT_ID
0	101	Mark Stench	mstench	233-4268	12/02/1998	FI_MGR	60000	106	FIN
1	102	Jorge Perez	jperez	448-5268	05/14/1999	AC_MGR	60000	106	ACC
2	103	Edward Cartier	ecartier	742-8429	03/01/2003	SA_MGR	60000	106	SAL
3	104	Teresa Gonzalez	tgonzalez	134-8329	12/20/2002	AC_AUD	55000	102	ACC
4	105	Michelle Blanche	mblanche	745-7496	01/02/2001	SA_REP	35000	103	SAL
5	106	Peter Spencer	pspencer	111-2222	01/01/1996	GE_MGR	120000	NULL	GEN

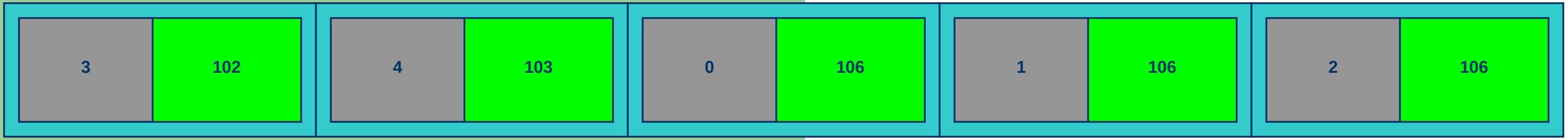
Applying the insert routine for all the rows in the table EMPLOYEES, we obtain the following indexes:

B+Tree(T_0)



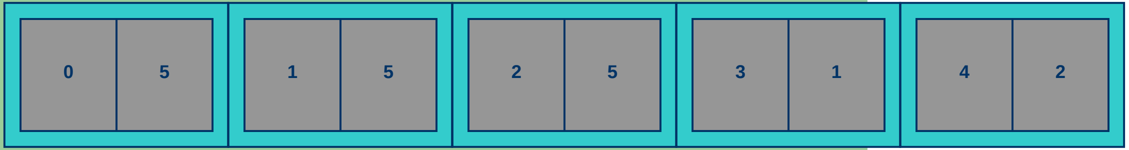
0	101
1	102
2	103
3	104
4	105
5	106

B+Tree(T_1)



3	102
4	103
0	106
1	106
2	106

B+Tree(T_2)



0	5
1	5
2	5
3	1
4	2

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.