

# B<sup>+</sup>Tree

**A mechanism to drive existing B<sup>+</sup>Trees to do Join Internally.**

# RDBMS challenges

Performance has been always a challenge for relational databases. A major problem with relational database that a good schema requires many tables in relation between them, and in consequence the calculation of many joins to satisfy the queries.

DBA spends lots of time to tune the database. Some database schemas are not in BCNF just to avoid some joins. Also some materialized views are just to avoid it. Star Schema born from the fact that joining is complex and to simplify joins.

# Standard indexes

The standard indexes used in relational databases are B<sup>+</sup>tree, hashed keys and bitmap indexes but the problem all suffers from some restrictions.

B<sup>+</sup>tree have among others the fact that it work just on one table.

Hashed keys are very fast but they require a full key lookup, a perfect match, and a unique identifying value.

Bitmap is also good but it has a limit on the number of different values a column can have.

# Bitmap Join Index & Materialized Views

Bitmap join index is efficient but is not general; it is based on Star Schema. Has a lots of bitmap arrays depending on the size of the dimensions tables.

Materialized Views are redundant. No one use Materialized View to order a table, because index is the more natural way. Also B<sup>+</sup>Tree index is the more natural way to get the join.

# B<sup>⋈</sup>TREE Overview

B<sup>⋈</sup>tree is a new index technology that is based on B<sup>+</sup>tree that pre-join the tables inside it.

B<sup>⋈</sup>tree uses “Virtual Tables” and “Join Path Lists” to make pre-join internally, so it doesn't use the multidimensional index technique with the benefit of more easier and more concise algorithm, no limit for the number of tables in join and easy to use: the same way as a native B<sup>+</sup>tree.

Given n Tables in join, scanning the B<sup>⋈</sup>tree return a set of pointers for the rows in join for any possible combination of tables in join.

# B $\bowtie$ TREE Index

To understand how B $\bowtie$ 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  $R_n \dots R_z$  in  $T_k$  that satisfy the join condition with  $R_m$  and insert Rows references to  $R_m R_n \dots R_m R_z$  in the virtual table  $T_{ik}$ .

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

# Transformation of Existing B<sup>+</sup>Tree

- **The internal definition for the creation of a B<sup>+</sup>Tree take in consideration the following:**
  - Name of B<sup>+</sup>Tree index follow by an index
  - Number of base tables constituting the virtual table indexed by the B<sup>+</sup>Tree
  - Length and type of Keys
  - Length and type of Inherited Keys (They are supplementary fields inserted in the B<sup>+</sup>Tree but they are not part of the key and they are not used for comparison)
- **Declare the page of B<sup>+</sup>Tree as a buffer of bytes and divide it as needed. Many existing B<sup>+</sup>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<sup>+</sup>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 all the keys in the page
- **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<sup>+</sup>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.**



# Transformation of Existing B+Tree (continue)

Many advanced B+Tree in the market use

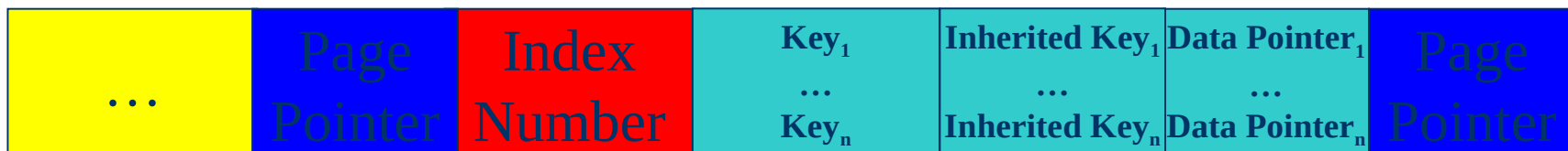
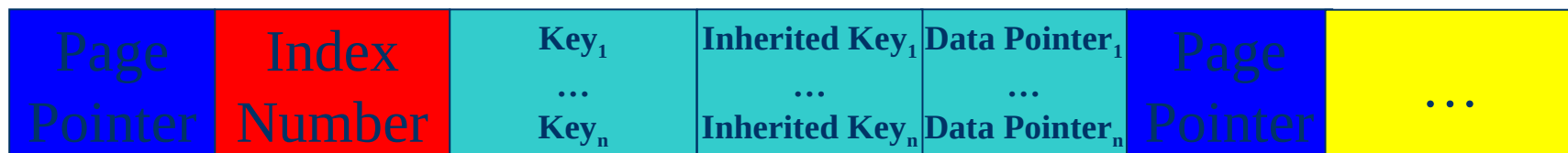
(Key, Data Reference) combination to refer to unique Row eliminating duplicates internally and use additional fields others than the one forming the key to avoid access to the table.

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

B-Tree page structure



B<sup>+</sup>Tree page structure



# 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).**

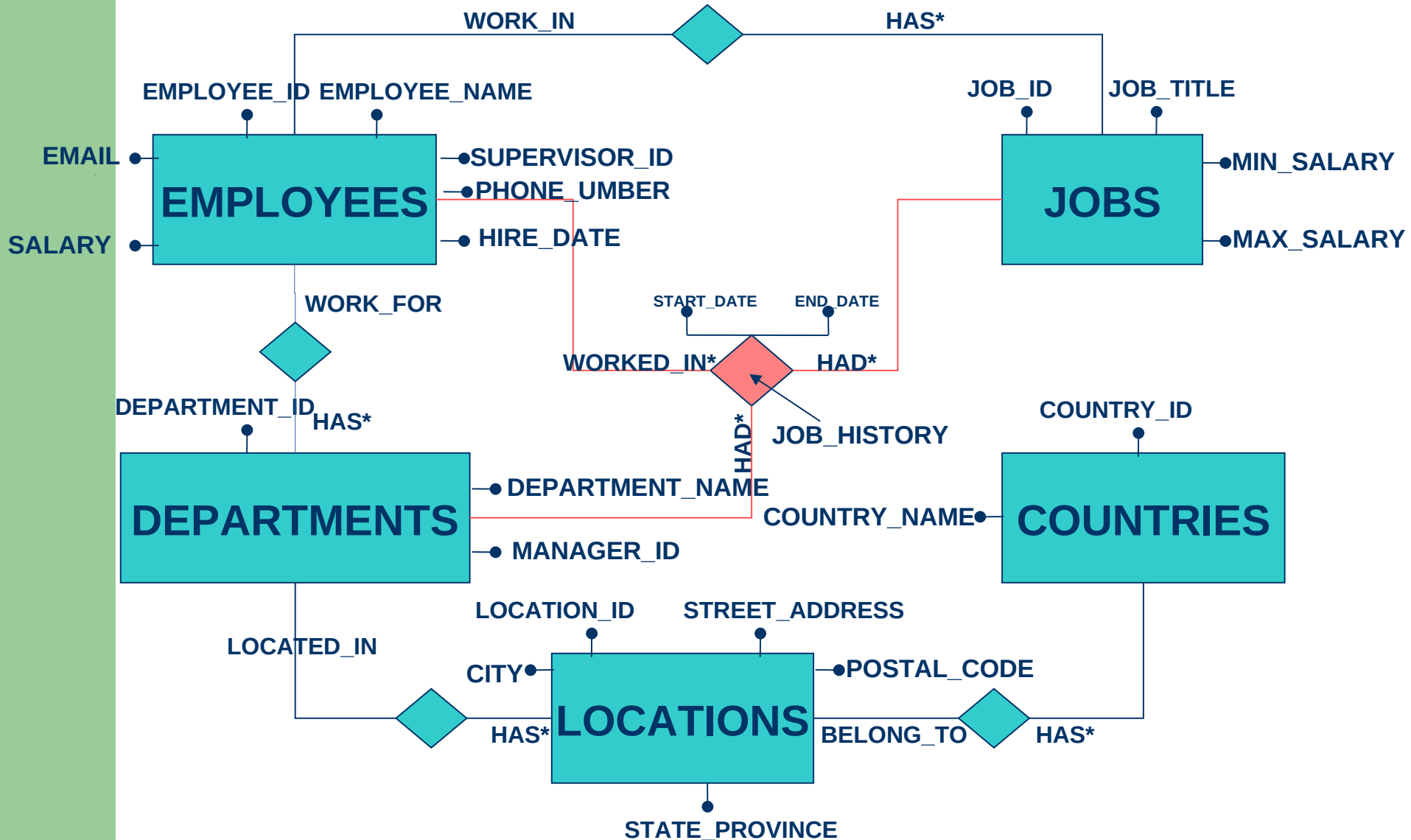
Adjacency List:

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

Join Graph:

**A graph representing direct join between tables.**

# Example:



Transforming the entity relationship schema into the relational model, we get the following tables:

**CREATE TABLE EMPLOYEES**

```
(  
    EMPLOYEE_ID INT NOT NULL,  
    EMPLOYEE_NAME VARCHAR(35),  
    EMAIL VARCHAR(25),  
    PHONE_NUMBER VARCHAR(20),  
    HIRE_DATE DATE,  
    SUPERVISOR_ID INT NOT NULL,  
    JOB_ID VARCHAR(10),  
    SALARY NUMERIC(8,2),  
    DEPARTMENT_ID VARCHAR(3)  
);
```

**CREATE TABLE JOB\_HISTORY**

```
(  
    EMPLOYEE_ID INT,  
    START_DATE DATE,  
    END_DATE DATE,  
    DEPARTMENT_ID VARCHAR(3),  
    JOB_ID VARCHAR(10)  
);
```

**CREATE TABLE JOBS**

```
(  
    JOB_ID VARCHAR(10),  
    JOB_TITLE VARCHAR(35),  
    MIN_SALARY DOUBLE,  
    MAX_SALARY DOUBLE  
);
```

**CREATE TABLE DEPARTMENTS**

```
(  
    DEPARTMENT_ID VARCHAR(3),  
    DEPARTMENT_NAME VARCHAR(30),  
    MANAGER_ID INT,  
    LOCATION_ID INT  
);
```

**CREATE TABLE LOCATIONS**

```
(  
    LOCATION_ID INT,  
    STREET_ADDRESS VARCHAR(40),  
    POSTAL_CODE VARCHAR(12),  
    CITY VARCHAR(30),  
    STATE_PROVINCE VARCHAR(25),  
    COUNTRY_ID CHAR(2)  
);
```

**CREATE TABLE COUNTRIES**

```
(  
    COUNTRY_ID CHAR(2),  
    COUNTRY_NAME VARCHAR(40)  
);
```

The join (example A) going step by step over, is the following:

List where every employee have been worked before along with the department that he is working now:

```
CREATE JOIN INDEX RECENT_IDX
```

```
ON      EMPLOYEES(LAST_NAME), DEPARTMENTS, JOBS, JOB_HISTORY, LOCATIONS, COUNTRIES
WHERE  EMPLOYEES.EMPLOYEE_ID = JOB_HISTORY.EMPLOYEE_ID
      AND JOBS.JOB_ID = JOB_HISTORY.JOB_ID
      AND EMPLOYEES.DEPARTMENT_ID = DEPARTMENTS.DEPARTMENT_ID
      AND DEPARTMENTS.LOCATION_ID = LOCATIONS.LOCATION_ID
      AND LOCATIONS.COUNTRY_ID = COUNTRIES.COUNTRY_ID;
```

```
SELECT EMPLOYEES.EMPLOYEE_NAME,
        JOBS.JOB_TITLE AS JOB_TITLE,
        DEPARTMENTS.DEPARTMENT_NAME AS DEPARTMENT_NAME,
        COUNTRIES.COUNTRY_NAME AS COUNTRY_NAME
FROM    EMPLOYEES, JOB_HISTORY, DEPARTMENTS,
        LOCATIONS, JOBS, COUNTRIES
WHERE  EMPLOYEES.EMPLOYEE_ID = JOB_HISTORY.EMPLOYEE_ID
      AND JOBS.JOB_ID = JOB_HISTORY.JOB_ID
      AND EMPLOYEES.DEPARTMENT_ID = DEPARTMENTS.DEPARTMENT_ID
      AND DEPARTMENTS.LOCATION_ID = LOCATIONS.LOCATION_ID
      AND LOCATIONS.COUNTRY_ID = COUNTRIES.COUNTRY_ID;
```

## Example A

This is another join (example B) on the same tables:

List where every employee have been worked before along with the department that he is working before:

```
CREATE JOIN INDEX HISTORY_IDX
```

```
ON      EMPLOYEES(LAST_NAME), DEPARTMENTS, JOBS, JOB_HISTORY, LOCATIONS, COUNTRIES
WHERE  EMPLOYEES.EMPLOYEE_ID = JOB_HISTORY.EMPLOYEE_ID
      AND JOBS.JOB_ID = JOB_HISTORY.JOB_ID
      AND JOB_HISTORY.DEPARTMENT_ID = DEPARTMENTS.DEPARTMENT_ID
      AND DEPARTMENTS.LOCATION_ID = LOCATIONS.LOCATION_ID
      AND LOCATIONS.COUNTRY_ID = COUNTRIES.COUNTRY_ID;
```

```
SELECT EMPLOYEES.EMPLOYEE_NAME, Example B
      JOBS.JOB_TITLE AS JOB_TITLE,
      DEPARTMENTS.DEPARTMENT_NAME AS DEPARTMENT_NAME,
      COUNTRIES.COUNTRY_NAME AS COUNTRY_NAME
FROM   EMPLOYEES, JOB_HISTORY, DEPARTMENTS,
      LOCATIONS, JOBS, COUNTRIES
WHERE  EMPLOYEES.EMPLOYEE_ID = JOB_HISTORY.EMPLOYEE_ID
      AND JOBS.JOB_ID = JOB_HISTORY.JOB_ID
      AND JOB_HISTORY.DEPARTMENT_ID = DEPARTMENTS.DEPARTMENT_ID
      AND DEPARTMENTS.LOCATION_ID = LOCATIONS.LOCATION_ID
      AND LOCATIONS.COUNTRY_ID = COUNTRIES.COUNTRY_ID;
```



# 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 know which tables are in direct join with other 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

This function is different from the one in BjoinTree.pas because this one is simplified for the example and the other is general.

→ 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

Employees	→ Job_History Employee_Id
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	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

→  $\text{AdjacentList}[T_i] += T_k$  follow by the common key

$\text{AdjacentList}[T_k] += T_i$  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				
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				
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

→  $\text{AdjacentList}[T_i] += T_k$  follow by the common key

$\text{AdjacentList}[T_k] += T_i$  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

→  $\text{AdjacentList}[T_i] += T_k$  follow by the common key

$\text{AdjacentList}[T_k] += T_i$  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)

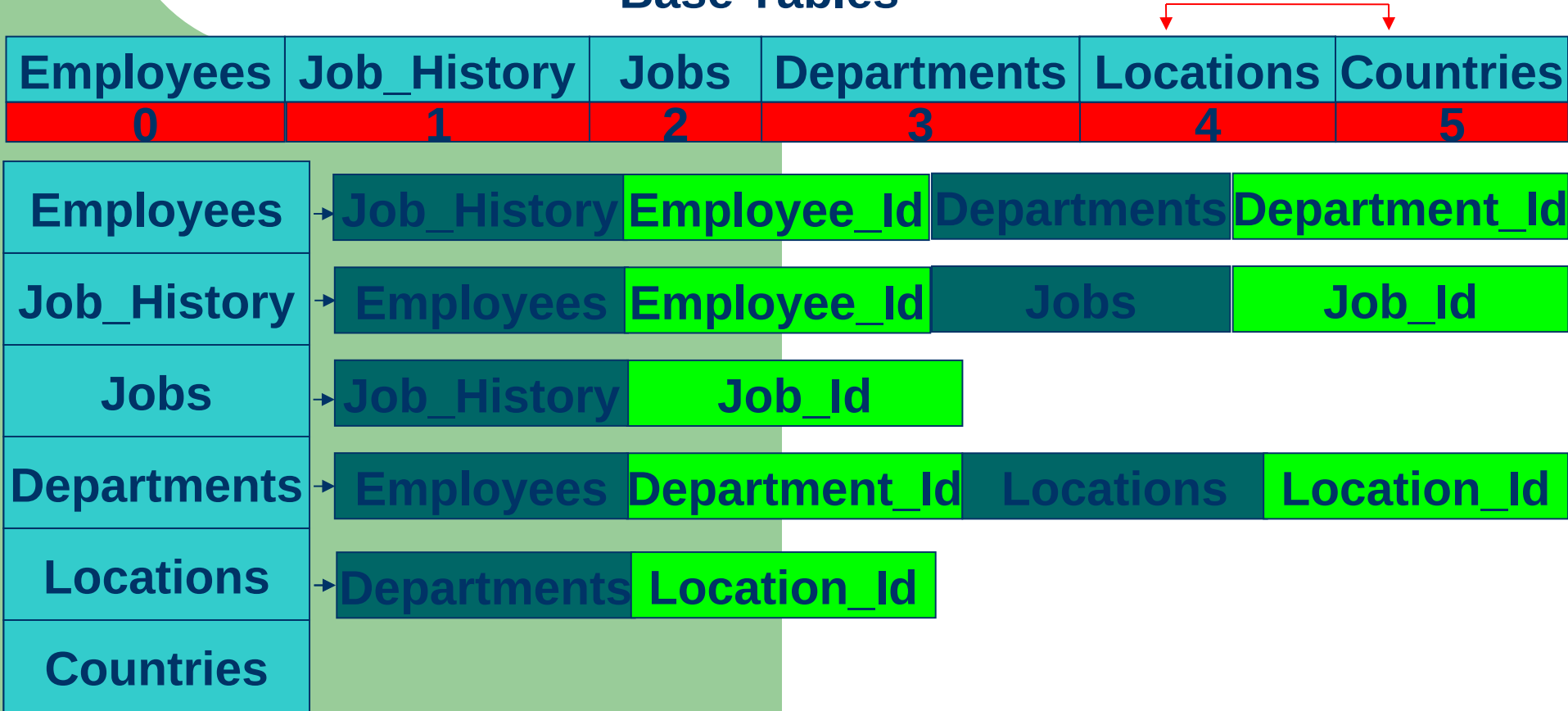
insert the base tables as vertexes of the graph

→ 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



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

→  $\text{AdjacentList}[T_i] += T_k$  follow by the common key

$\text{AdjacentList}[T_k] += T_i$  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

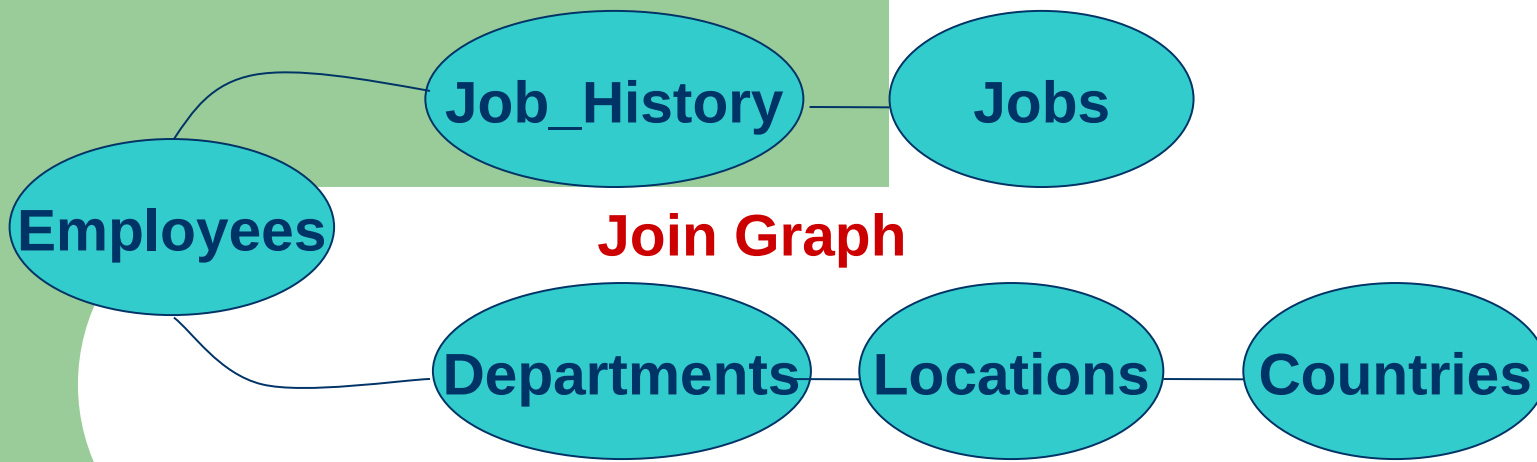
for every direct join between 2 tables of the form  $T_i$  and  $T_k$  where  $T_i$  is the table of order  $i$  and  $T_k$  is the table of order  $k$  as defined by the DBA do

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			

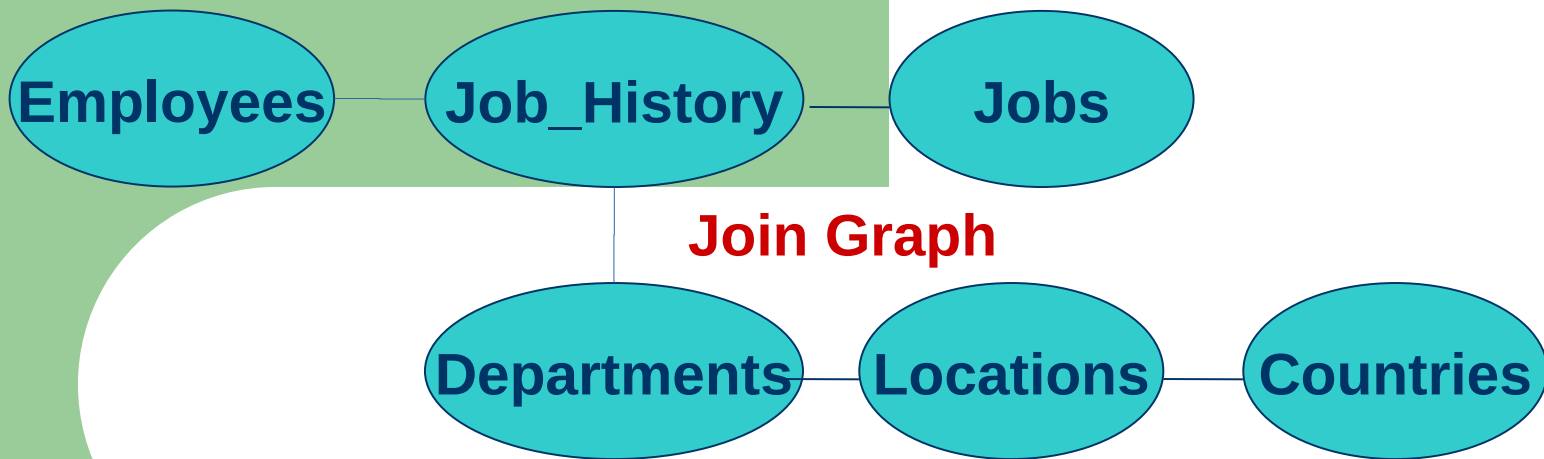


**Example A**

**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**



**Join Graph**

### Example B

**Linked List representation of the Join Graph**

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

**Adjacency List**

# Definition

Join Path List:

**A sequence of tables  $T_0 \dots T_{n-1}$  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: `getFirstAdjacentListKey(T[j], T[k])`

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

    Take one at a time

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

        Take one at a time

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

            return(key( $T_i, T_{\text{Link}(l)}$ ))

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 $T_0 \dots T_m$

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

$T_{\text{Element}}$  = First Table in queue

    for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  from queue

until queue is empty

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

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

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one  $T_i$  at a time

$\text{PathJoinAdjacentList}(T_i) = T_{[\text{buf}]}$

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

$\text{PathJoinAdjacentList}(T_{[\text{buf}]}) = T_i$

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

$T_{[\text{buf}]} \oplus = T_i$

$\text{Insert NodeList}[T_{[\text{buf}]}] = T_{[\text{buf}]}$

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

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

→ 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

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  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  $T_0 \dots T_m$  be the base tables

create 2 dynamic arrays queue and path

insert  $T_0$  into path

insert  $T_0$  into queue

repeat

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  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  $T_0 \dots T_m$  be the base tables

→ create 2 dynamic arrays queue and path

insert  $T_0$  into path

insert  $T_0$  into queue

repeat

$T_{\text{Element}}$  = First Table in queue

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

if the Link Item is in the join sequence then

if path doesn't contain the Link Item then

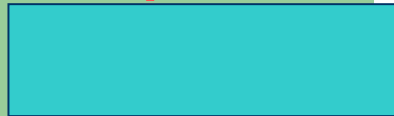
insert Link Item into path

insert Link Item into queue

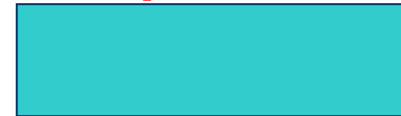
remove  $T_{\text{Element}}$  from queue

until queue is empty

queue



path



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

$T_{\text{Element}}$  = First Table in queue

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

if the Link Item is in the join sequence then

if path doesn't contain the Link Item then

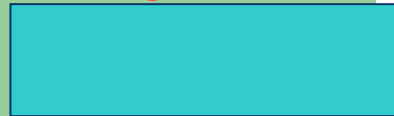
insert Link Item into path

insert Link Item into queue

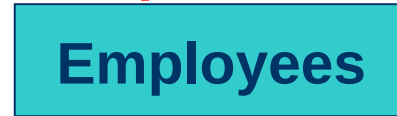
remove  $T_{\text{Element}}$  from queue

until queue is empty

queue



path



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

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  from queue

until queue is empty

queue

Employees

path

Employees



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

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  from queue

until queue is empty

queue

Employees

path

Employees

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

→  $T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  from queue

until queue is empty

queue



Employees

path

Employees

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

$T_{\text{Element}}$  = First Table in queue

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

if the Link Item is in the join sequence then

if path doesn't contain the Link Item then

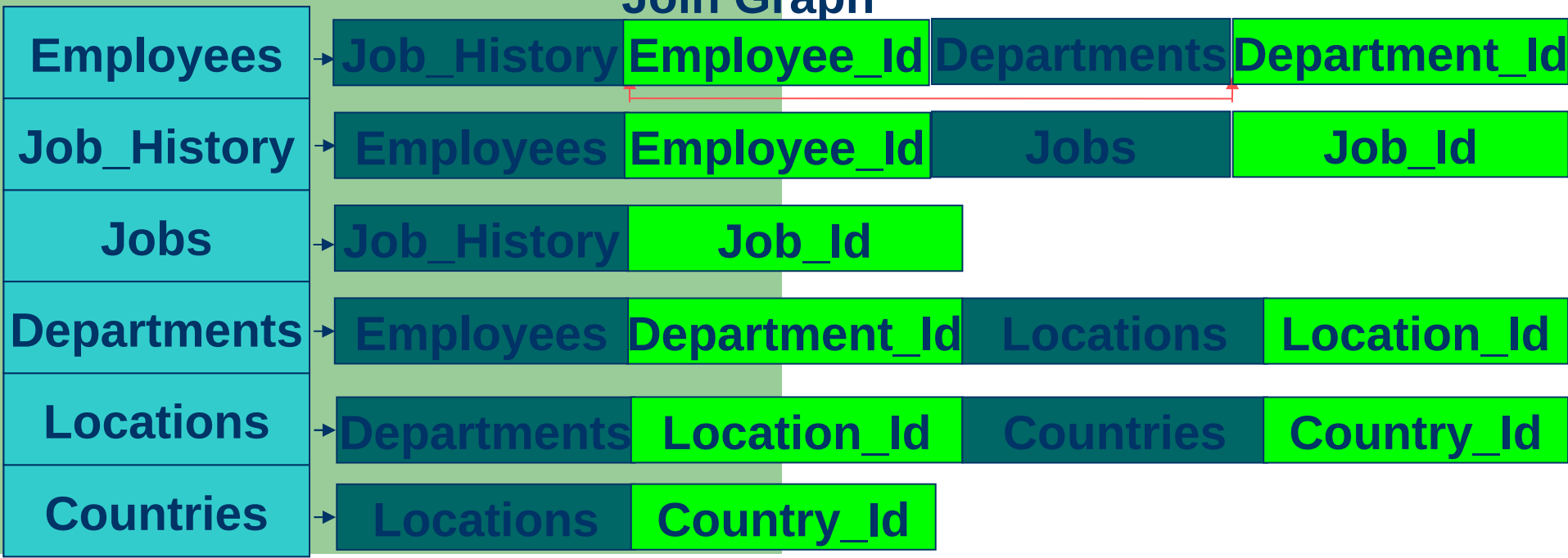
insert Link Item into path

insert Link Item into queue

remove  $T_{\text{Element}}$  from queue

until queue is empty

## Join Graph



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

$T_{\text{Element}}$  = First Table in queue

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

if the Link Item is in the join sequence then

if path doesn't contain the Link Item then

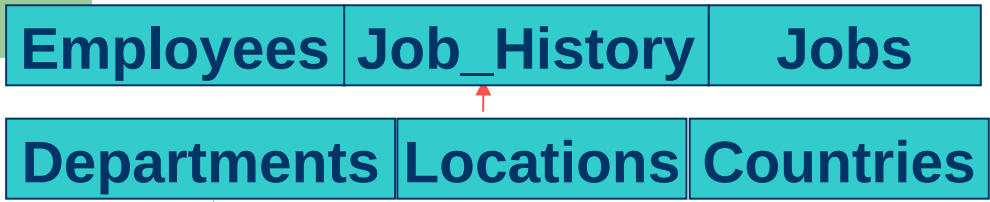
insert Link Item into path

insert Link Item into queue

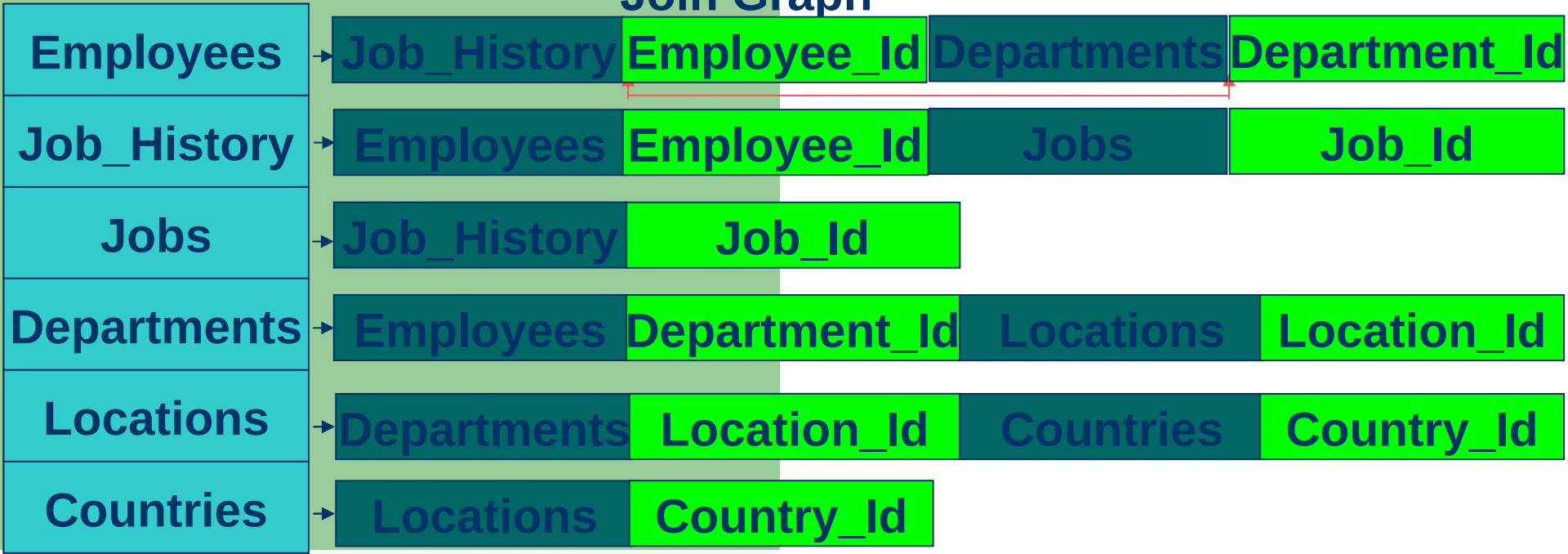
remove  $T_{\text{Element}}$  from queue

until queue is empty

## Join Base Tables



## Join Graph



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

$T_{\text{Element}}$  = First Table in queue

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

if the Link Item is in the join sequence then

if path doesn't contain the Link Item then

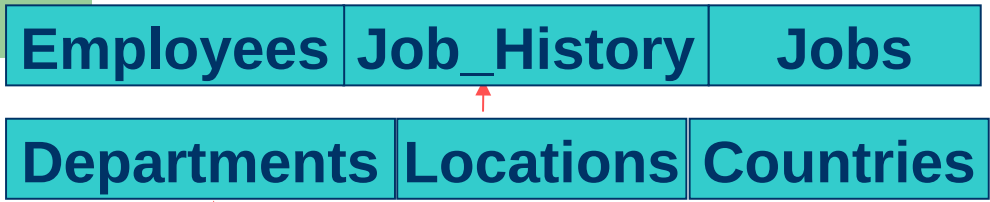
insert Link Item into path

insert Link Item into queue

remove  $T_{\text{Element}}$  from queue

until queue is empty

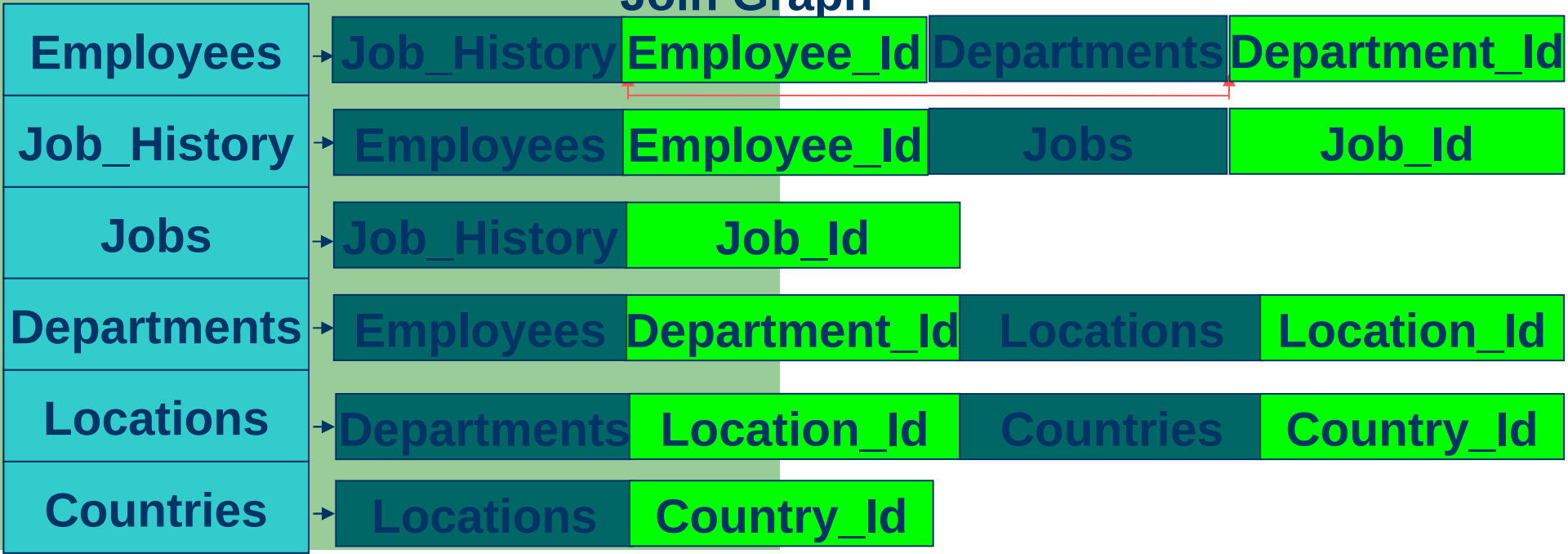
## Join Base Tables



path



## Join Graph



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

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  from queue

until queue is empty

queue



Employees

path

Employees

Job\_History

Departments

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

let  $T_0 \dots T_m$  be the base tables

create 2 dynamic arrays queue and path

insert  $T_0$  into path

insert  $T_0$  into queue

repeat

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  from queue

until queue is empty

queue



Employees
Job_History
Departments

path

Employees
Job_History
Departments

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

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  from queue

until queue is empty

queue

Job\_History

Departments

path

Employees

Job\_History

Departments



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

let  $T_0 \dots T_m$  be the base tables

create 2 dynamic arrays queue and path

insert  $T_0$  into path

insert  $T_0$  into queue

repeat

→  $T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  from queue

until queue is empty

queue



Job\_History

Departments

path

Employees

Job\_History

Departments

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

let  $T_0 \dots T_m$  be the base tables

create 2 dynamic arrays queue and path

insert  $T_0$  into path

insert  $T_0$  into queue

repeat

$T_{\text{Element}}$  = First Table in queue

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

if the Link Item is in the join sequence then

if path doesn't contain the Link Item then

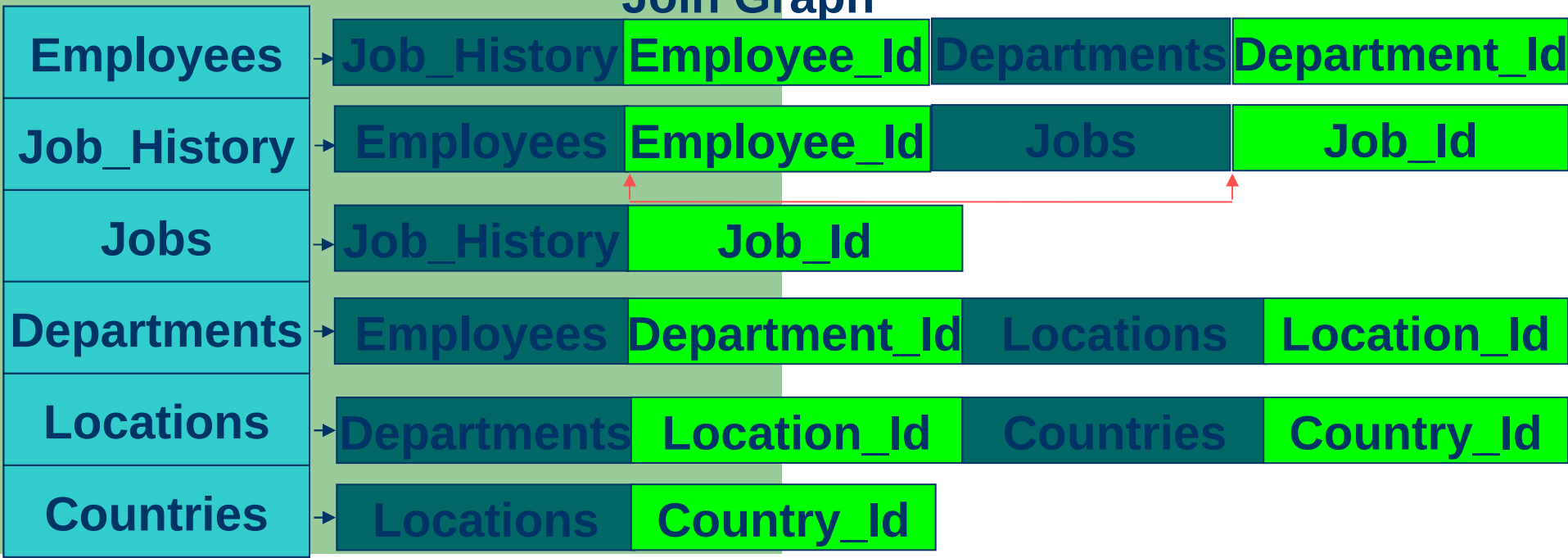
insert Link Item into path

insert Link Item into queue

remove  $T_{\text{Element}}$  from queue

until queue is empty

## Join Graph



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

$T_{\text{Element}}$  = First Table in queue

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

if the Link Item is in the join sequence then

if path doesn't contain the Link Item then

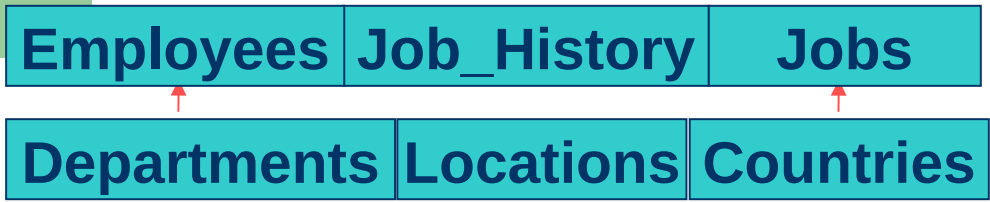
insert Link Item into path

insert Link Item into queue

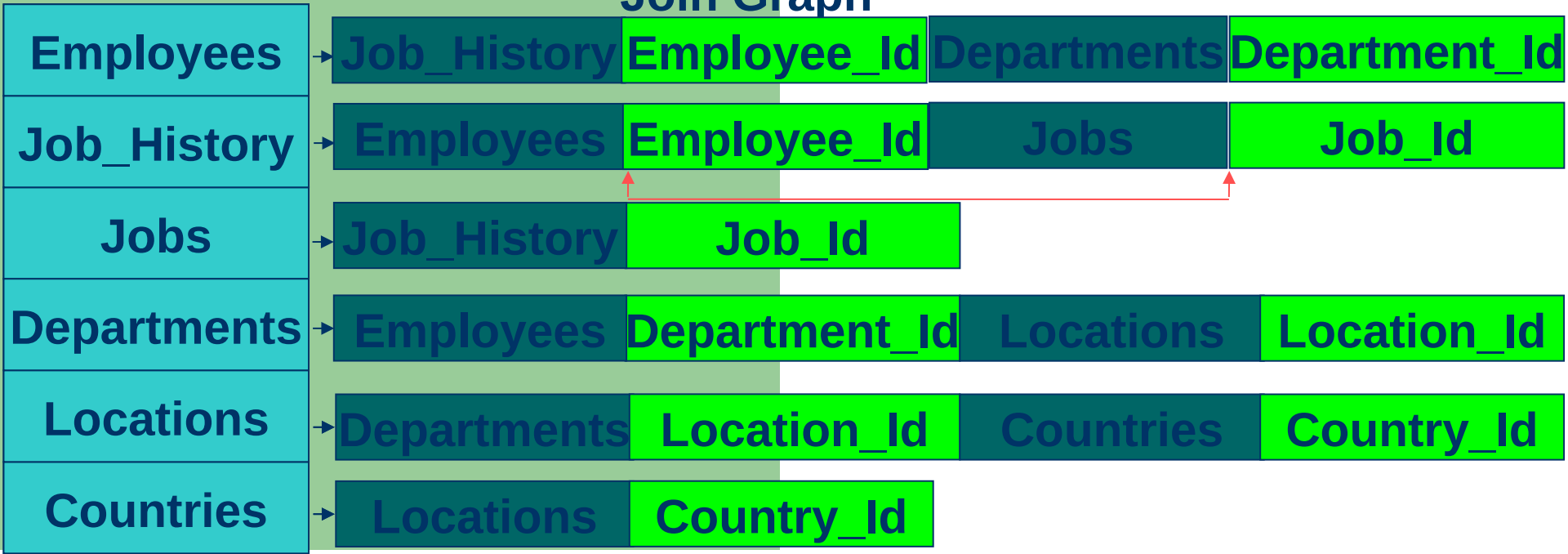
remove  $T_{\text{Element}}$  from queue

until queue is empty

## Join Base Tables



## Join Graph



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

$T_{\text{Element}}$  = First Table in queue

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

if the Link Item is in the join sequence then

if path doesn't contain the Link Item then

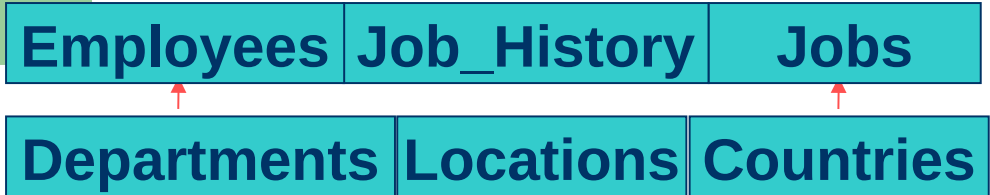
insert Link Item into path

insert Link Item into queue

remove  $T_{\text{Element}}$  from queue

until queue is empty

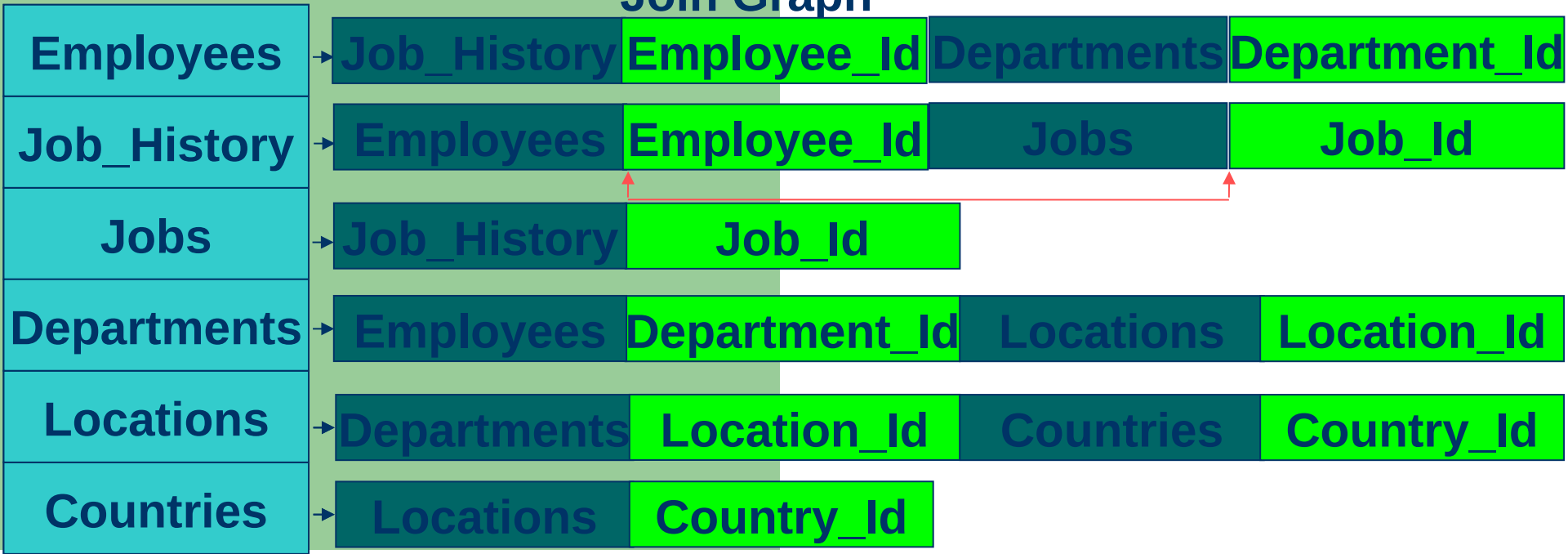
## Join Base Tables



path



## Join Graph



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

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  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  $T_0 \dots T_m$  be the base tables

create 2 dynamic arrays queue and path

insert  $T_0$  into path

insert  $T_0$  into queue

repeat

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  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  $T_0 \dots T_m$  be the base tables

create 2 dynamic arrays queue and path

insert  $T_0$  into path

insert  $T_0$  into queue

repeat

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  from queue

until queue is empty

queue

Departments

Jobs

path

Employees

Job\_History

Departments

Jobs

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

let  $T_0 \dots T_m$  be the base tables

create 2 dynamic arrays queue and path

insert  $T_0$  into path

insert  $T_0$  into queue

repeat

→  $T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  from queue

until queue is empty

queue



Departments

Jobs

path

Employees

Job\_History

Departments

Jobs



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

let  $T_0 \dots T_m$  be the base tables

create 2 dynamic arrays queue and path

insert  $T_0$  into path

insert  $T_0$  into queue

repeat

$T_{\text{Element}}$  = First Table in queue

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

if the Link Item is in the join sequence then

if path doesn't contain the Link Item then

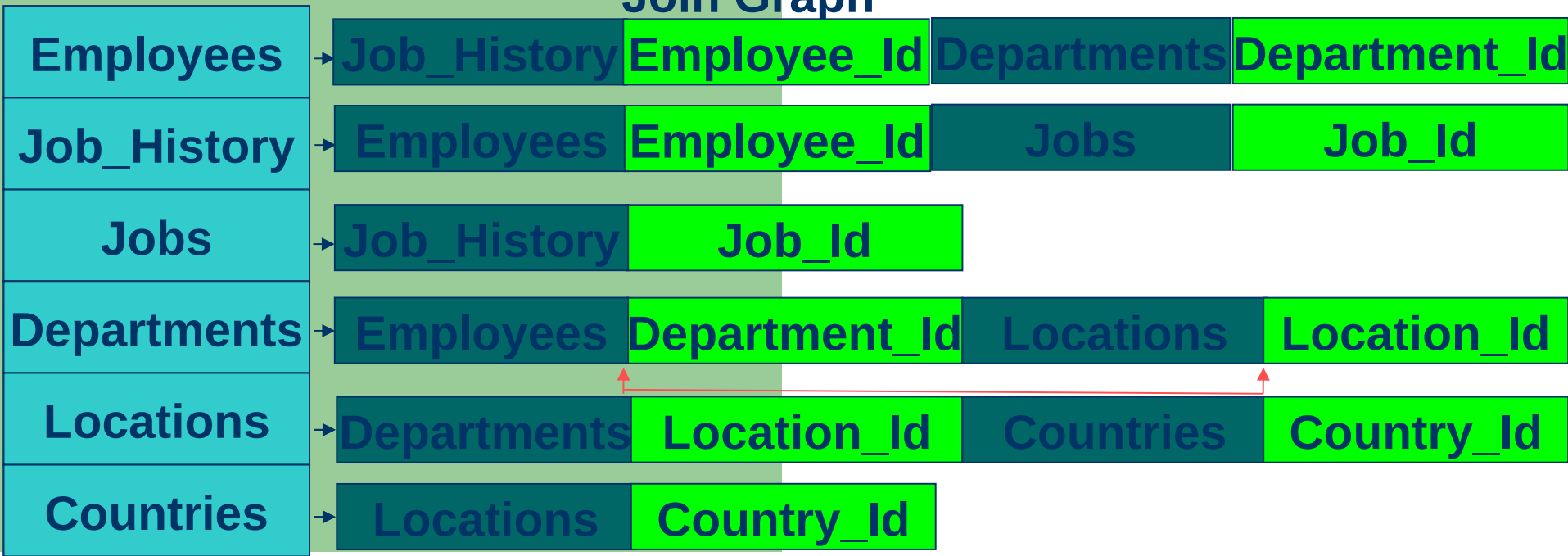
insert Link Item into path

insert Link Item into queue

remove  $T_{\text{Element}}$  from queue

until queue is empty

## Join Graph



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

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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

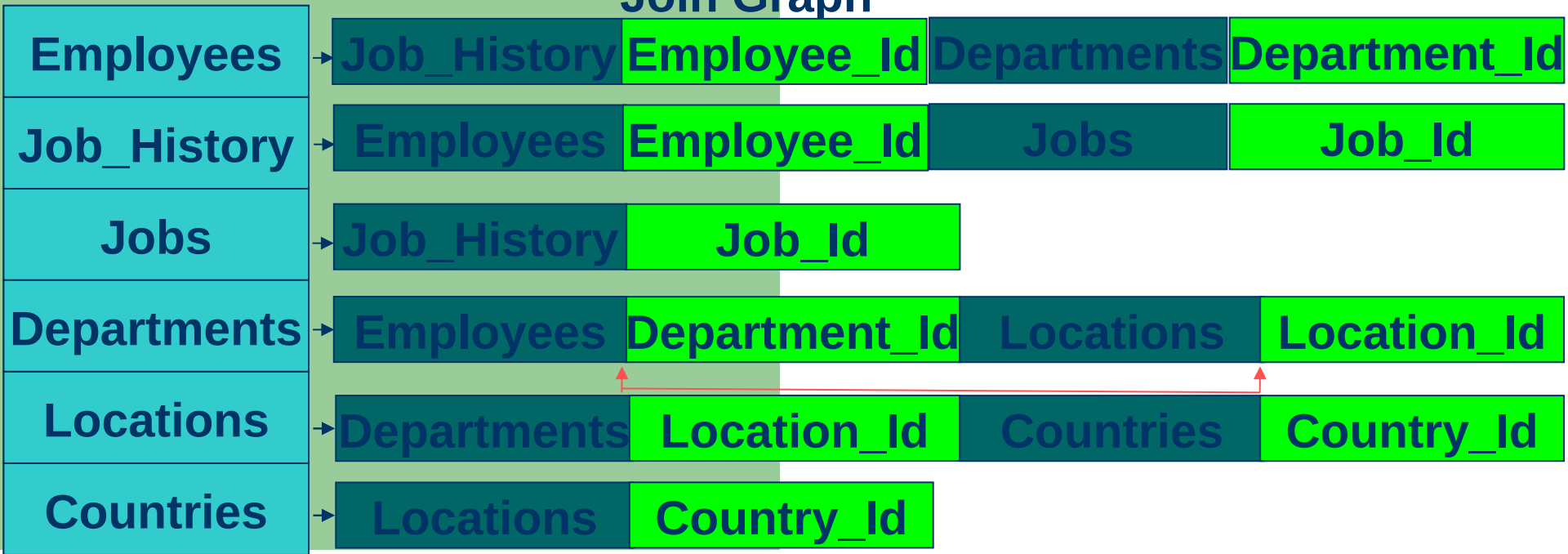
remove  $T_{\text{Element}}$  from queue

until queue is empty

## Join Base Tables



## Join Graph



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

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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

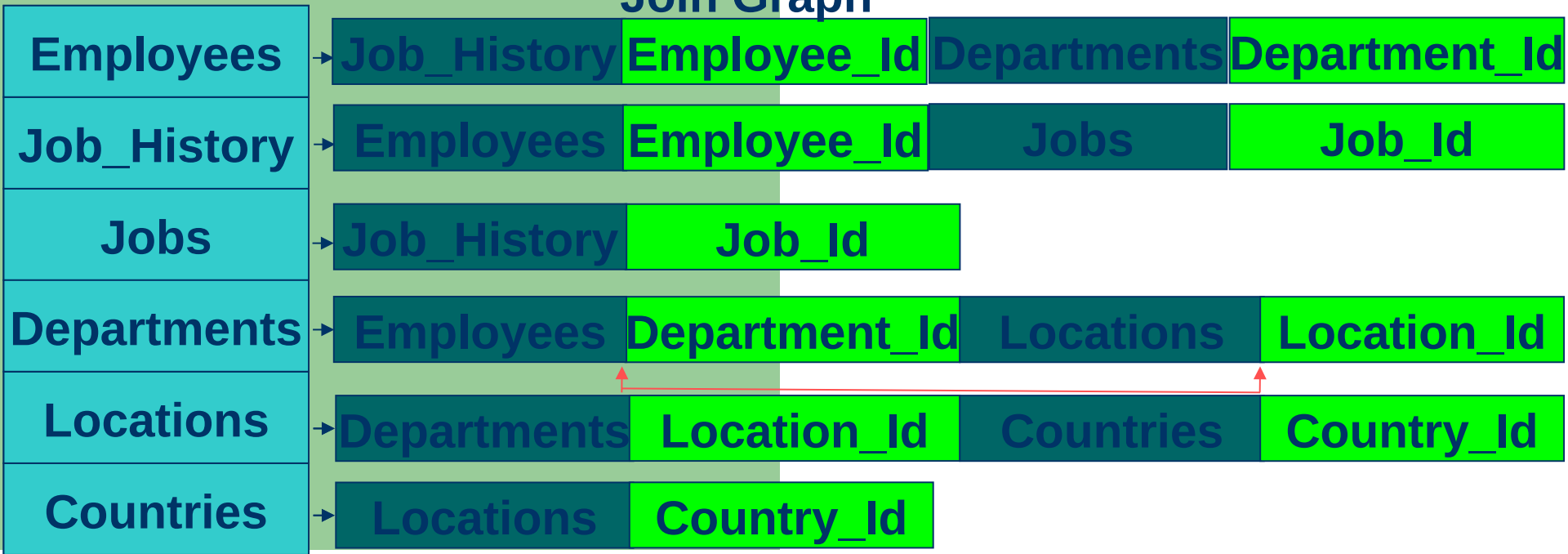
remove  $T_{\text{Element}}$  from queue

until queue is empty

## Join Base Tables



## Join Graph



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

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  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  $T_0 \dots T_m$  be the base tables

create 2 dynamic arrays queue and path

insert  $T_0$  into path

insert  $T_0$  into queue

repeat

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  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  $T_0 \dots T_m$  be the base tables

create 2 dynamic arrays queue and path

insert  $T_0$  into path

insert  $T_0$  into queue

repeat

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  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  $T_0 \dots T_m$  be the base tables

create 2 dynamic arrays queue and path

insert  $T_0$  into path

insert  $T_0$  into queue

repeat

→  $T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  from queue

until queue is empty

queue



path



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

$T_{\text{Element}}$  = First Table in queue

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

if the Link Item is in the join sequence then

if path doesn't contain the Link Item then

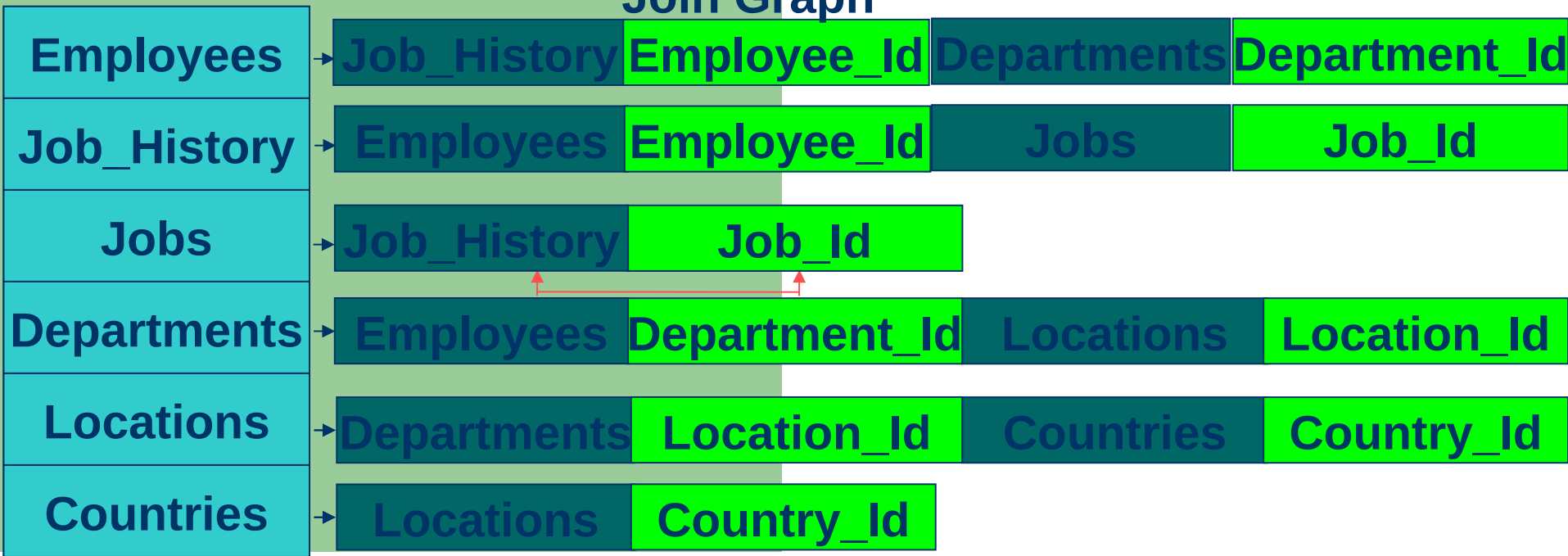
insert Link Item into path

insert Link Item into queue

remove  $T_{\text{Element}}$  from queue

until queue is empty

## Join Graph





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

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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

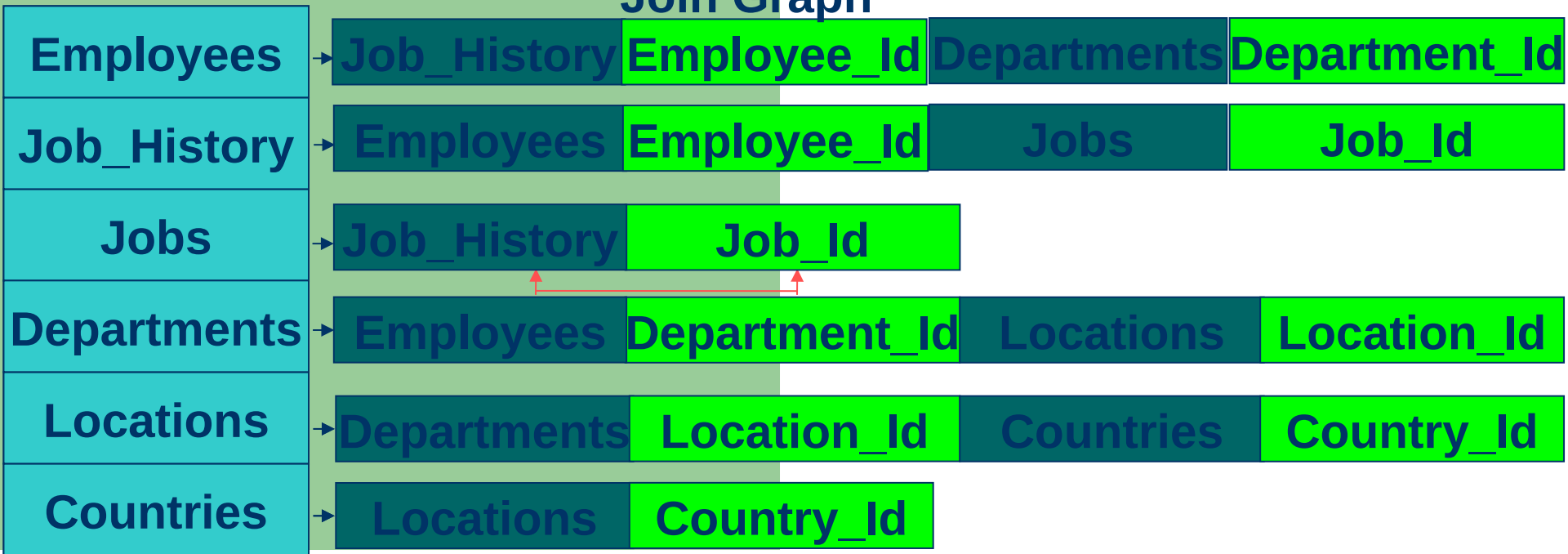
remove  $T_{\text{Element}}$  from queue

until queue is empty

## Join Base Tables



## Join Graph



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

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  from queue

until queue is empty

## Join Base Tables

Employees	Job_History	Jobs
Departments	Locations	Countries

path

Employees	Jobs
Job_History	Locations
Departments	

## 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  $T_0 \dots T_m$  be the base tables

create 2 dynamic arrays queue and path

insert  $T_0$  into path

insert  $T_0$  into queue

repeat

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  from queue

until queue is empty

queue

Locations

path

Employees

Job\_History

Departments

Jobs

Locations

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

let  $T_0 \dots T_m$  be the base tables

create 2 dynamic arrays queue and path

insert  $T_0$  into path

insert  $T_0$  into queue

repeat

→  $T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  from queue

until queue is empty

queue



Locations

path

Employees

Job\_History

Departments

Jobs

Locations

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

let  $T_0 \dots T_m$  be the base tables

create 2 dynamic arrays queue and path

insert  $T_0$  into path

insert  $T_0$  into queue

repeat

$T_{\text{Element}}$  = First Table in queue

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

if the Link Item is in the join sequence then

if path doesn't contain the Link Item then

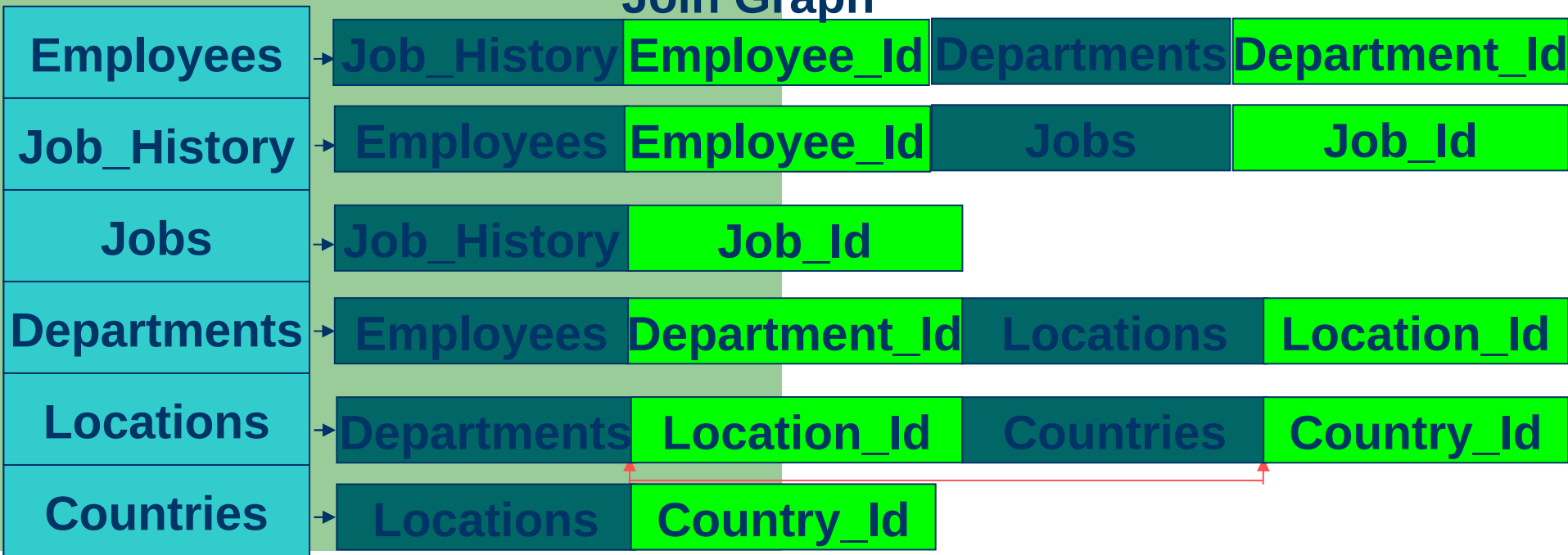
insert Link Item into path

insert Link Item into queue

remove  $T_{\text{Element}}$  from queue

until queue is empty

## Join Graph



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

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  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  $T_0 \dots T_m$  be the base tables

create 2 dynamic arrays queue and path

insert  $T_0$  into path

insert  $T_0$  into queue

repeat

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  from queue

until queue is empty

## Join Base Tables

Employees	Job_History	Jobs
-----------	-------------	------

Departments	Locations	Countries
-------------	-----------	-----------

path

Employees
-----------

Jobs
------

Job_History
-------------

Locations
-----------

Departments
-------------

## 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  $T_0 \dots T_m$  be the base tables

create 2 dynamic arrays queue and path

insert  $T_0$  into path

insert  $T_0$  into queue

repeat

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  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  $T_0 \dots T_m$  be the base tables

create 2 dynamic arrays queue and path

insert  $T_0$  into path

insert  $T_0$  into queue

repeat

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  from queue

until queue is empty

queue



Locations

Countries

path

Employees

Job\_History

Departments

Jobs

Locations

Countries

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

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  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  $T_0 \dots T_m$  be the base tables

create 2 dynamic arrays queue and path

insert  $T_0$  into path

insert  $T_0$  into queue

repeat

→  $T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  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  $T_0 \dots T_m$  be the base tables

create 2 dynamic arrays queue and path

insert  $T_0$  into path

insert  $T_0$  into queue

repeat

$T_{\text{Element}}$  = First Table in queue

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

if the Link Item is in the join sequence then

if path doesn't contain the Link Item then

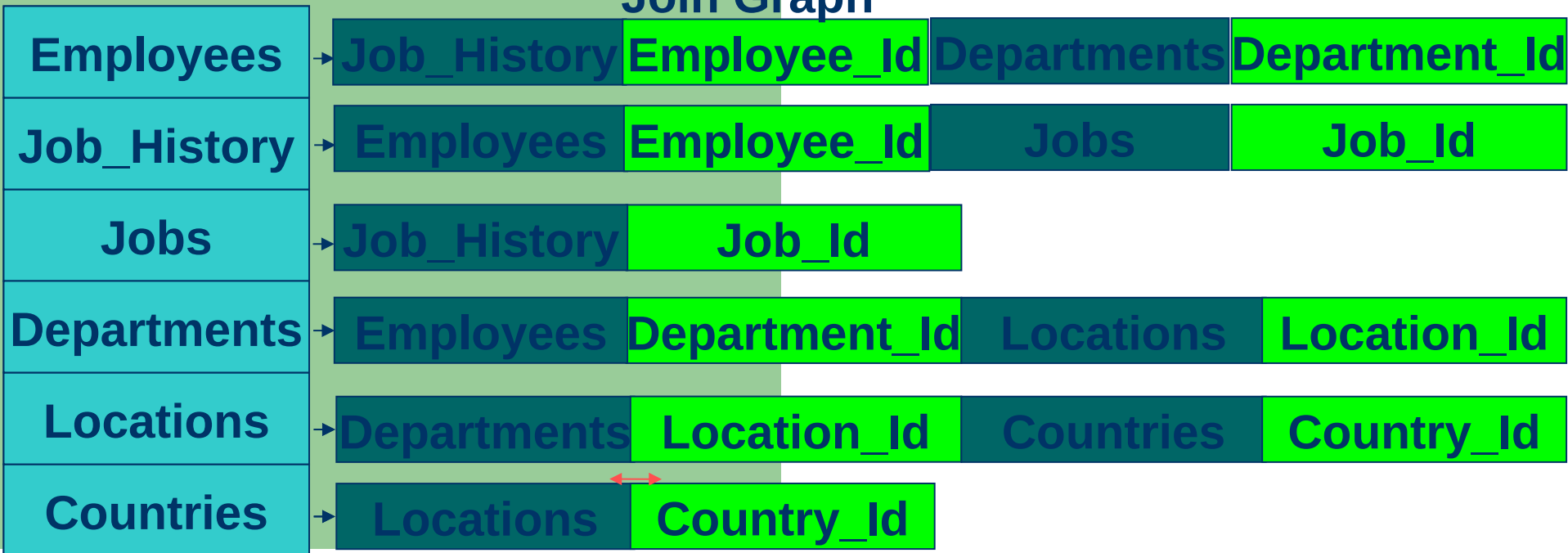
insert Link Item into path

insert Link Item into queue

remove  $T_{\text{Element}}$  from queue

until queue is empty

## Join Graph



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

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  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  $T_0 \dots T_m$  be the base tables

create 2 dynamic arrays queue and path

insert  $T_0$  into path

insert  $T_0$  into queue

repeat

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  from queue

until queue is empty

## Join Base Tables

Employees	Job_History	Jobs
Departments	Locations	Countries

↑  
**path**

Employees	Jobs
Job_History	Locations
Departments	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  $T_0 \dots T_m$  be the base tables

create 2 dynamic arrays queue and path

insert  $T_0$  into path

insert  $T_0$  into queue

repeat

$T_{\text{Element}}$  = First Table in queue

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

if the Link Item is in the join sequence then

if path doesn't contain the Link Item then

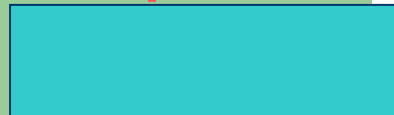
insert Link Item into path

insert Link Item into queue

→ remove  $T_{\text{Element}}$  from queue

until queue is empty

queue



path



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

$T_{\text{Element}}$  = First Table in queue

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

if the Link Item is in the join sequence then

if path doesn't contain the Link Item then

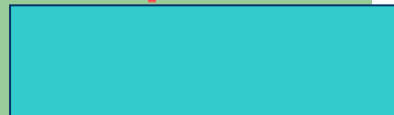
insert Link Item into path

insert Link Item into queue

remove  $T_{\text{Element}}$  from queue

→ until queue is empty

queue



path





→ insert all the names of base tables from path as vertexes in JoinPathList  
create a local buffer buf  
insert into buf the first entry from path  
for all the remainder entries in path do

vertexes

Employees

Job\_History

Departments

Jobs

Locations

Countries

take one  $T_i$  at a time

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

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

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

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

$T_{[buf]} += T_i$

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

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

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

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

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

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

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

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

vertexes

Employees

Job\_History

Departments

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

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

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

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

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

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

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

vertexes

Employees

Job\_History

Departments

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

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

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

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

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

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

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

vertexes

Employees

Job\_History

Departments

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

vertexes

Employees

Job\_History

Departments

Jobs

Locations

Countries

→ take one  $T_i$  at a time

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

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

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

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

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

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

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

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

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

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

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

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

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

vertexes



buf

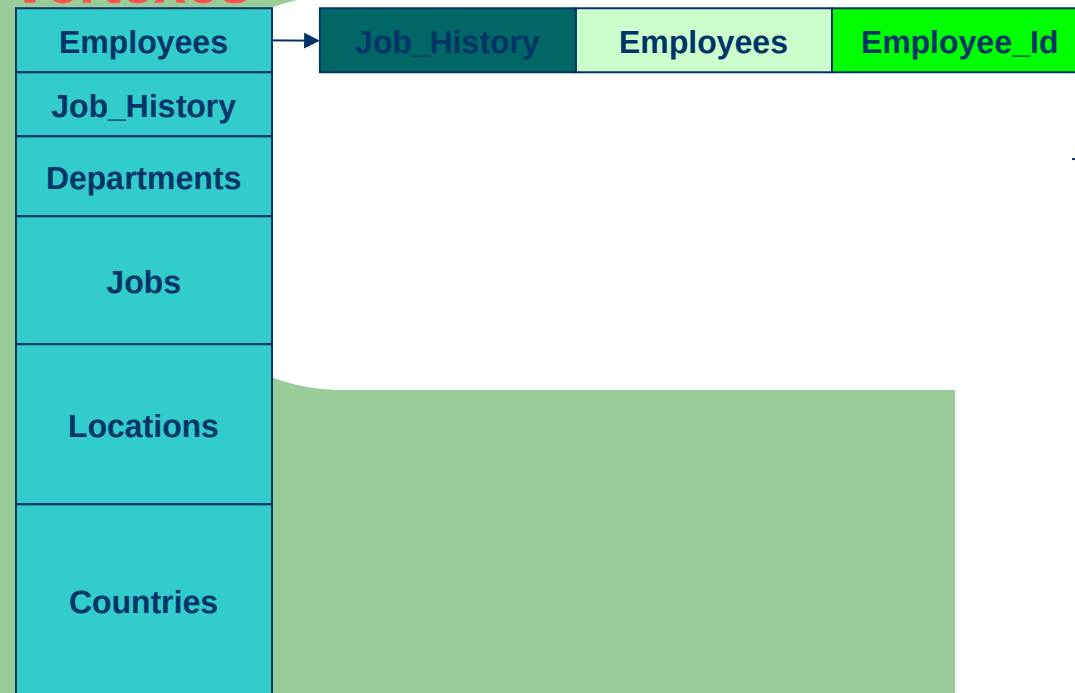


path



insert all the names of base tables from path as vertexes in JoinPathList  
create a local buffer buf  
insert into buf the first entry from path  
for all the remainder entries in path do

vertexes



take one  $T_i$  at a time

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

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

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

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

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

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

buf



path



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

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one  $T_i$  at a time

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

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

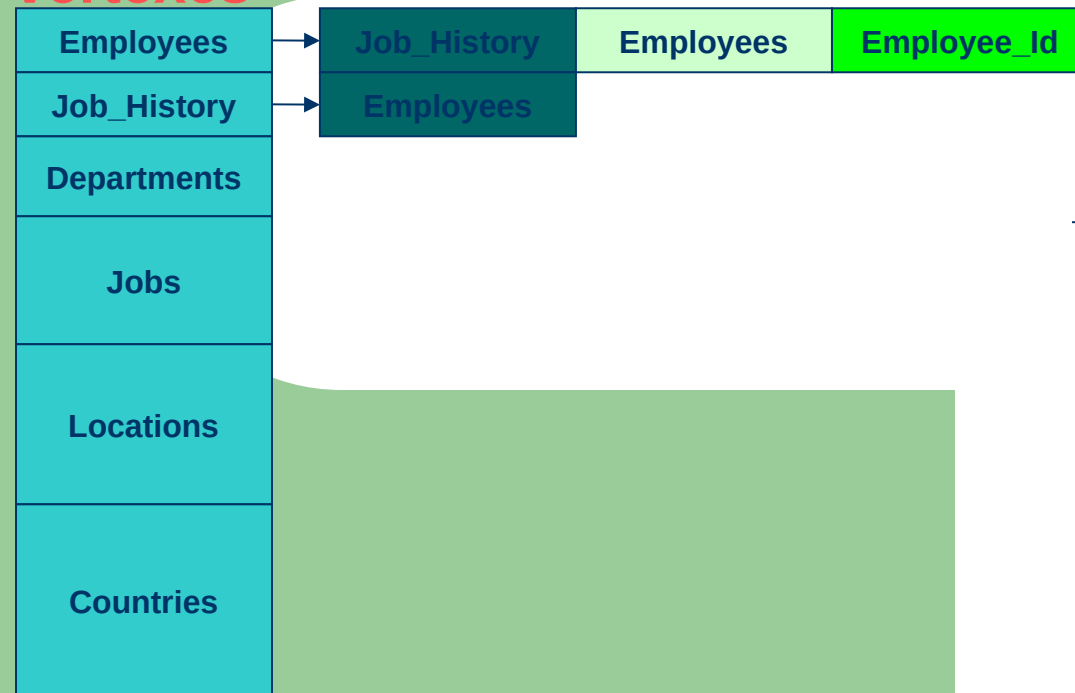
→  $\text{JoinPathAdjacentList}(T_{[\text{buf}]}) = T_i$

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

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

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

vertexes



buf



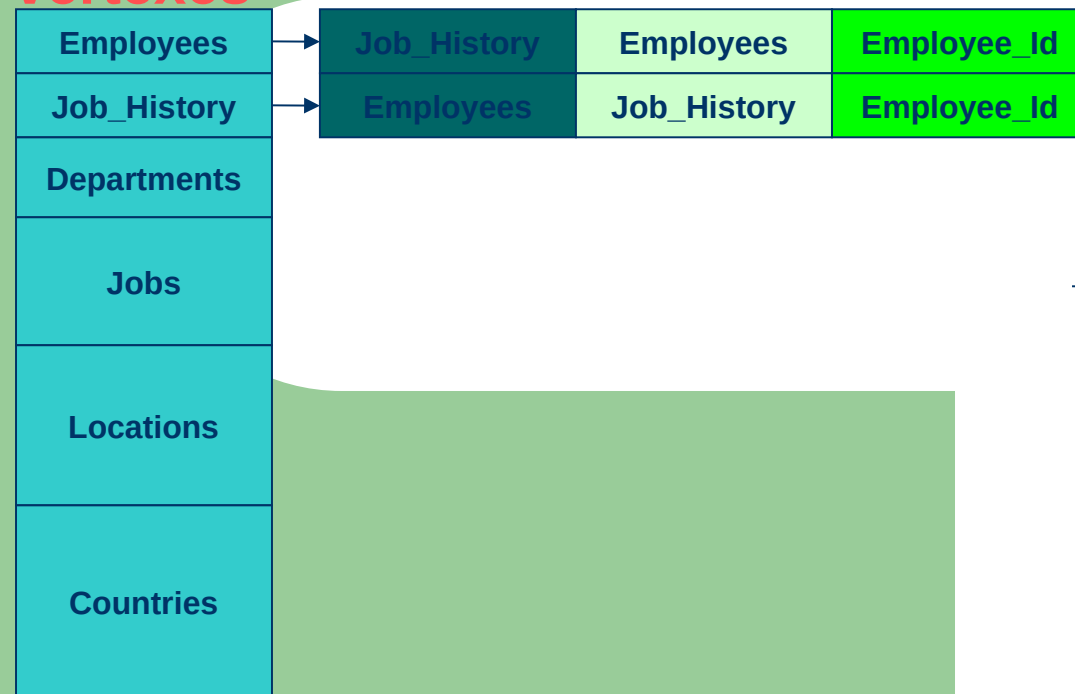
path





insert all the names of base tables from path as vertexes in JoinPathList  
 create a local buffer buf  
 insert into buf the first entry from path  
 for all the remainder entries in path do

**vertexes**



take one  $T_i$  at a time

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

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

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

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

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

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

**buf**



**path**



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

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one  $T_i$  at a time

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

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

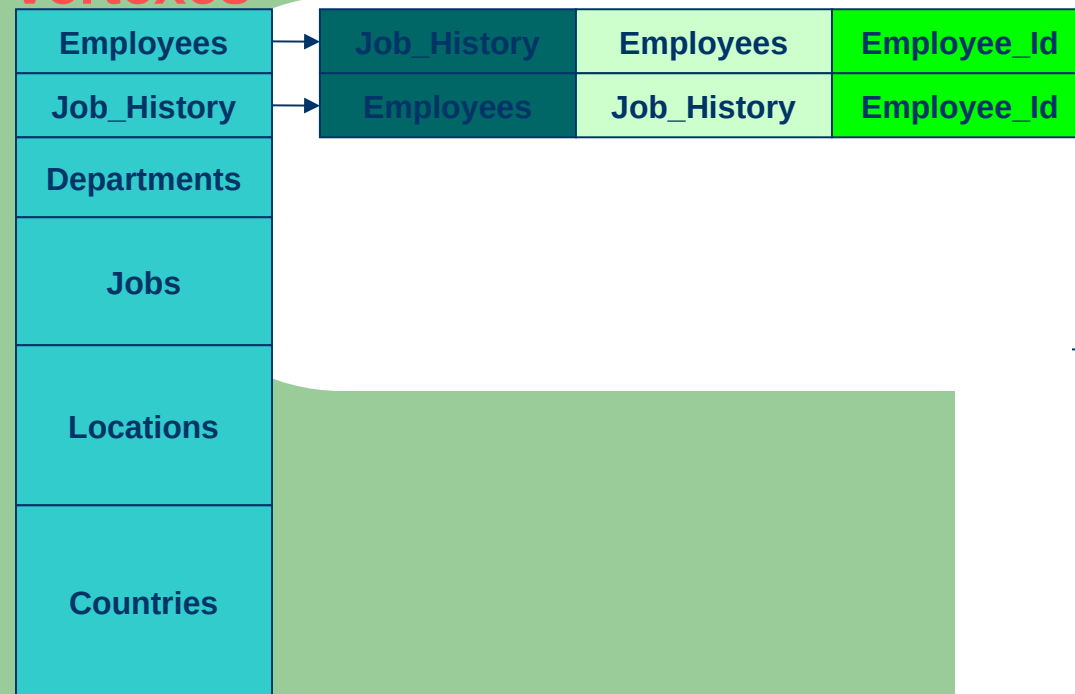
$\text{JoinPathAdjacentList}(T_{[\text{buf}]}) = T_i$

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

→  $T_{[\text{buf}]} += T_i$

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

vertexes



buf

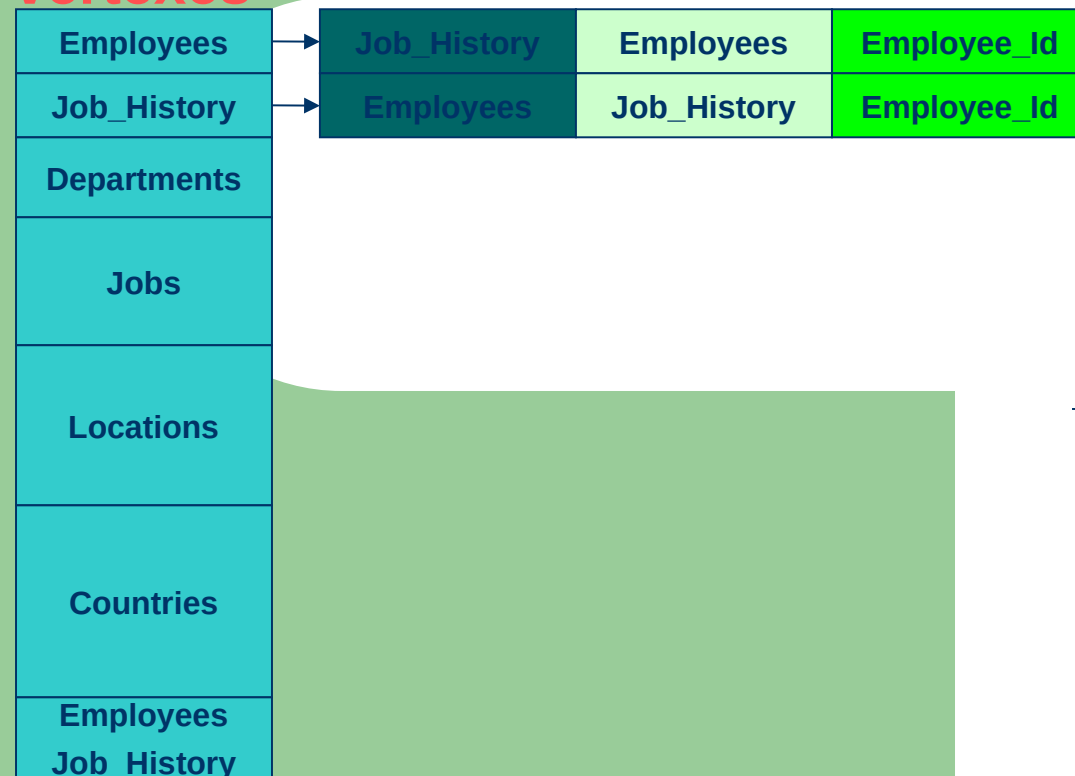


path



insert all the names of base tables from path as vertexes in JoinPathList  
 create a local buffer buf  
 insert into buf the first entry from path  
 for all the remainder entries in path do

vertexes



take one  $T_i$  at a time

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

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

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

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

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

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

buf



path



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

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one  $T_i$  at a time

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

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

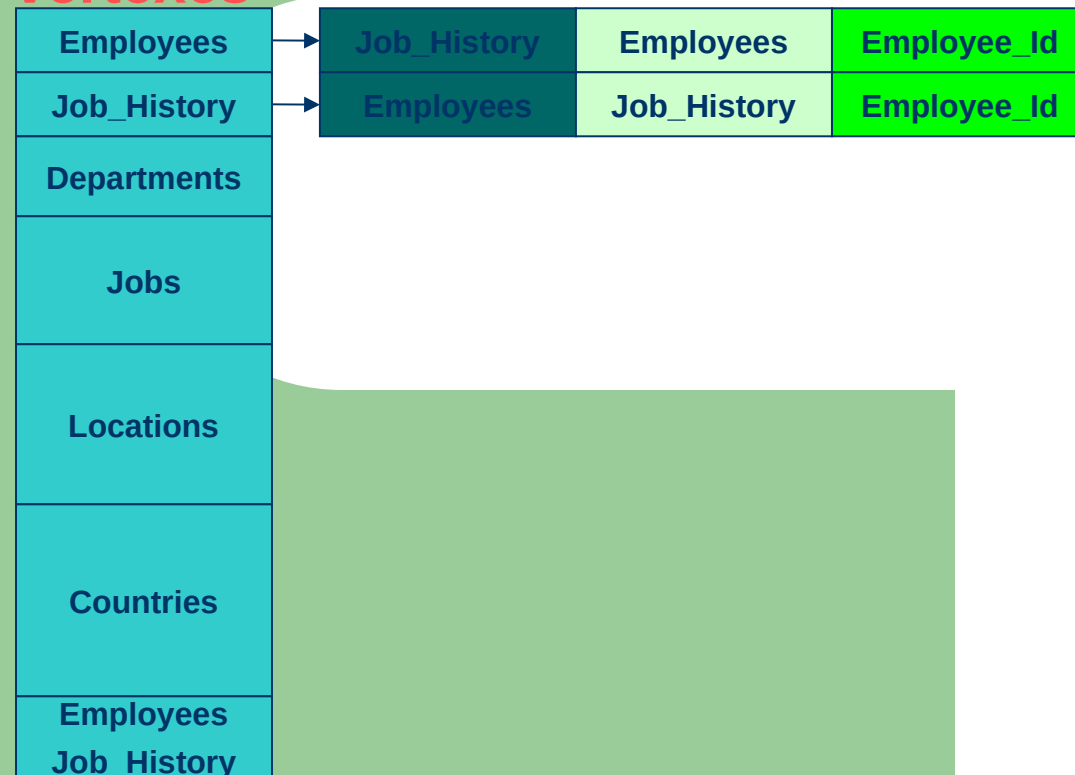
$\text{JoinPathAdjacentList}(T_{[\text{buf}]}) = T_i$

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

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

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

vertexes



buf

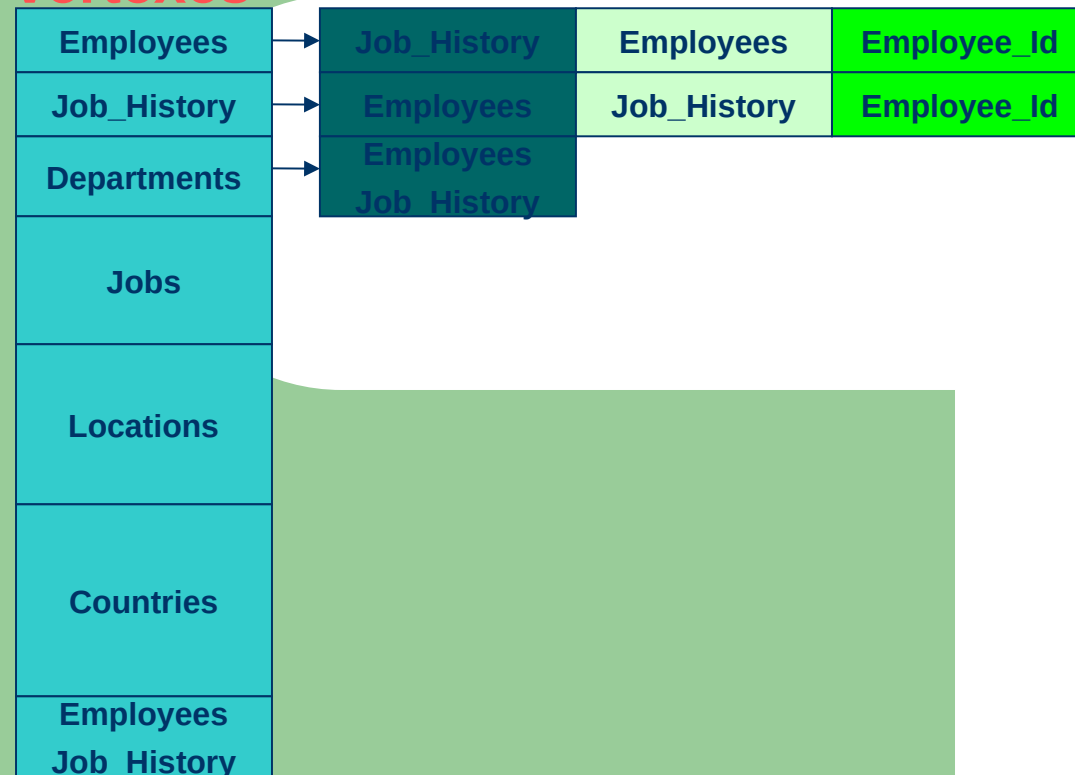
Employees  
Job\_History

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

vertexes



take one  $T_i$  at a time

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

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

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

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

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

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

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

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

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

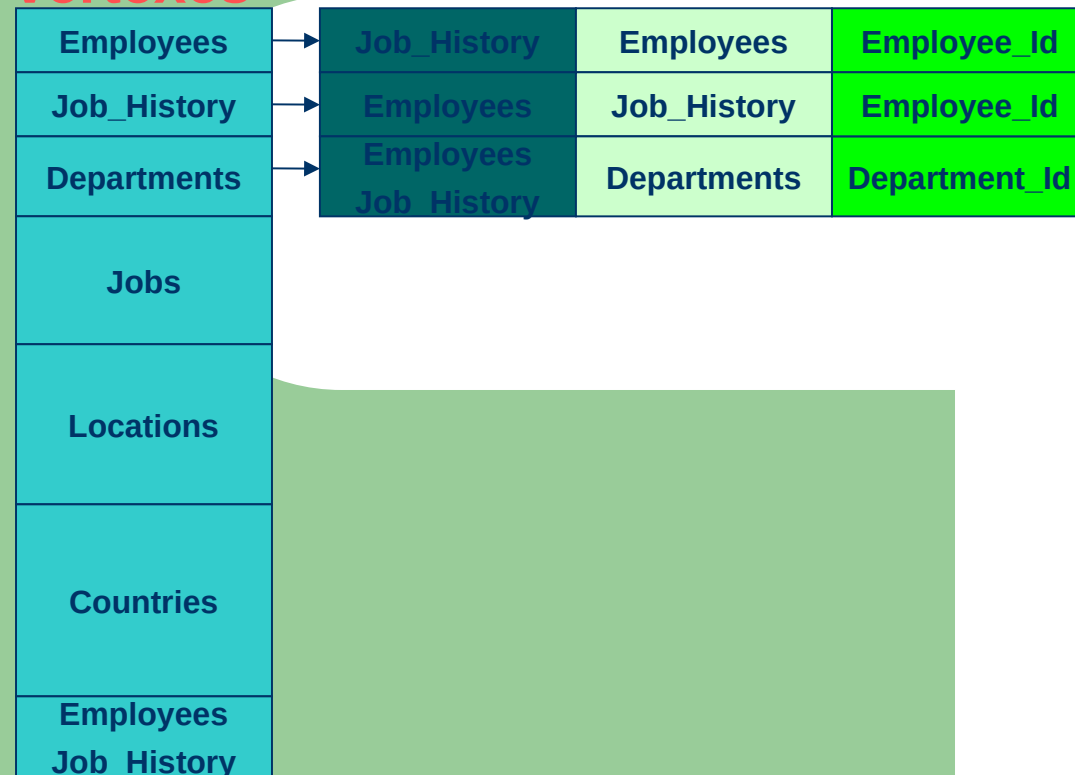
$\text{JoinPathAdjacentList}(T_{[\text{buf}]}) = T_i$

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

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

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

vertexes



buf

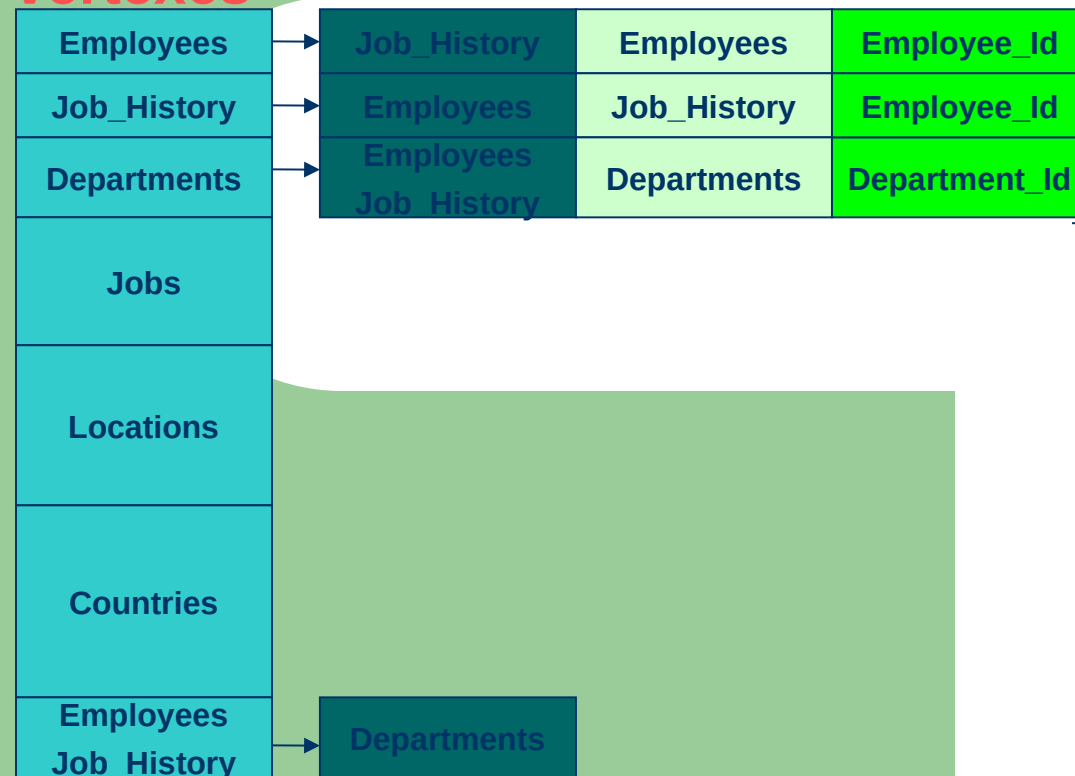
Employees  
Job\_History

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

vertexes



take one  $T_i$  at a time

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

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

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

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

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

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

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

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

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

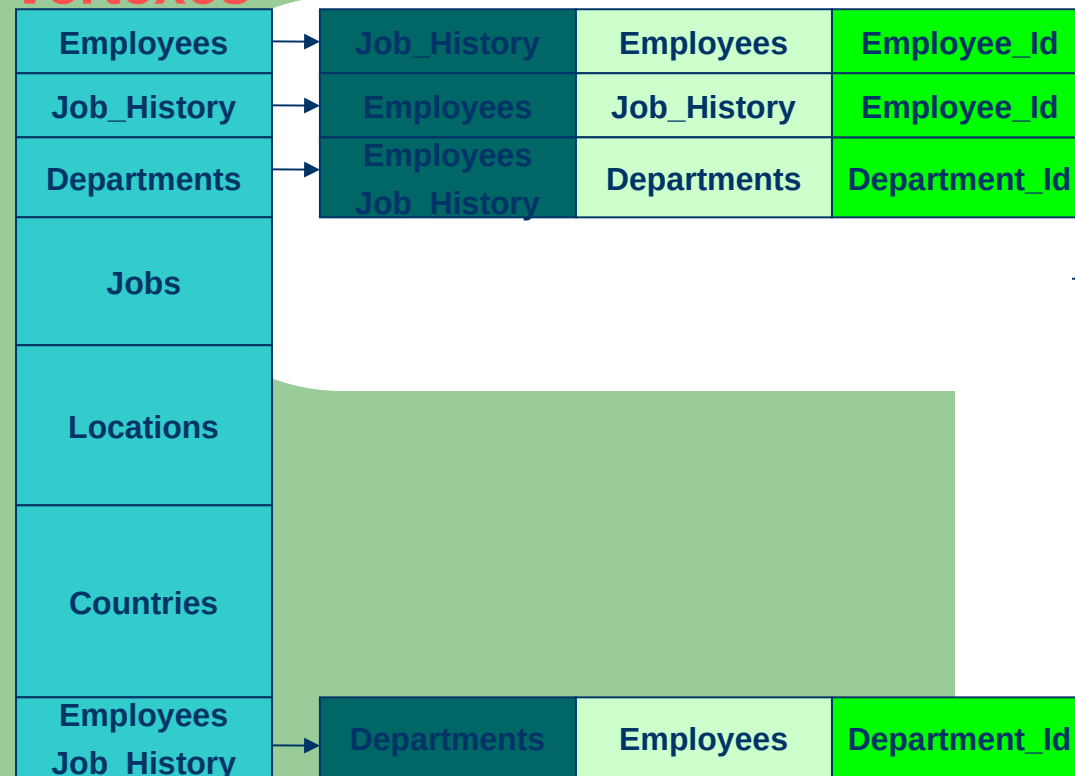
$\text{JoinPathAdjacentList}(T_{[\text{buf}]}) = T_i$

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

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

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

vertexes



buf

Employees  
Job\_History

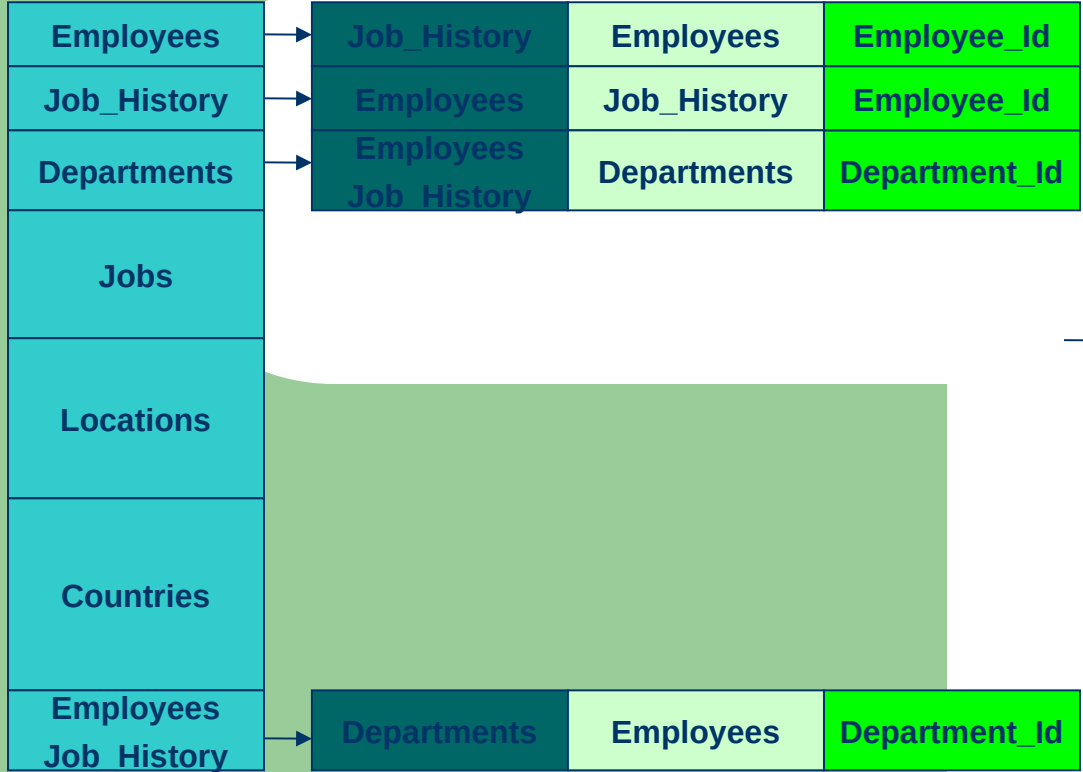
path

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

vertexes



take one  $T_i$  at a time

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

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

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

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

→  $T_{[buf]} += T_i$

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

buf



path



insert all the names of base tables from path as vertexes in JoinPathList  
 create a local buffer buf  
 insert into buf the first entry from path  
 for all the remainder entries in path do

vertexes

take one  $T_i$  at a time

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

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

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

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

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

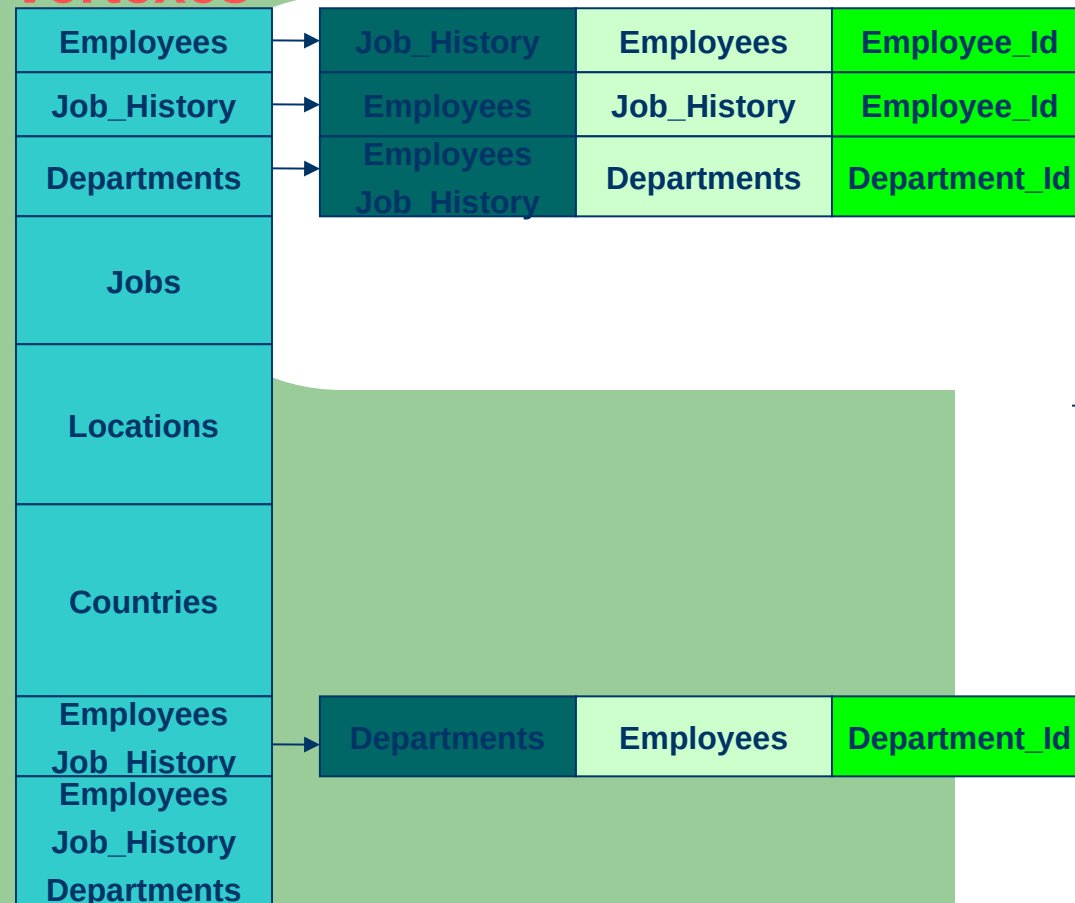
→  $\text{Insert NodesList}[T_{[\text{buf}]}] = T_{[\text{buf}]}$

buf

path

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

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

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

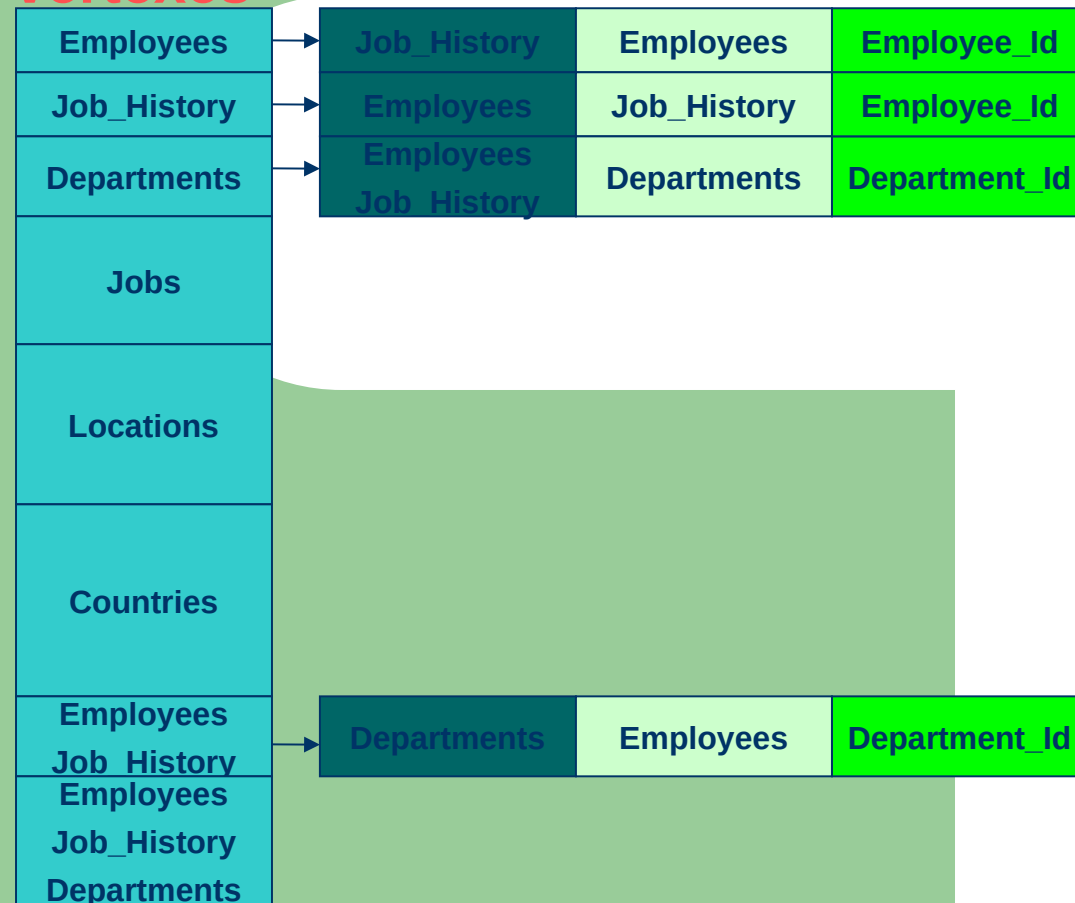
$\text{JoinPathAdjacentList}(T_{[\text{buf}]}) = T_i$

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

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

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

vertexes



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

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

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

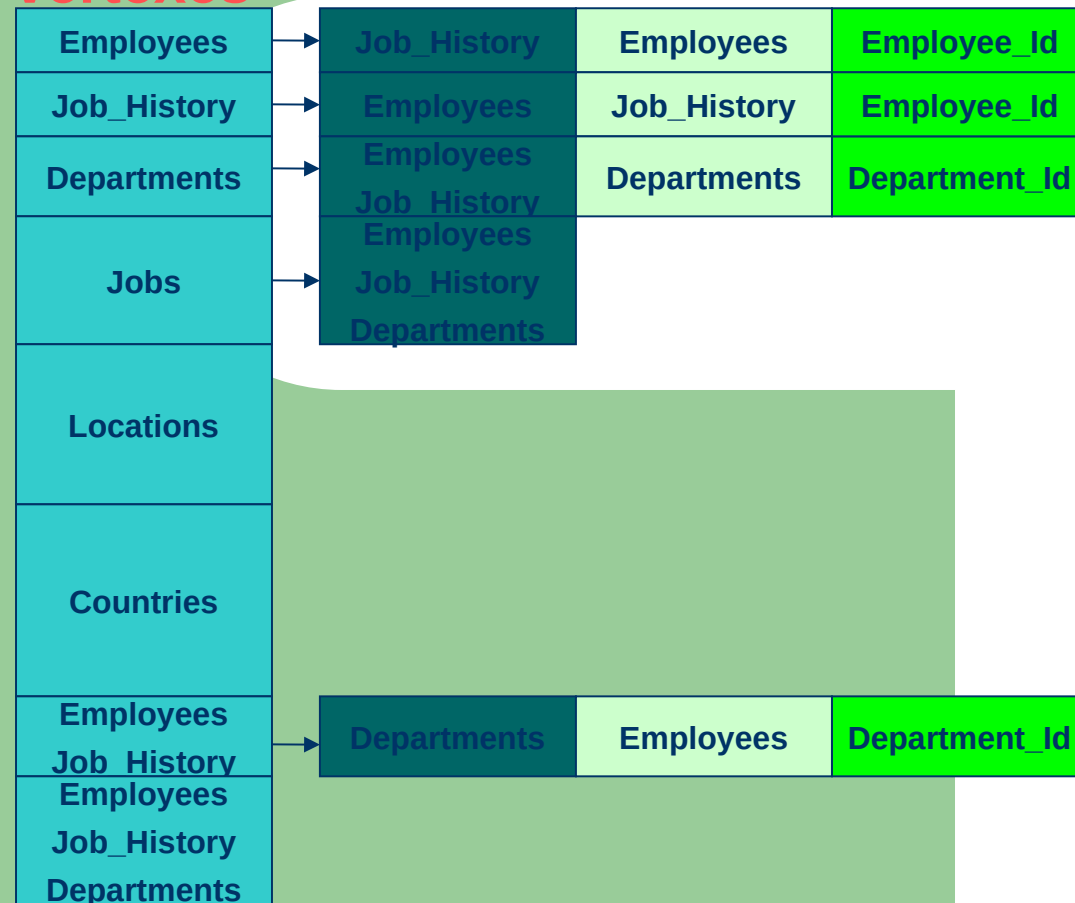
$\text{JoinPathAdjacentList}(T_{[\text{buf}]}) = T_i$

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

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

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

vertexes



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

vertexes

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

Departments	Employees	Department_Id
-------------	-----------	---------------

take one  $T_i$  at a time

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

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

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

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

$T_{[buf]} += T_i$

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

buf

Employees
Job_History
Departments

path

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

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

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

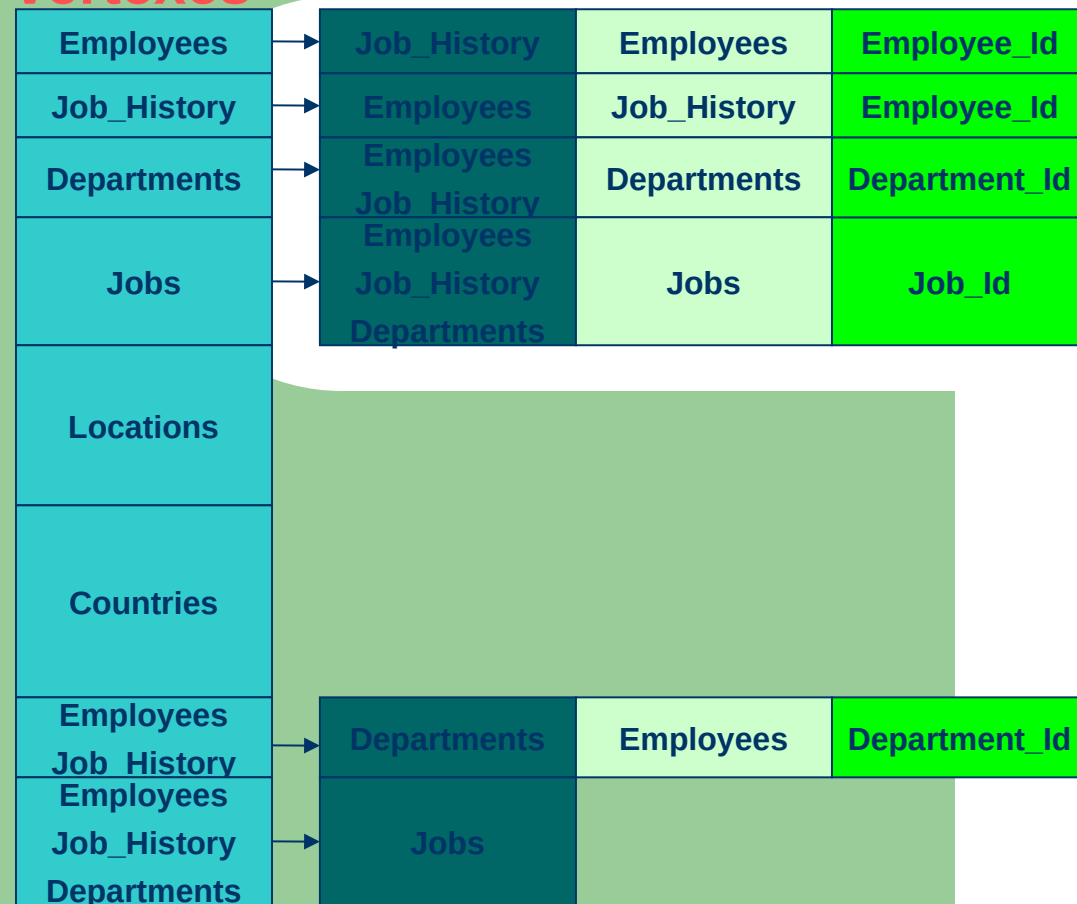
$\text{JoinPathAdjacentList}(T_{[\text{buf}]}) = T_i$

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

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

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

vertexes



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

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

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

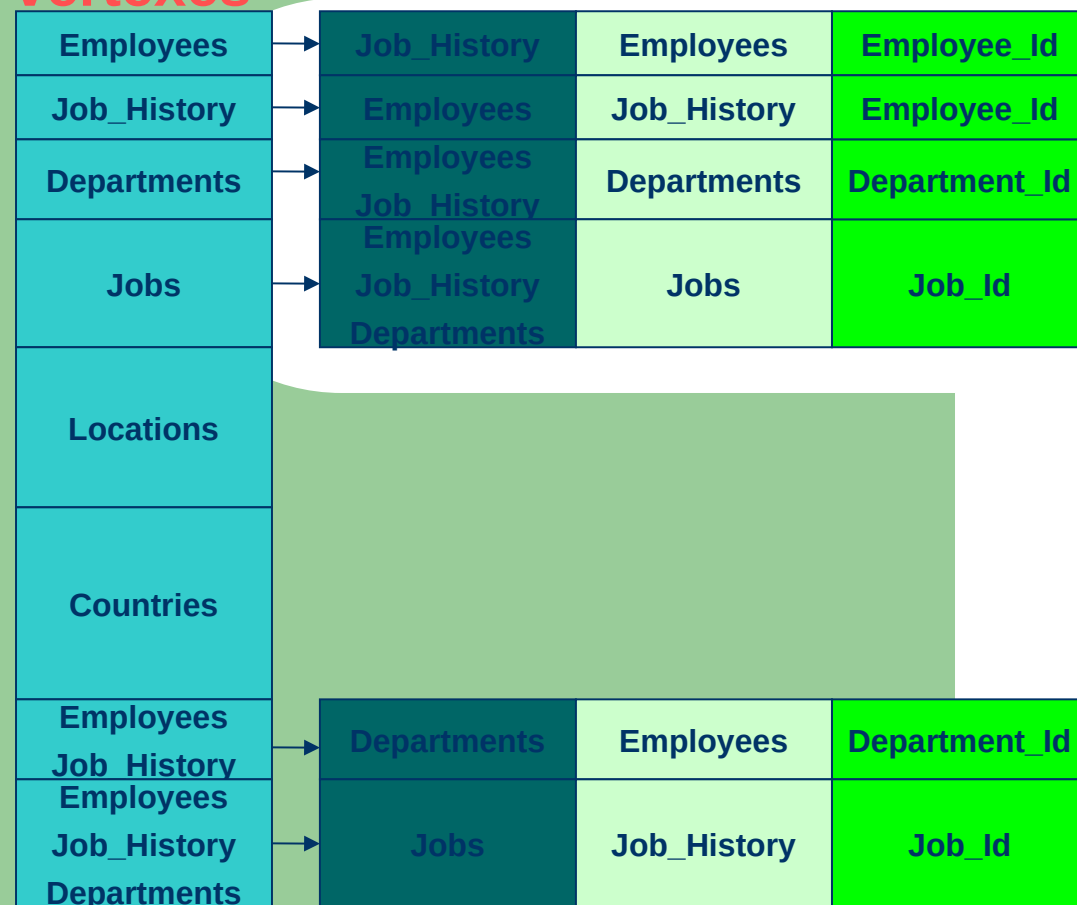
$\text{JoinPathAdjacentList}(T_{[\text{buf}]}) = T_i$

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

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

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

vertexes



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

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

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

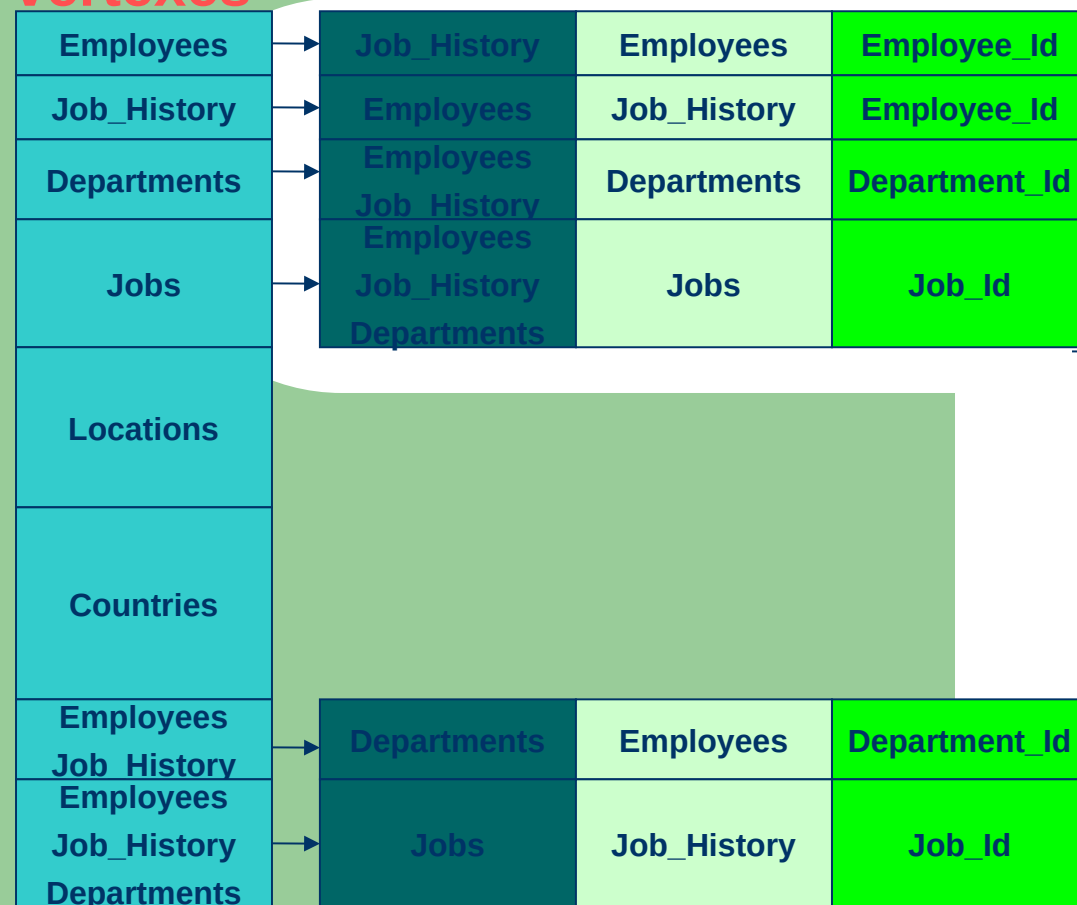
$\text{JoinPathAdjacentList}(T_{[\text{buf}]}) = T_i$

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

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

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

vertexes



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

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

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

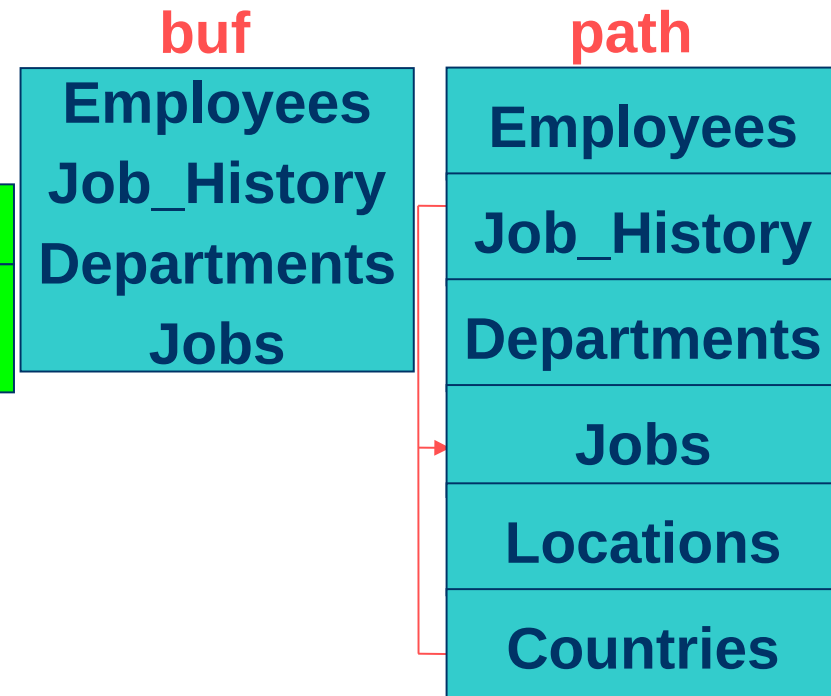
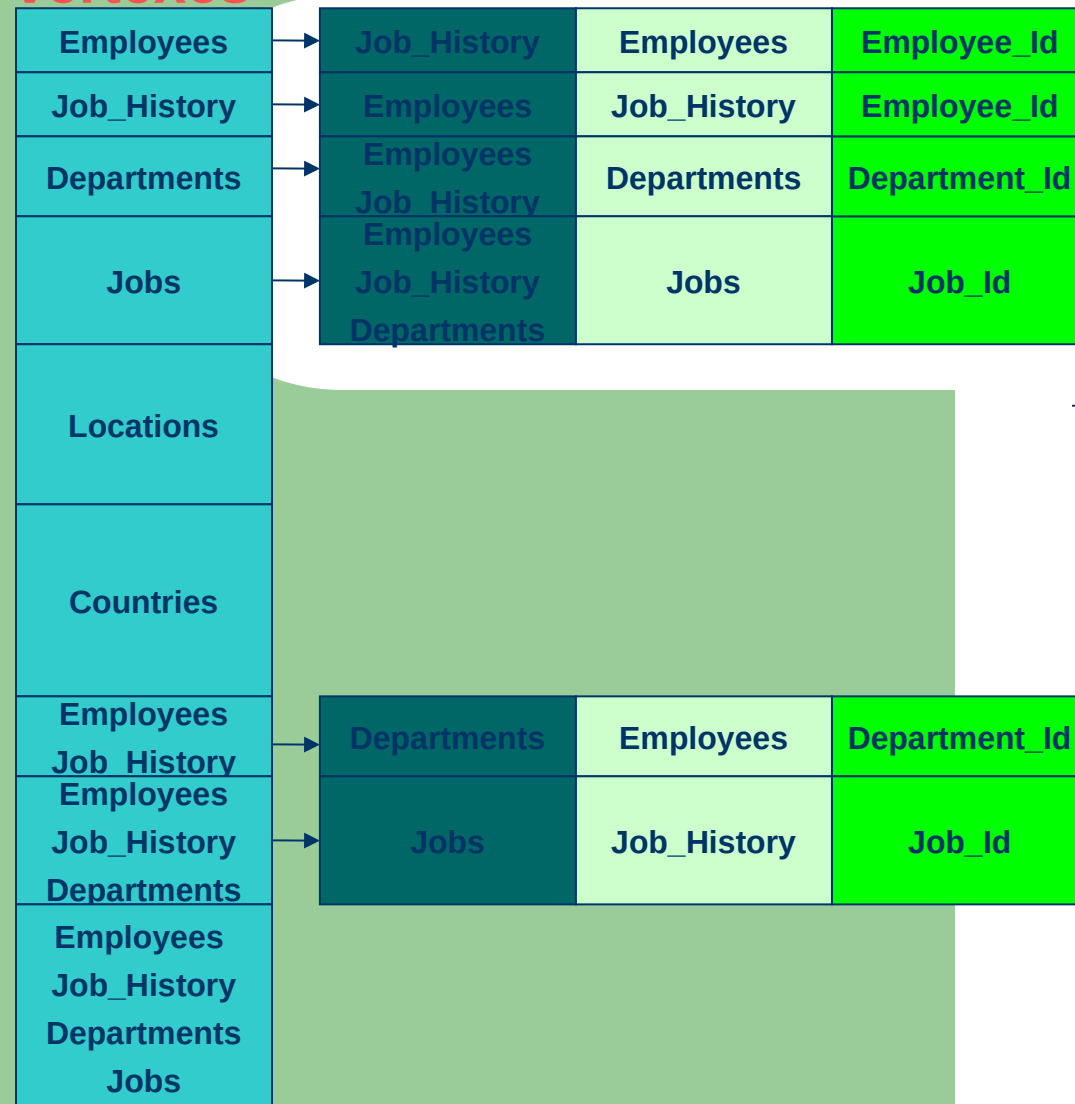
$\text{JoinPathAdjacentList}(T_{[\text{buf}]}) = T_i$

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

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

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

vertexes



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

vertexes

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

take one  $T_i$  at a time

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

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

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

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

$T_{[buf]} += T_i$

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

Locations
Countries

Employees	Departments	Employees	Department_Id
Job_History			
Employees			
Job_History	Jobs	Job_History	Job_Id
Departments			

Employees
Job_History
Departments
Jobs

buf

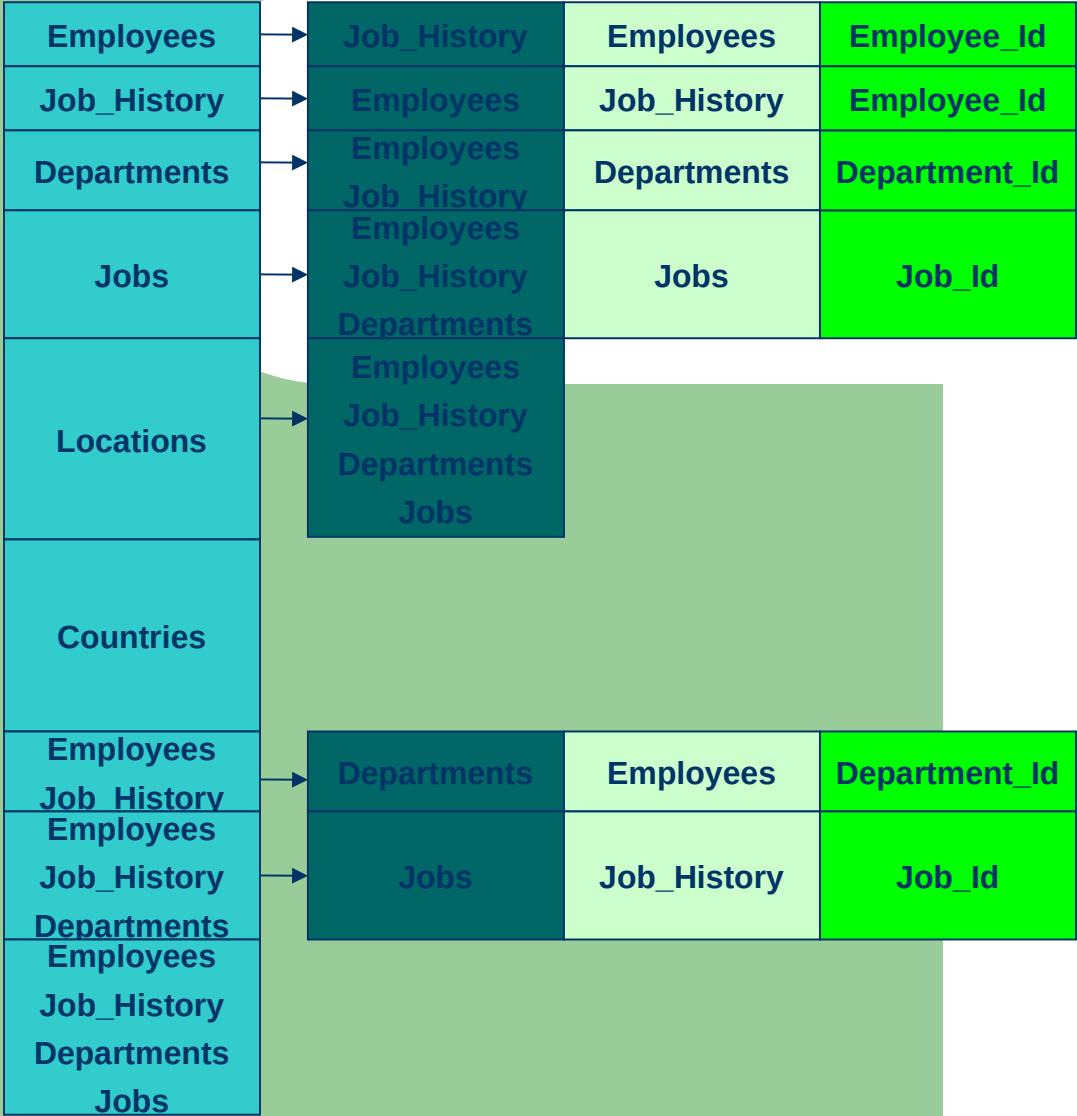
Employees
Job_History
Departments
Jobs

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

vertexes



take one  $T_i$  at a time

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

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

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

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

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

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

buf

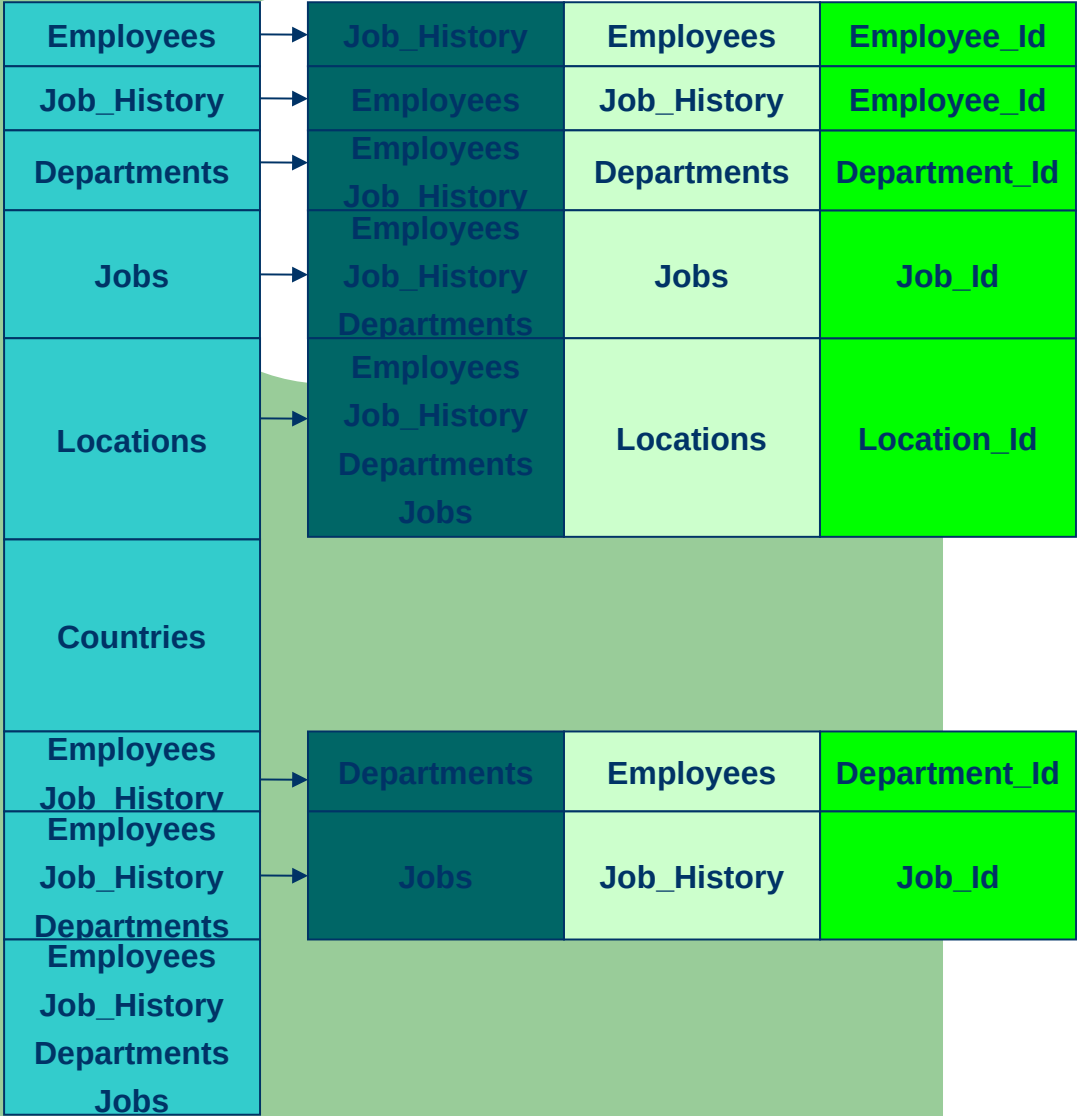


path



insert all the names of base tables from path as vertexes in JoinPathList  
create a local buffer buf  
insert into buf the first entry from path  
for all the remainder entries in path do

vertexes



take one  $T_i$  at a time

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

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

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

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

$T_{[buf]} += T_i$

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

buf



path



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

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one  $T_i$  at a time

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

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

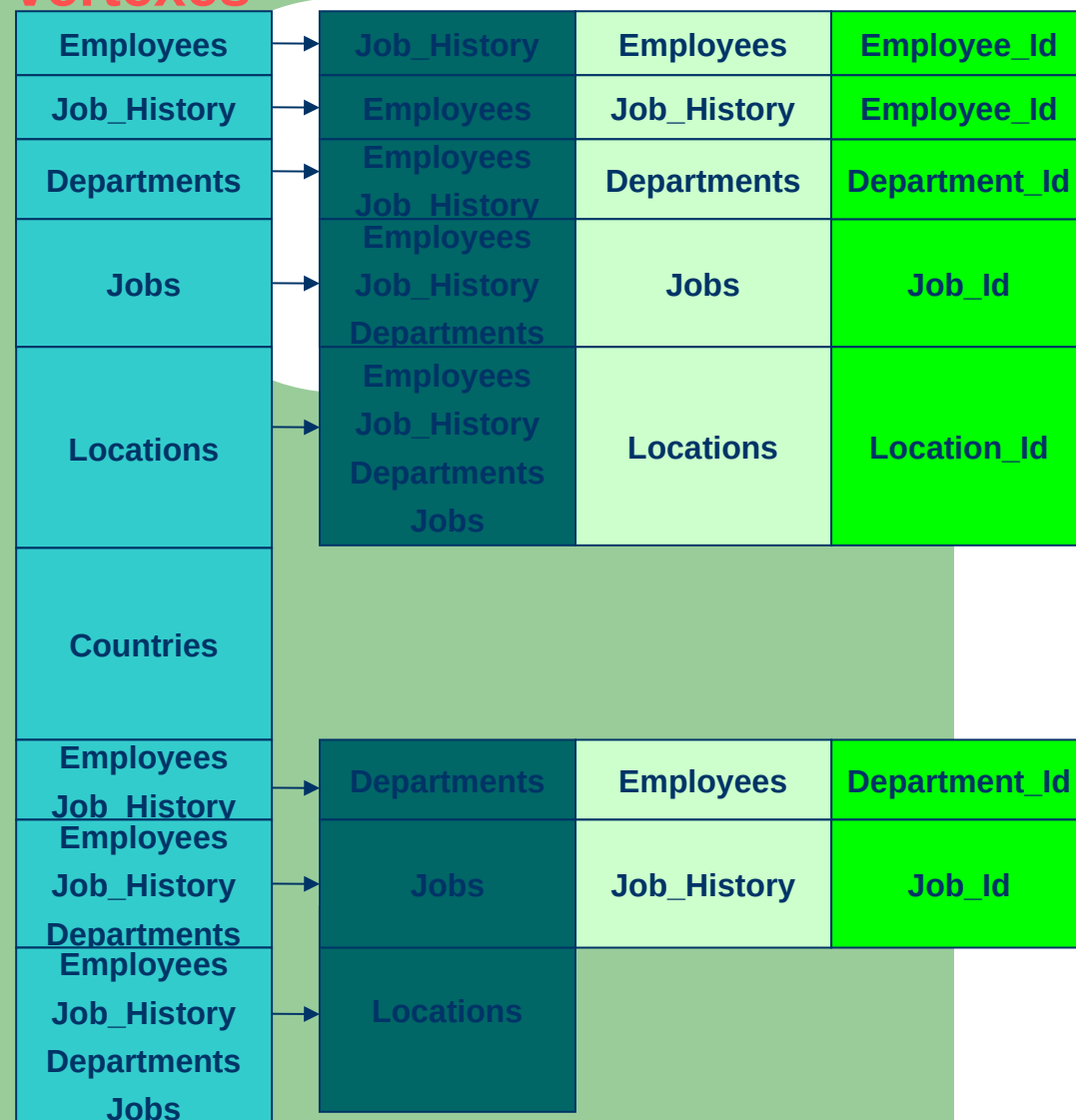
$\text{JoinPathAdjacentList}(T_{[\text{buf}]}) = T_i$

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

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

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

vertexes



buf

path



insert all the names of base tables from path as vertexes in JoinPathList  
create a local buffer buf  
insert into buf the first entry from path  
for all the remainder entries in path do

vertexes



take one  $T_i$  at a time

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

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

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

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

$T_{[buf]} += T_i$

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

buf



path



insert all the names of base tables from path as vertexes in JoinPathList  
create a local buffer buf  
insert into buf the first entry from path  
for all the remainder entries in path do

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 Employees	→	Departments	Employees	Department_Id
Job_History Departments	→	Jobs	Job_History	Job_Id
Employees Job_History Departments Jobs	→	Locations	Departments	Location_Id

take one  $T_i$  at a time

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

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

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

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

$T_{[buf]} += T_i$

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

buf

Employees
Job_History
Departments
Jobs
Locations

path

Employees
Job_History
Departments
Jobs
Locations
Countries

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

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

from path

h path do

take one  $T_i$  at a time

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

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

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

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

$T_{[buf]} += T_i$

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

Countries
-----------

Employees	Departments	Employees	Department_Id
Job_History	Jobs	Job_History	Job_Id
Employees	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



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

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

from path

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

Countries			
-----------	--	--	--

Employees Job_History Employees	Departments	Employees	Department_Id
Job_History Departments Employees	Jobs	Job_History	Job_Id
Job_History Departments Jobs	Locations	Departments	Location_Id

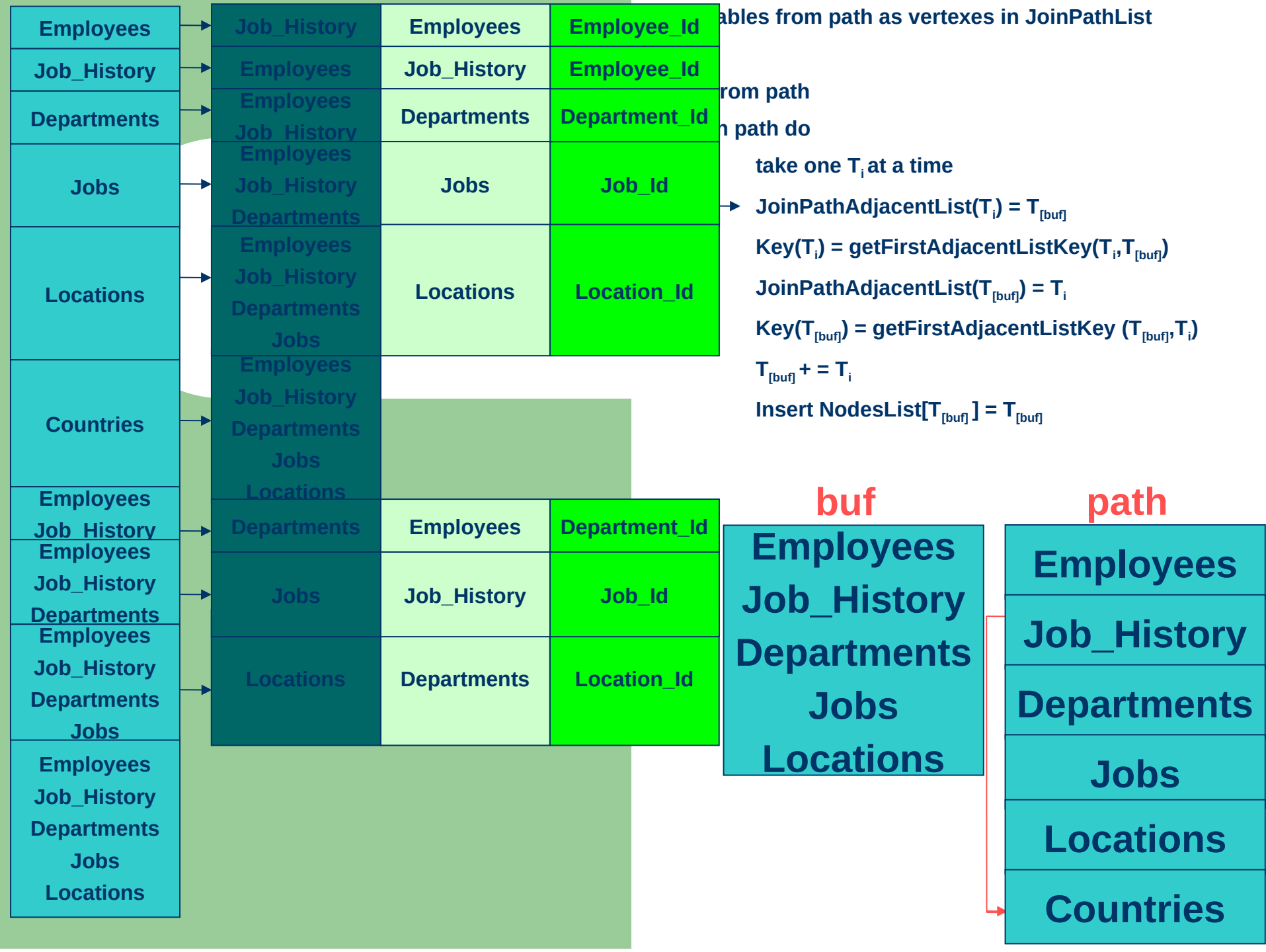
Employees Job_History Departments Jobs Locations			
--	--	--	--

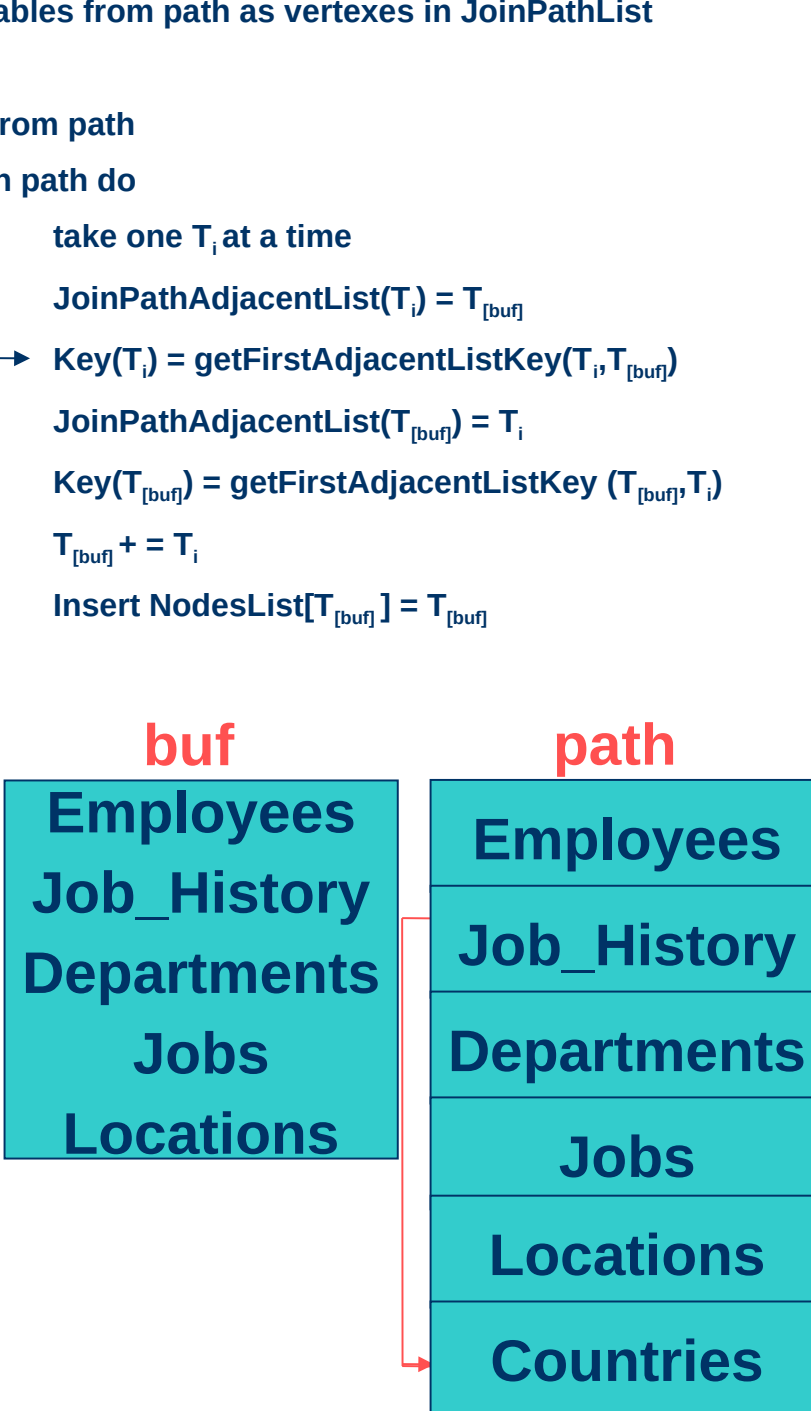
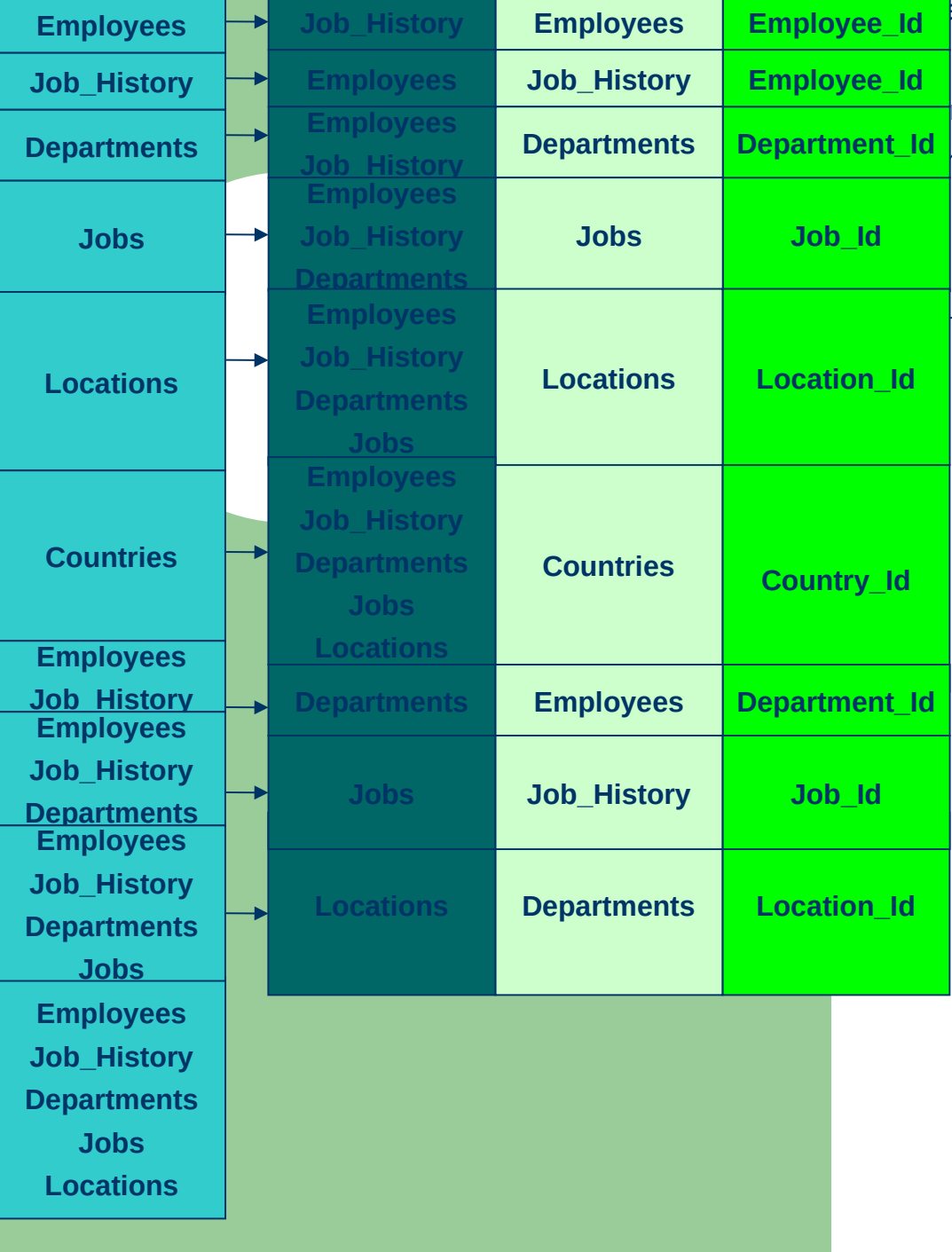
**buf**

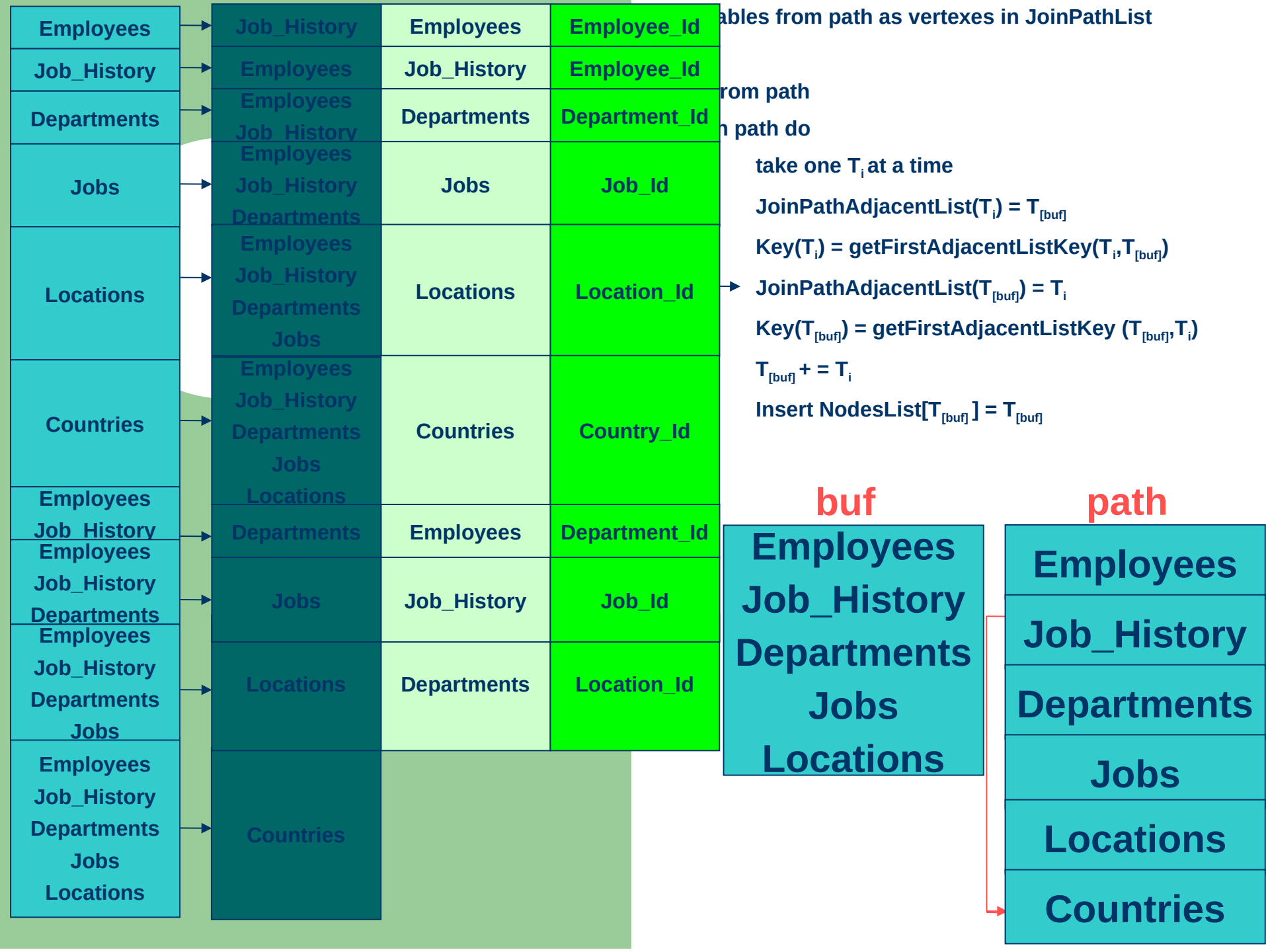
Employees
Job_History
Departments
Jobs
Locations

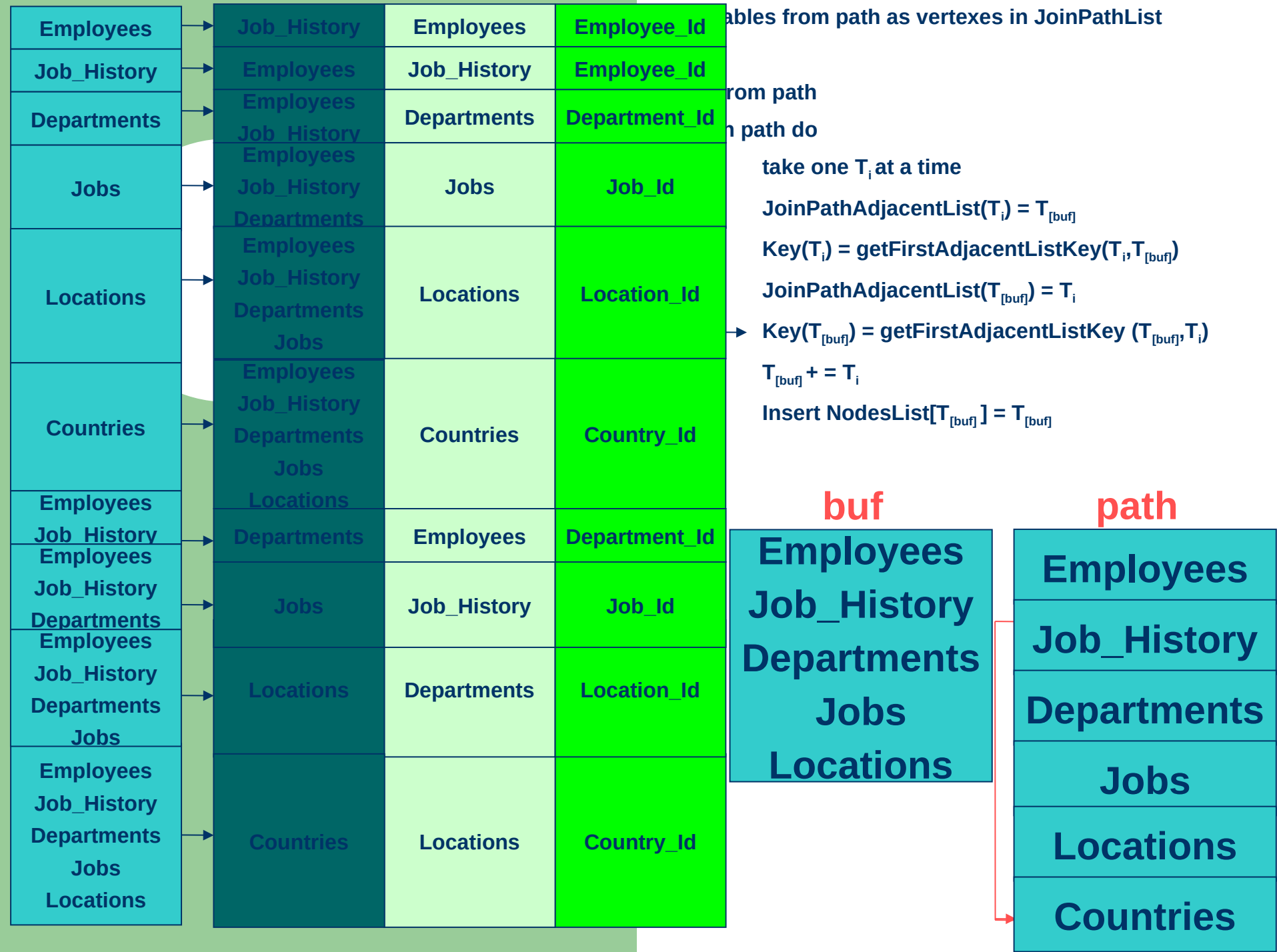
**path**

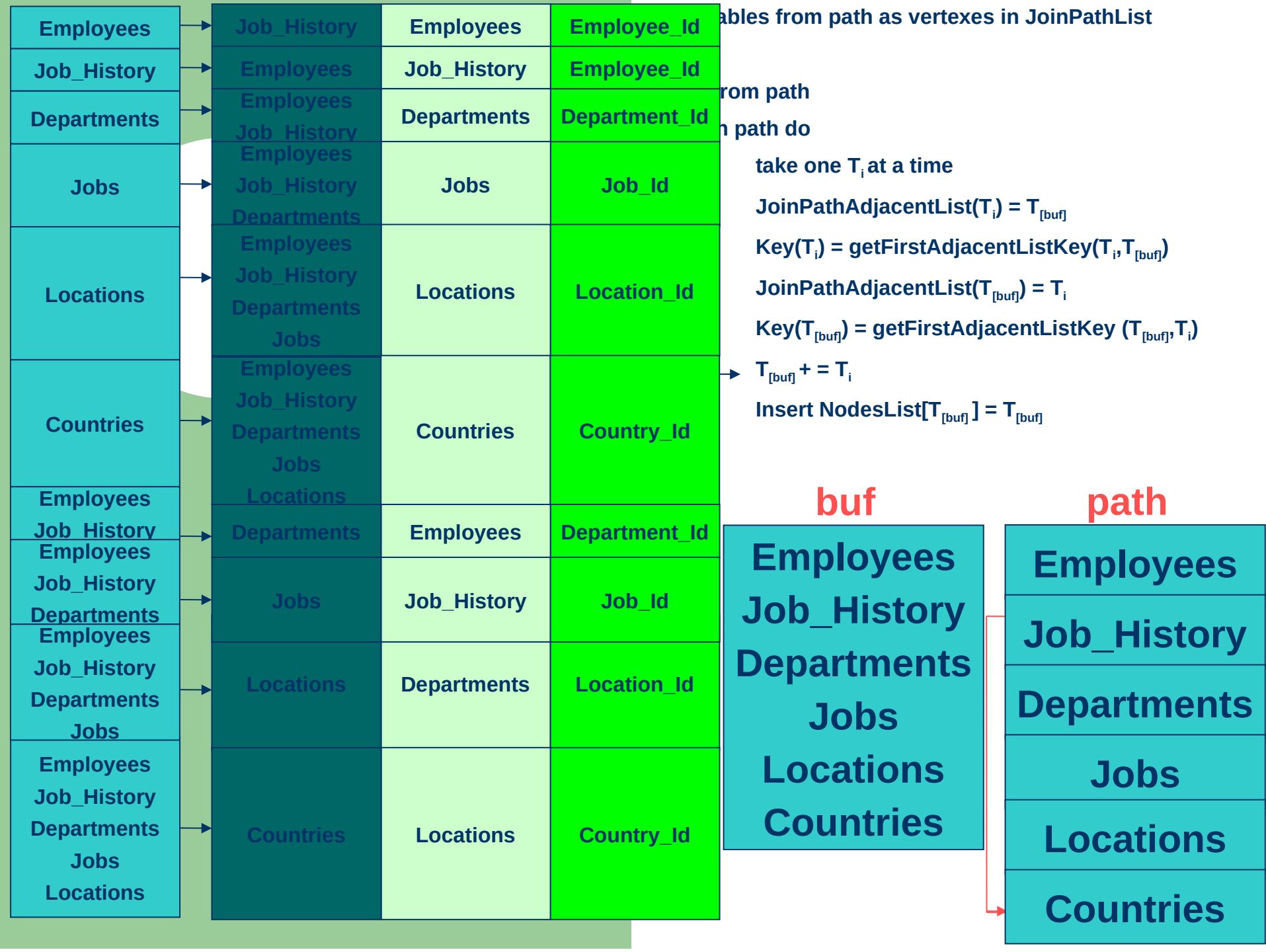
Employees
Job_History
Departments
Jobs
Locations
Countries











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

ables from path as vertexes in JoinPathList

from path

h path do

take one  $T_i$  at a time

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

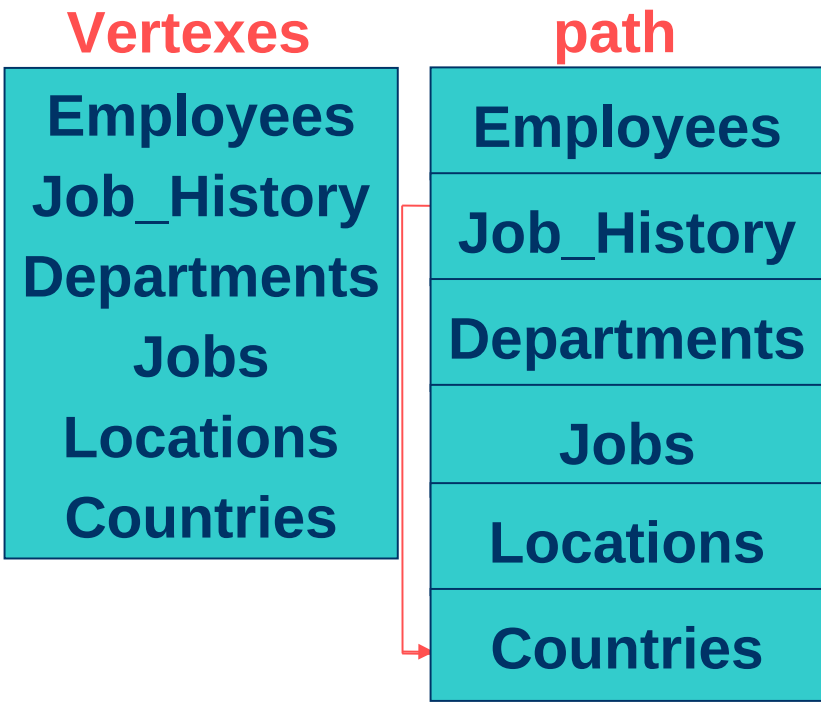
$Key(T_i) = getFirstAdjacentListKey(T_i, T_{[buf]})$

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

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

$T_{[buf]} += T_i$

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



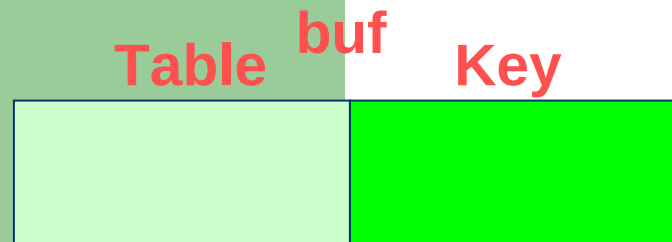


Join Path List

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 I  
                InheritedKey( $T_{[i]}$ ) += buf.key  
    if  $T_i$  is the table from which comes Key( $T_{[i]}$ ) then  
        buf.Table =  $T_i$   
        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

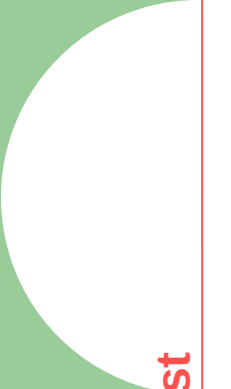
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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



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

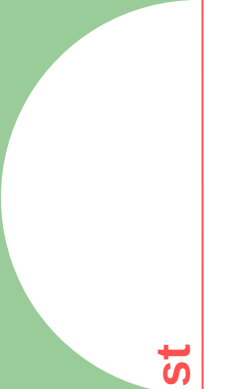
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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 Employees	→	Jobs	Job_History	Job_Id
Departments Employees Job_History	→	Locations	Departments	Location_Id
Jobs Employees Job_History Departments	→	Countries	Locations	Country_Id
Locations Jobs Departments Job_History 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

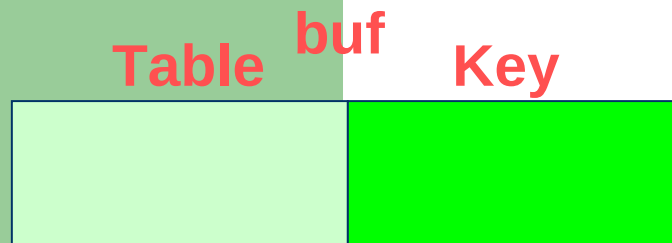
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

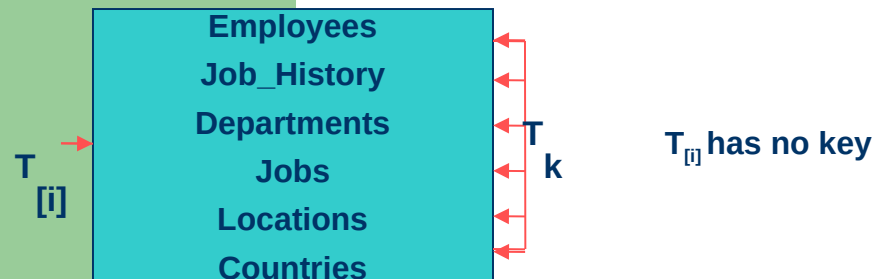
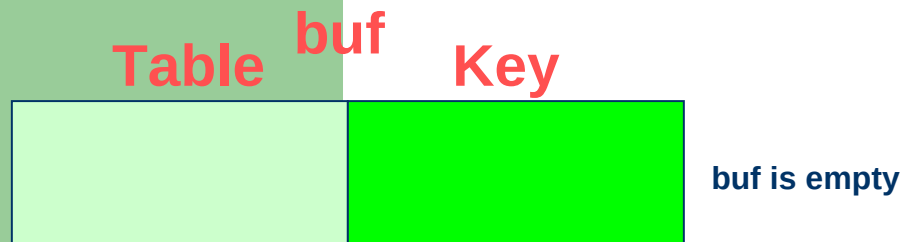
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

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

buf.Table =  $T_i$

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



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 Employees	→	Departments	Employees	Department_Id
Job_History Departments Employees	→	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

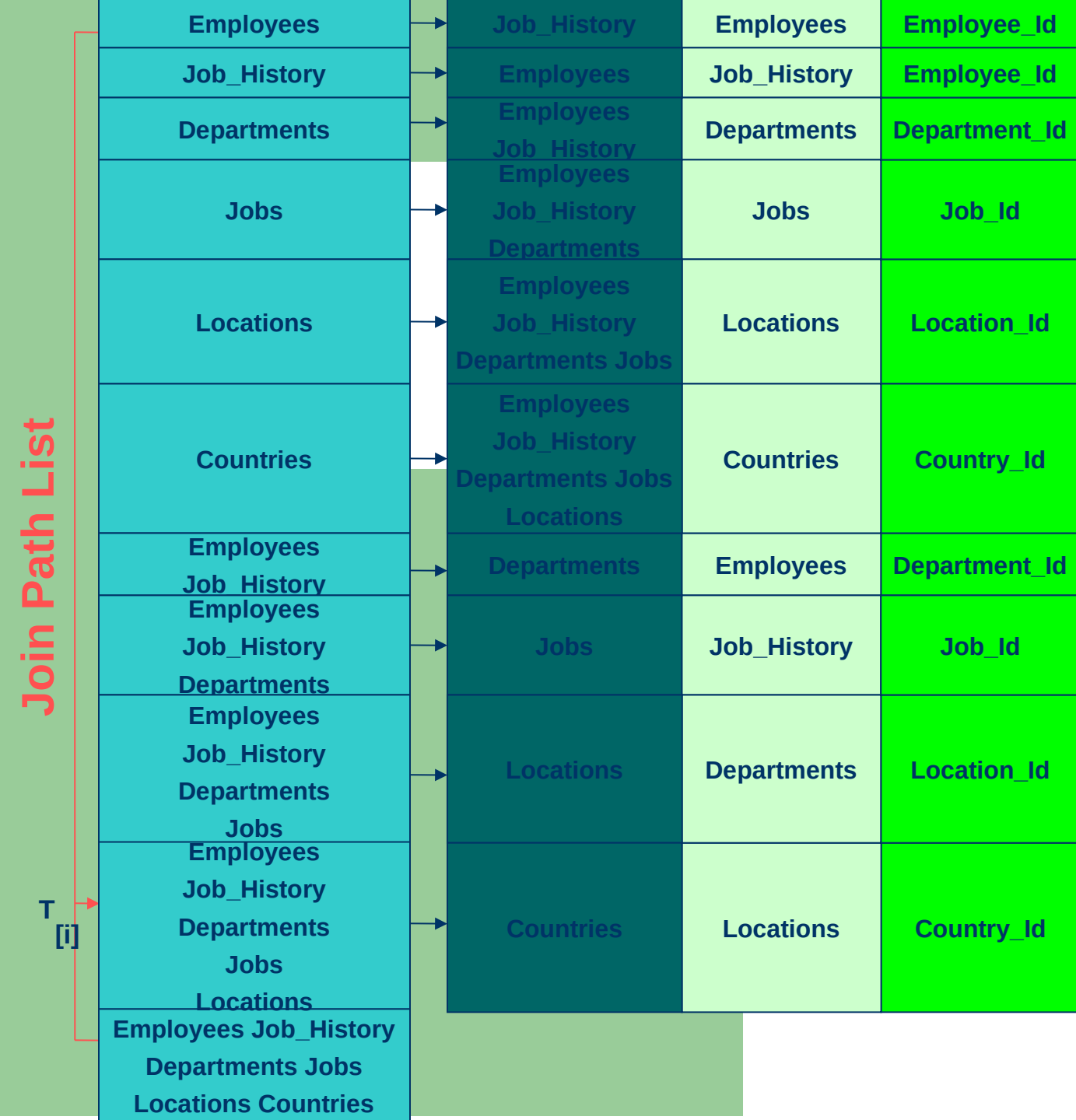
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

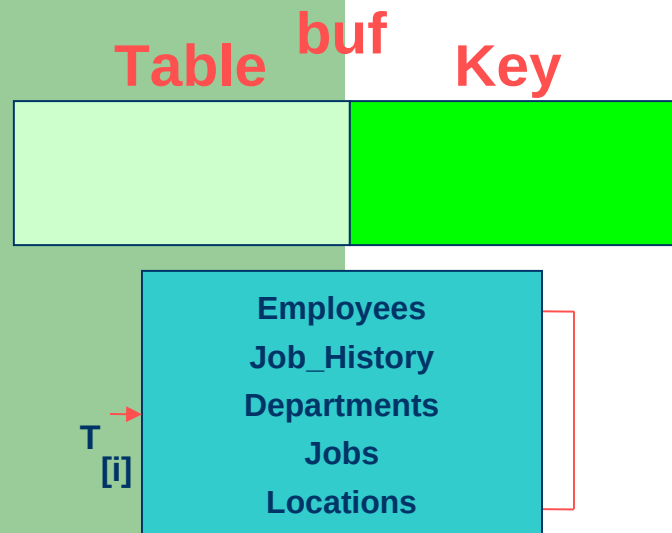
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

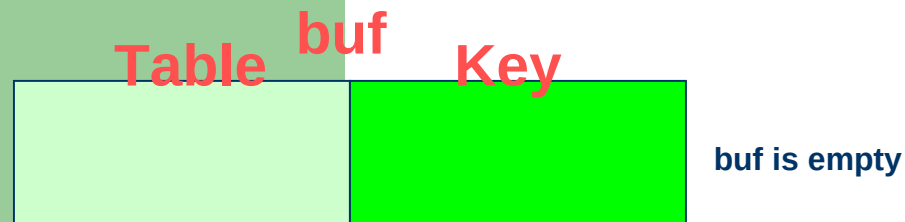
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

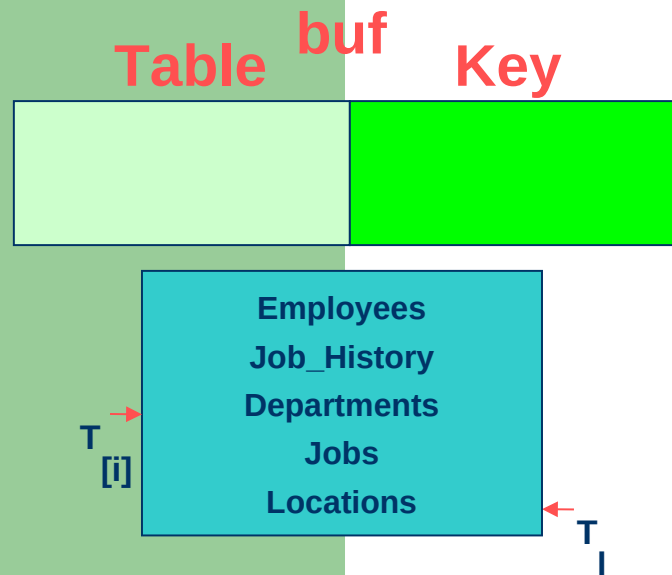
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

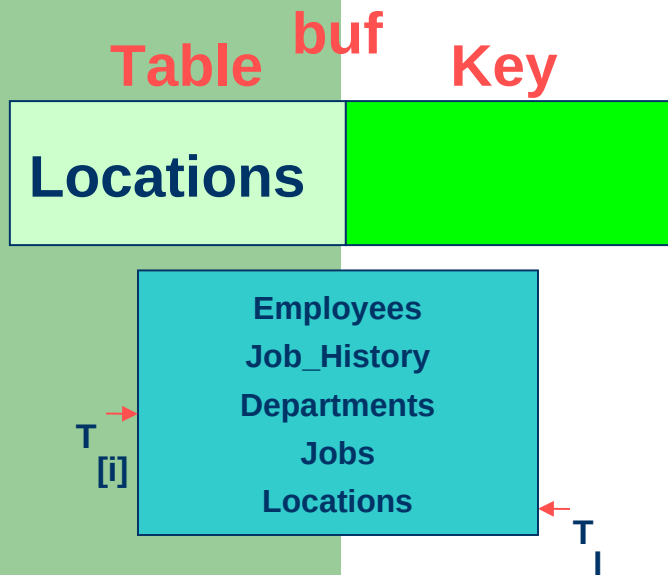
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

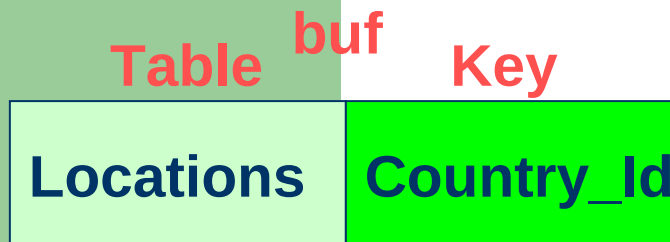
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

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

buf.Table =  $T_i$

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

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

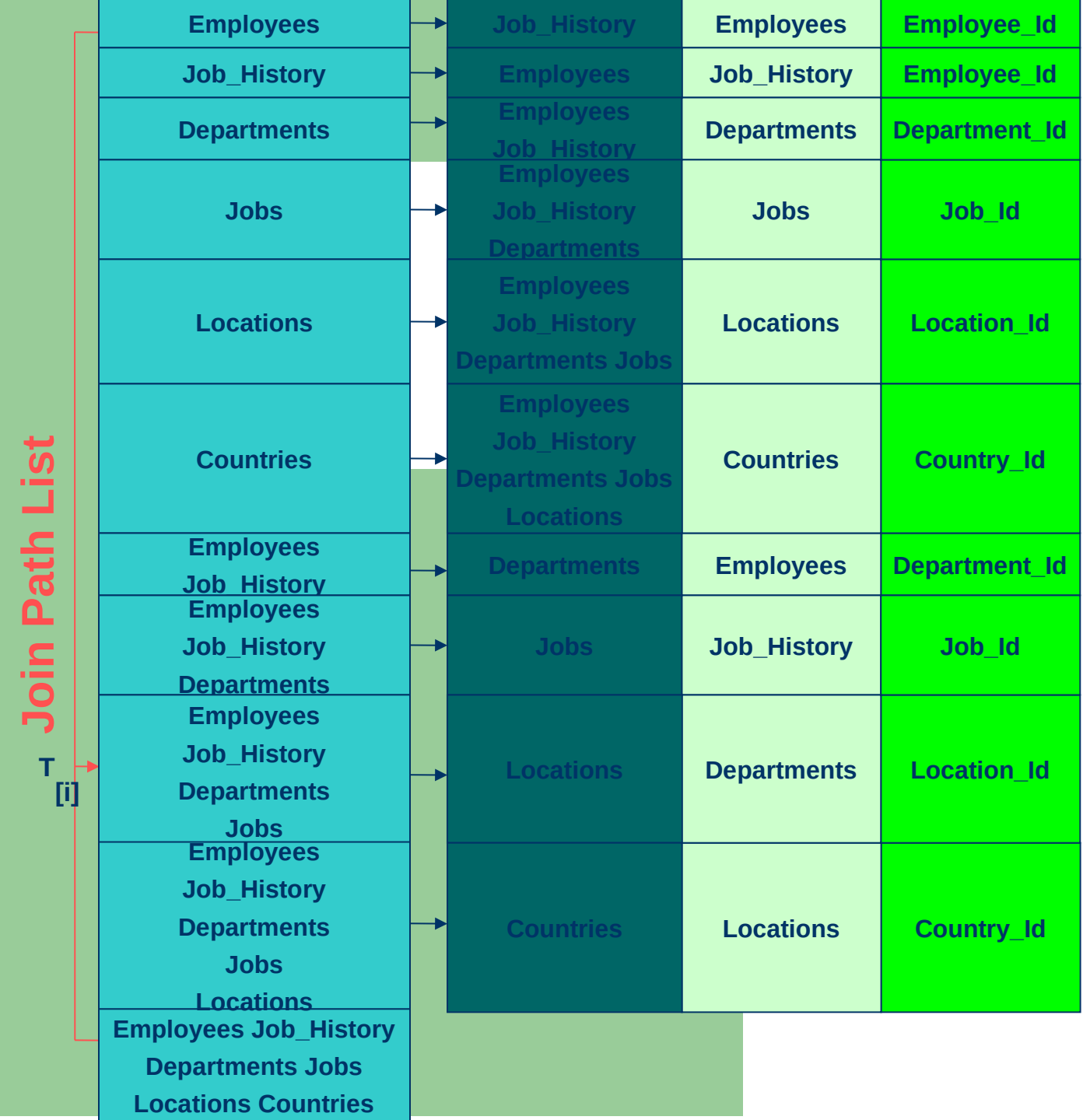
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

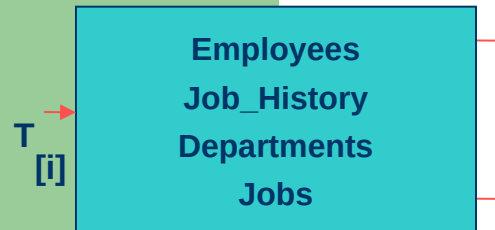
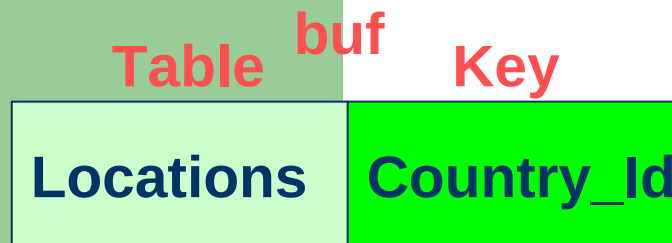
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

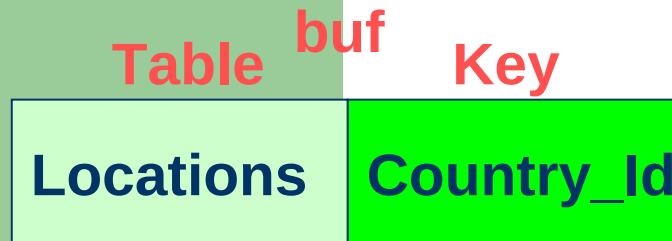
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

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

buf.Table =  $T_i$

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

Table buf Key

Locations

Country\_Id

Locations is not in  $T_{[i]}$

$T_{[i]}$

Employees  
Job\_History  
Departments  
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

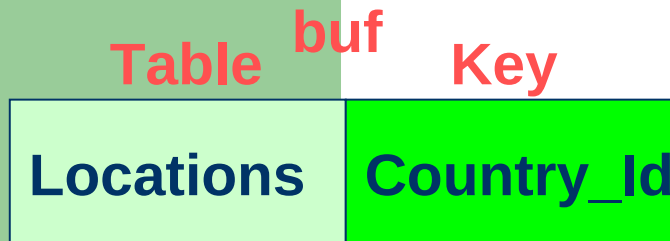
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

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

buf.Table =  $T_i$

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

Table buf Key

Locations

Country\_Id

Departments

Employees  
Job\_History  
Departments  
Jobs

$T_{[i]}$

$T_i$

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

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

buf.Table =  $T_i$

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

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

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

buf.Table =  $T_i$

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

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

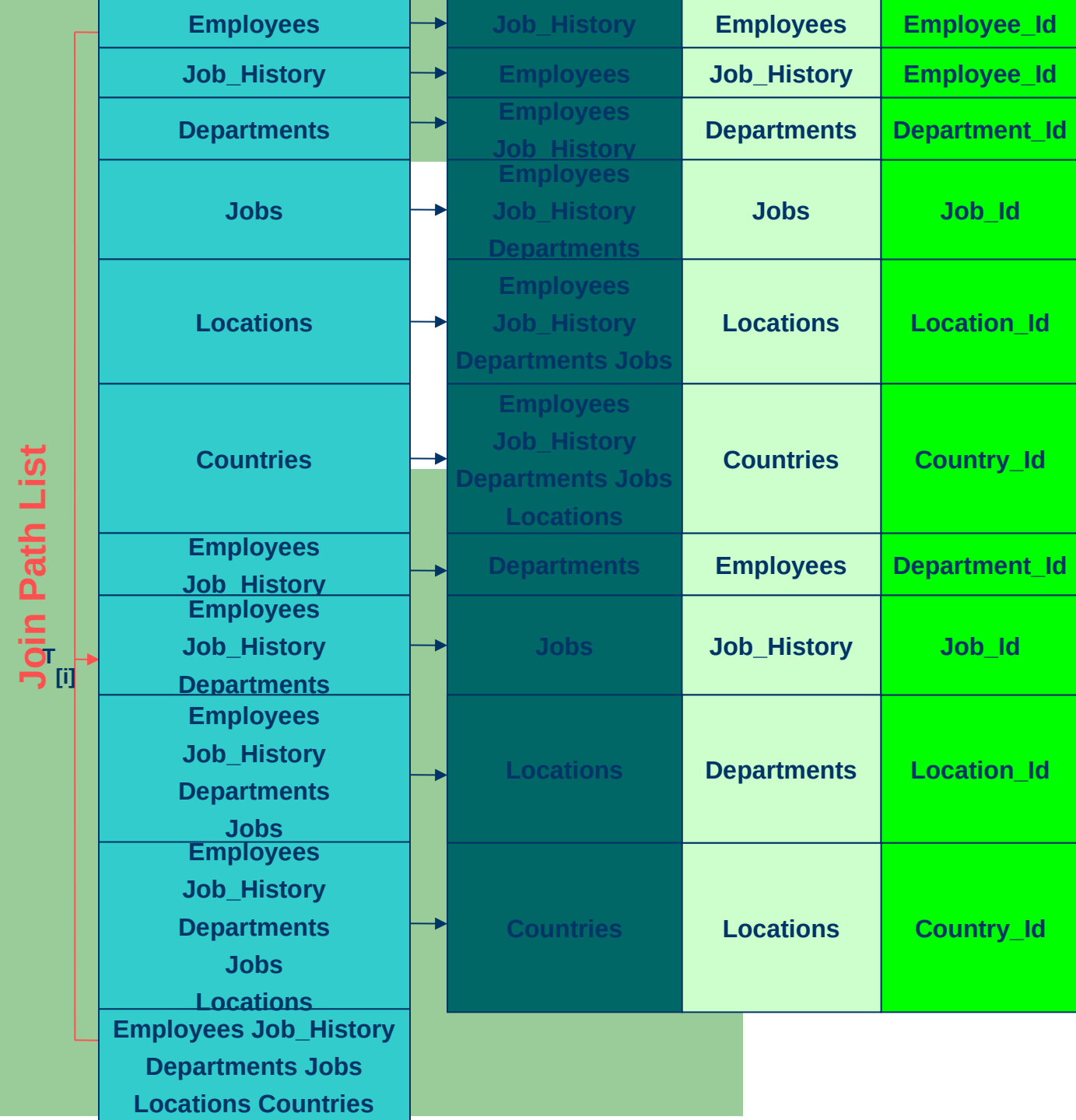
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

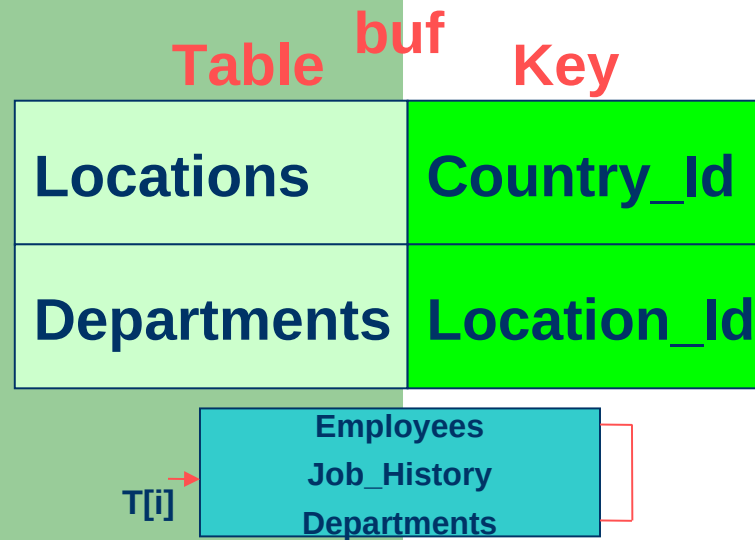
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

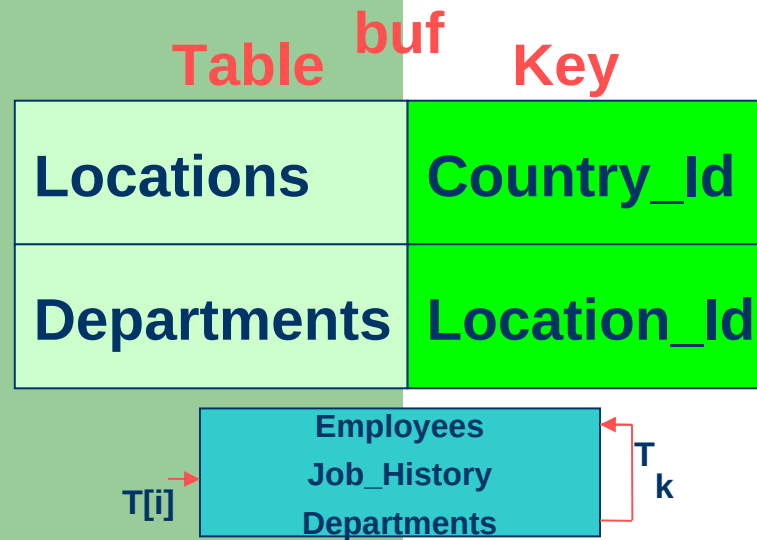
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

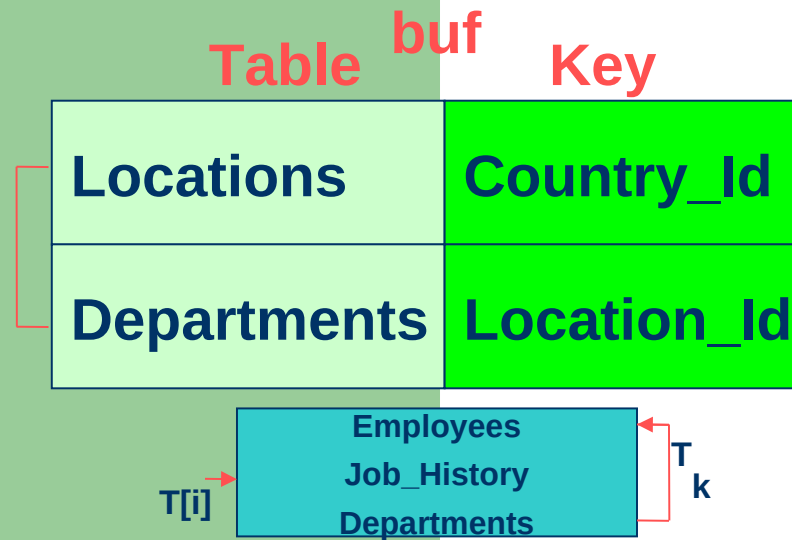
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

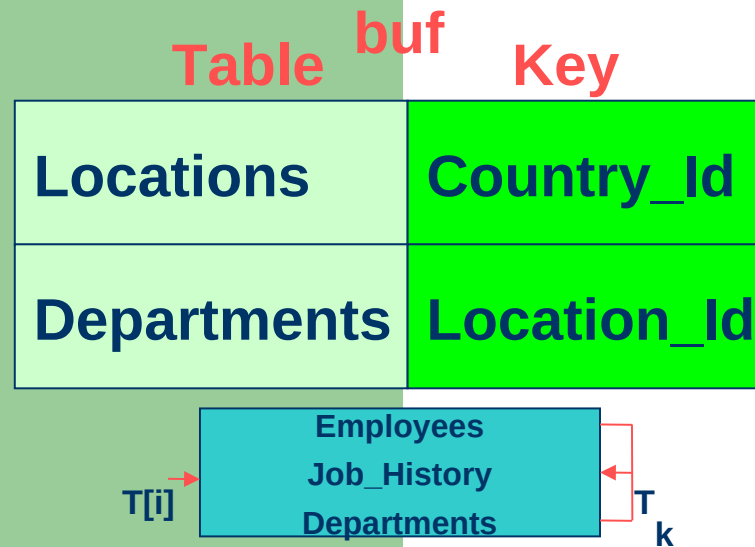
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

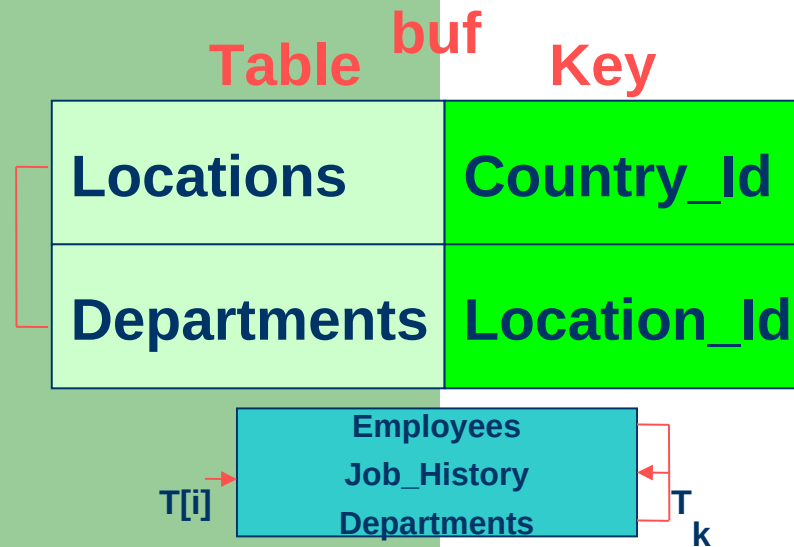
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

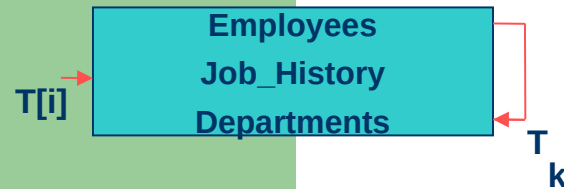
if  $T_i$  is the table from which comes Key( $T_{[i]}$ ) then

buf.Table =  $T_i$

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

Table buf 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

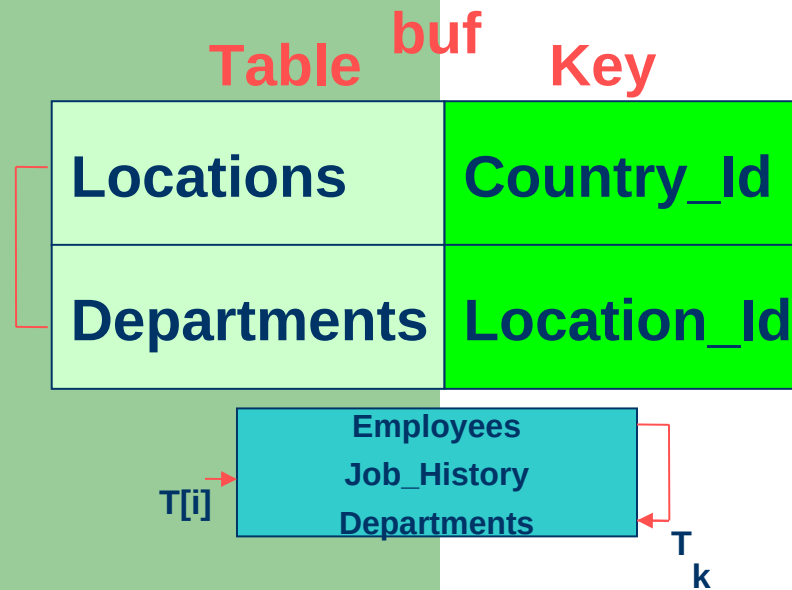
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

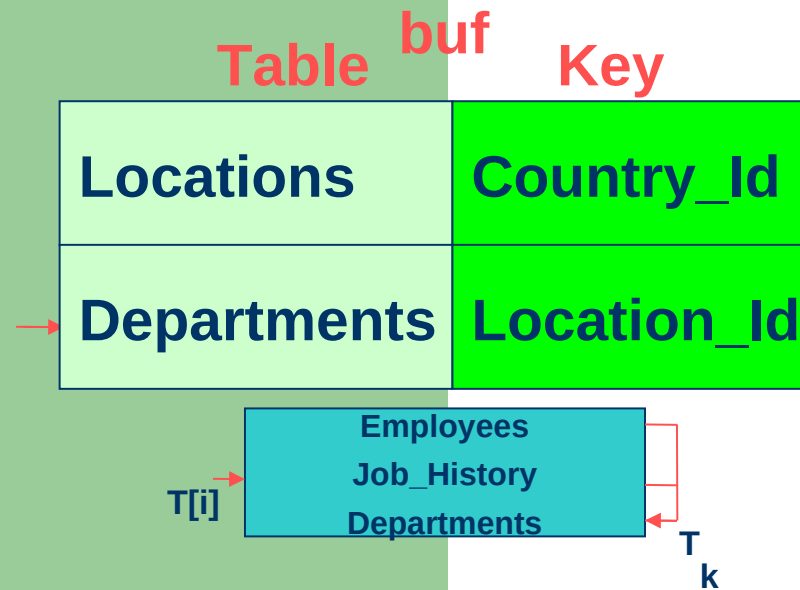
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

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

buf.Table =  $T_i$

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

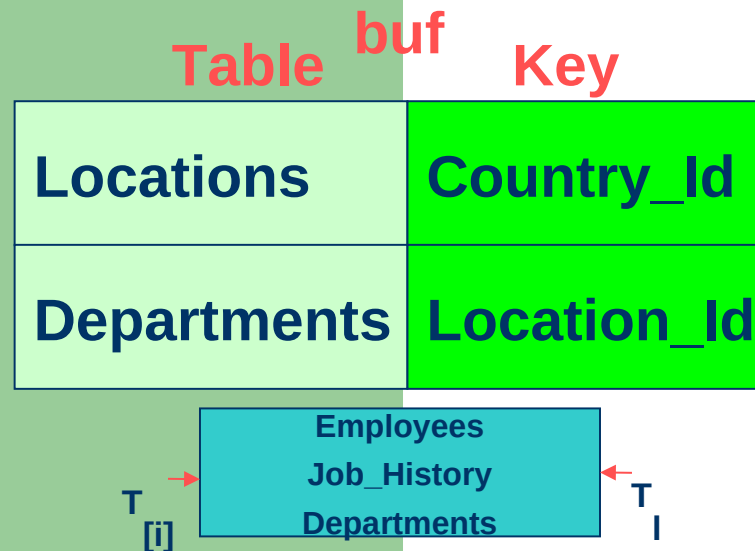
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

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

buf.Table =  $T_i$

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

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

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

buf.Table =  $T_i$

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

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

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

buf.Table =  $T_i$

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

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

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

buf.Table =  $T_i$

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

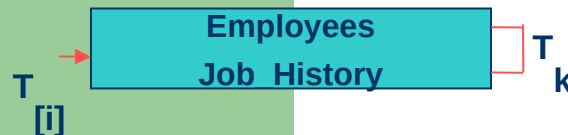
if (buf.key != Key( $T_{[i]}$ )) and (buf.Key not in InheritedKey( $T_{[i]}$ )) then I  
InheritedKey( $T_{[i]}$ ) += buf.key

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

buf.Table =  $T_i$

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

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

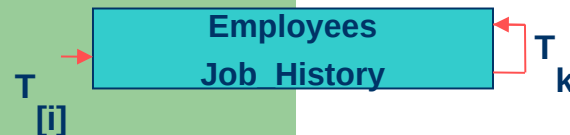
if (buf.key != Key( $T_{[i]}$ )) and (buf.Key not in InheritedKey( $T_{[i]}$ )) then I  
InheritedKey( $T_{[i]}$ ) += buf.key

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

buf.Table =  $T_i$

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

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

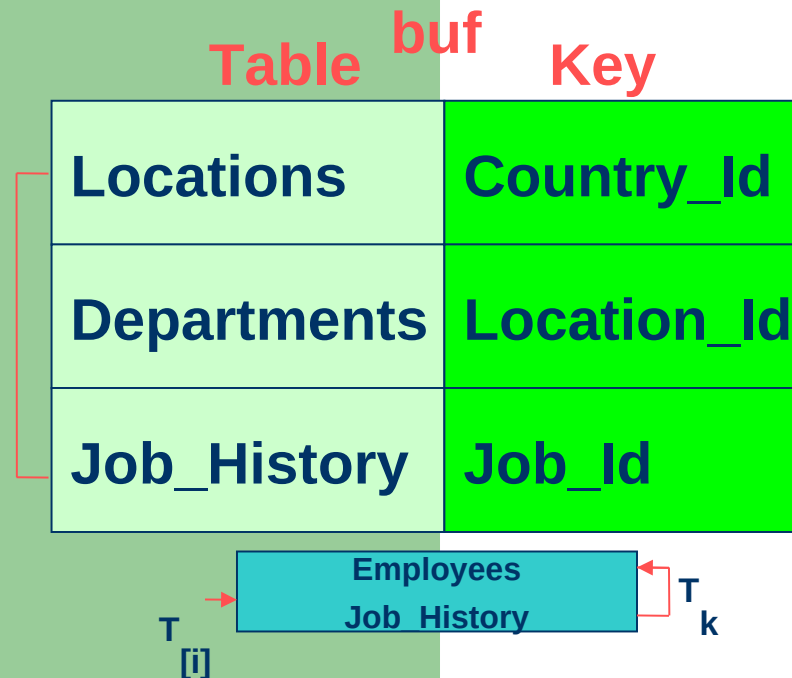
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

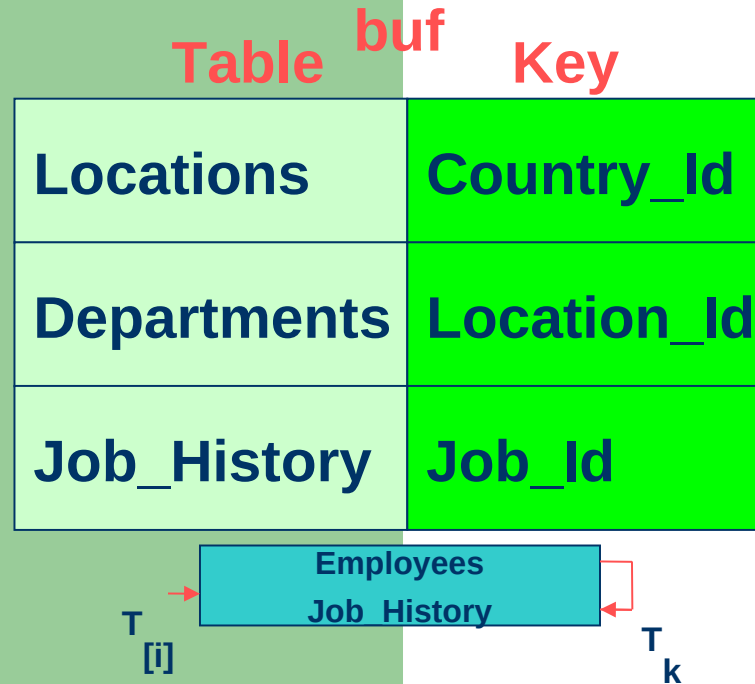
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

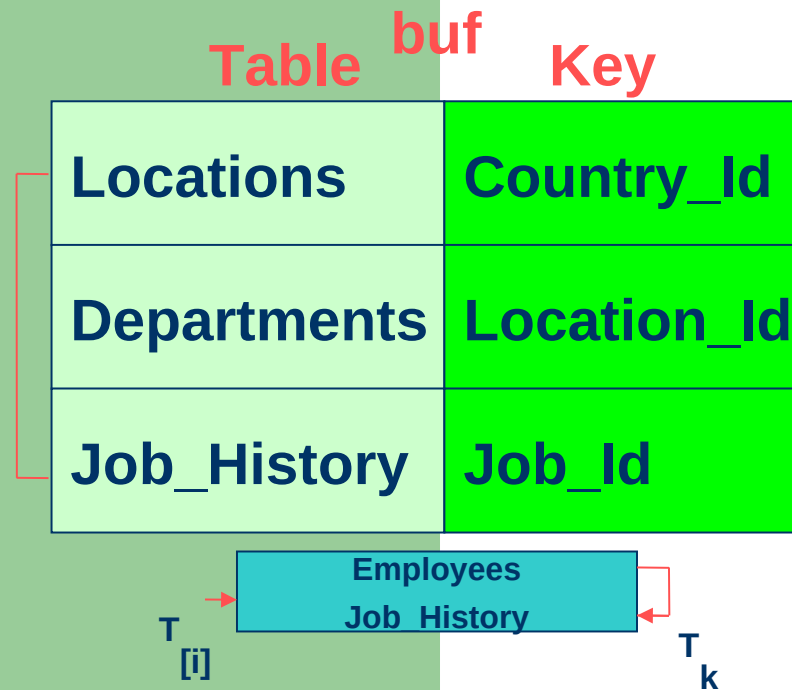
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

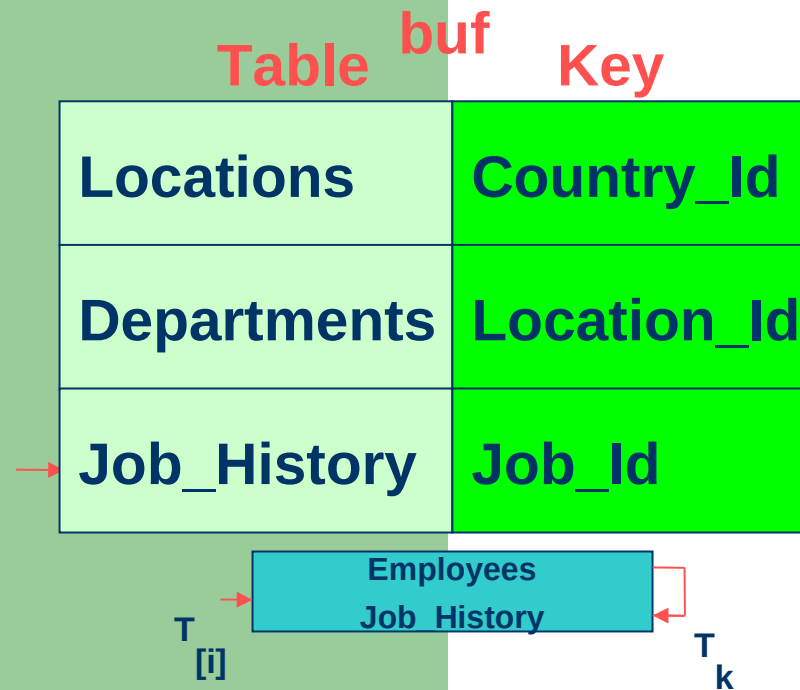
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

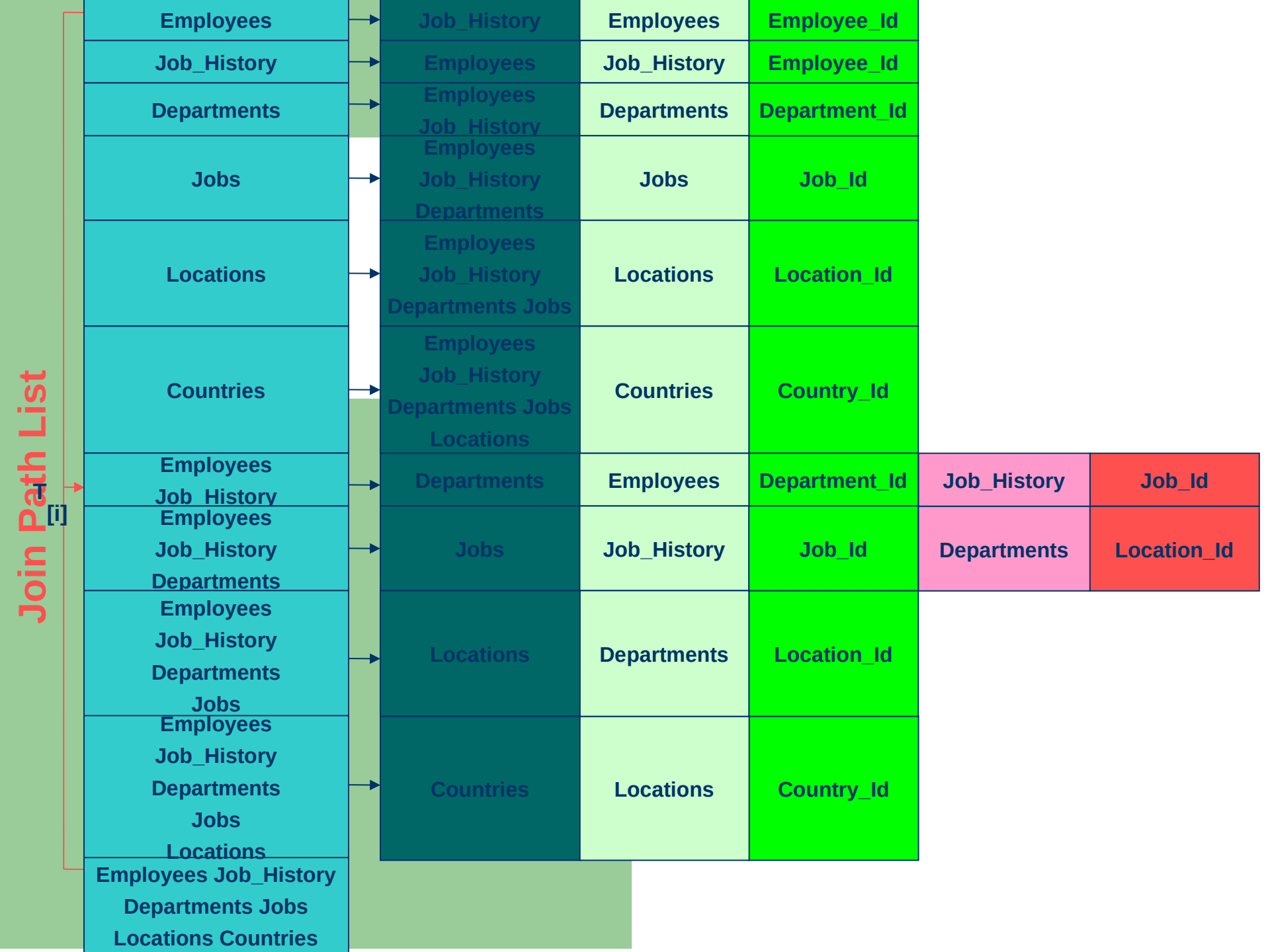
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

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

buf.Table =  $T_i$

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

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

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

buf.Table =  $T_i$

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

Table buf 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  $T_i$  is the table from which comes Key( $T_{[i]}$ ) then

buf.Table =  $T_i$

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

Table buf 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

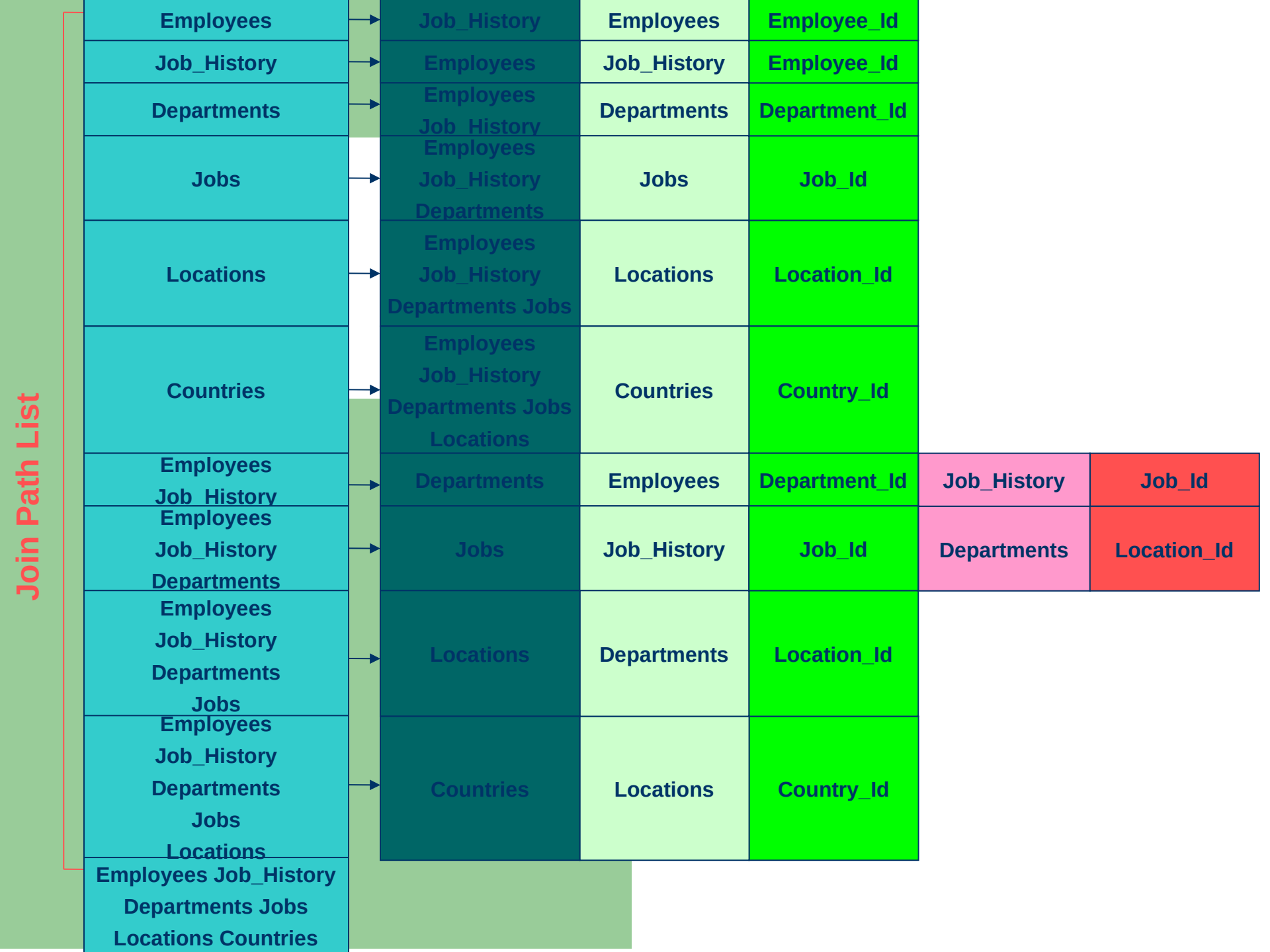
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

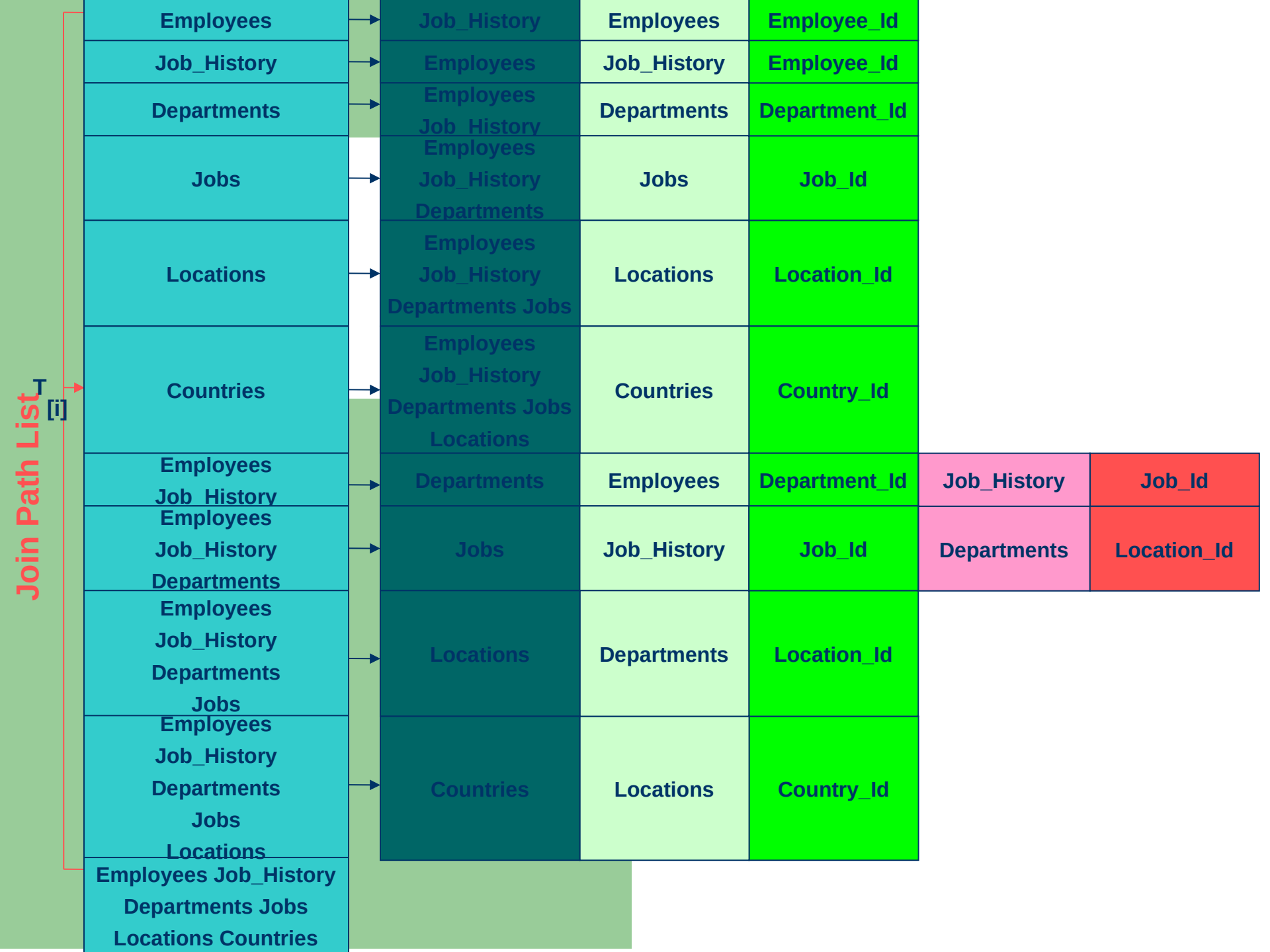
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

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

buf.Table =  $T_i$

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

Table buf 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  $T_i$  is the table from which comes Key( $T_{[i]}$ ) then

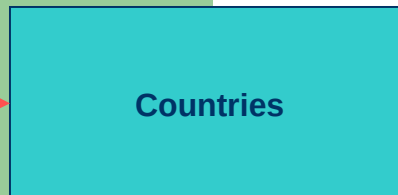
buf.Table =  $T_i$

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

Table buf 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

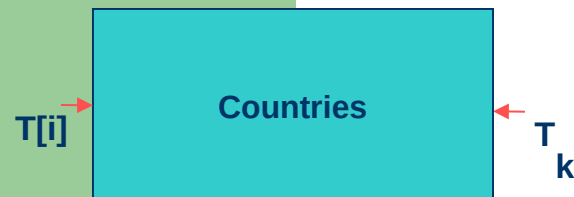
if (buf.key != Key( $T_{[i]}$ )) and (buf.Key not in InheritedKey( $T_{[i]}$ )) then  
InheritedKey( $T_{[i]}$ ) += buf.key

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

buf.Table =  $T_i$

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

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

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

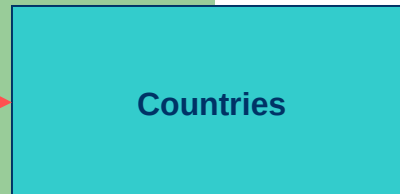
buf.Table =  $T_i$

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

Table buf 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

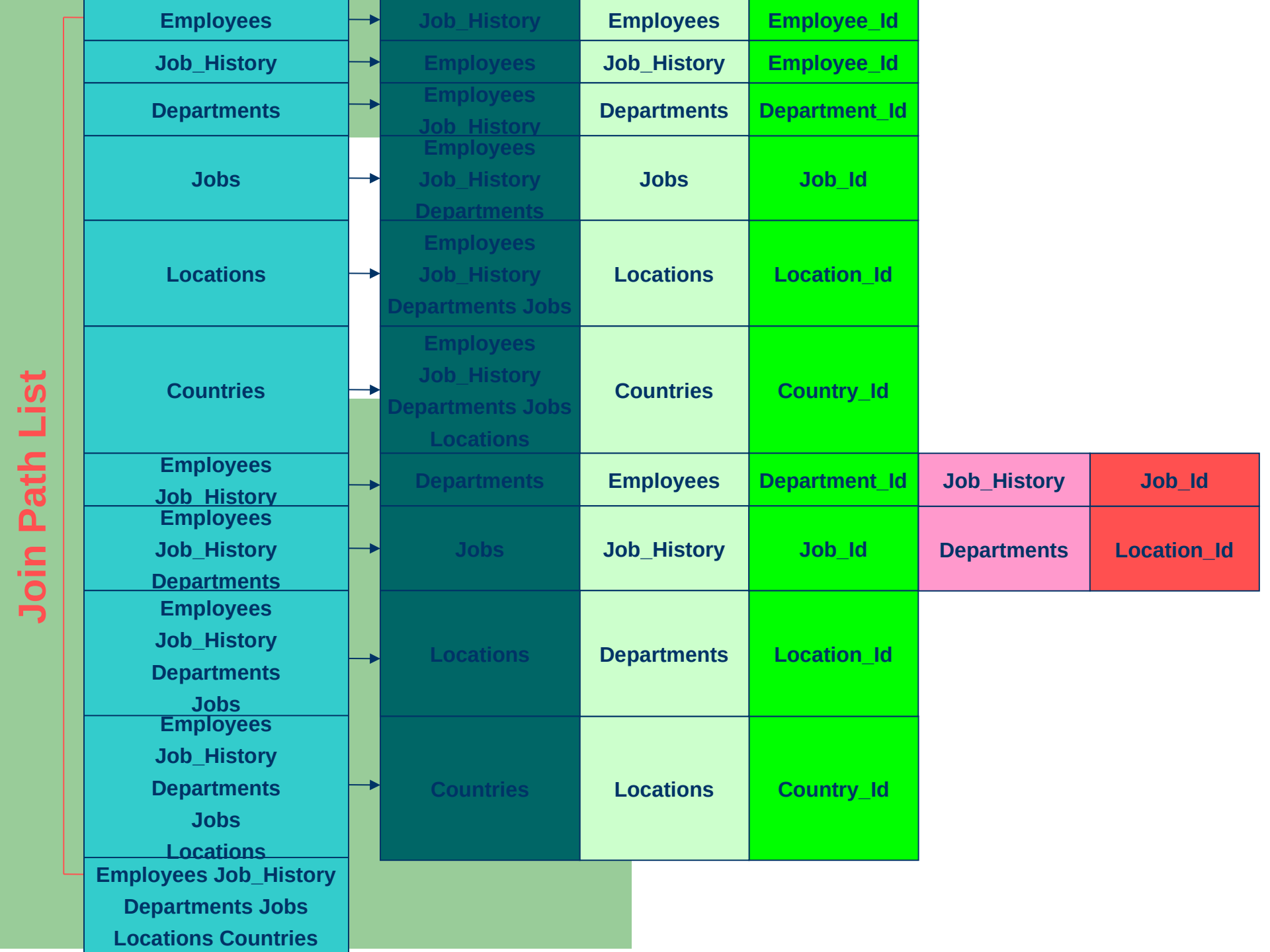
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

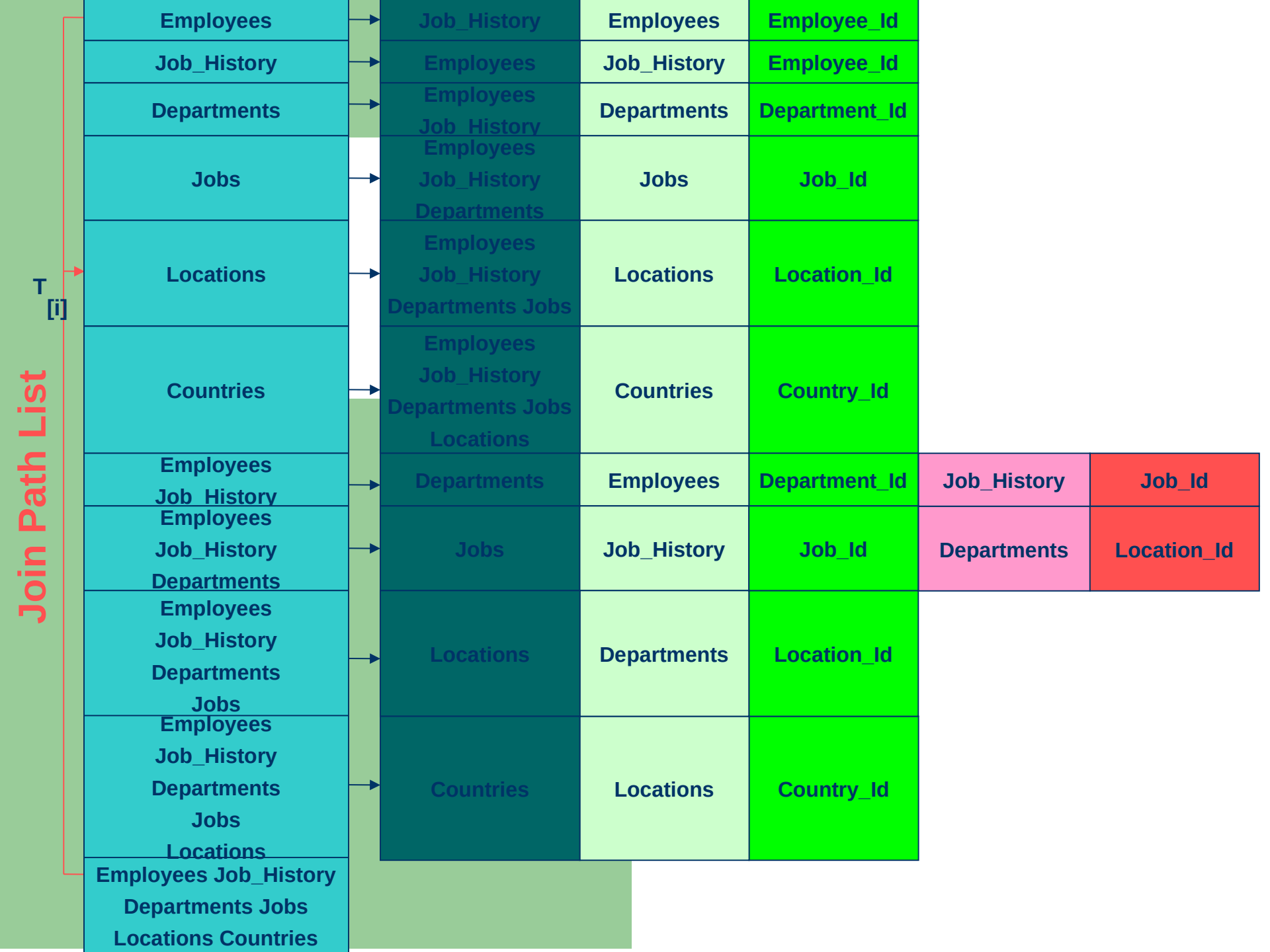
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

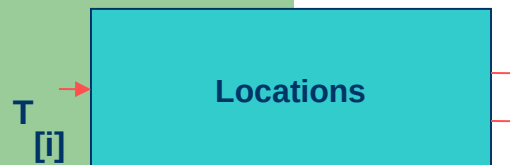
if  $T_i$  is the table from which comes Key( $T_{[i]}$ ) then

buf.Table =  $T_i$

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

Table buf 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

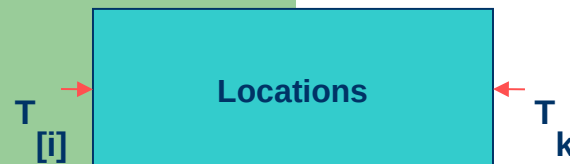
if  $T_i$  is the table from which comes Key( $T_{[i]}$ ) then

buf.Table =  $T_i$

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

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

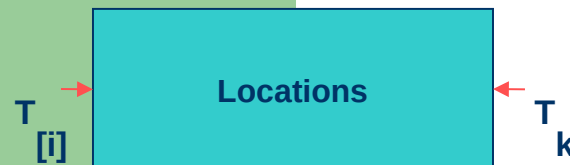
if  $T_i$  is the table from which comes Key( $T_{[i]}$ ) then

buf.Table =  $T_i$

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

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

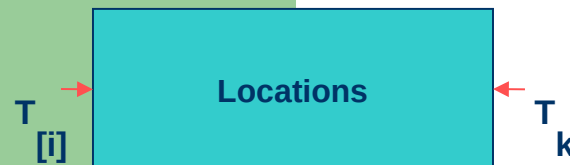
if  $T_i$  is the table from which comes Key( $T_{[i]}$ ) then

buf.Table =  $T_i$

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

Table buf 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  $\neq$  Key( $T_{[i]}$ ) ) and (buf.Key not in InheritedKey( $T_{[i]}$ )) then I  
InheritedKey( $T_{[i]}$ ) += buf.key

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

buf.Table =  $T_i$

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

Path Join 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	Locations	Country_Id
Countries	→	Employees Job_History Departments Jobs Locations	Countries	Country_Id		
Employees Job_History Employees	→	Departments	Employees	Department_Id	Job_History	Job_Id
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 I  
InheritedKey( $T_{[i]}$ ) += buf.key

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

buf.Table =  $T_i$

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

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	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

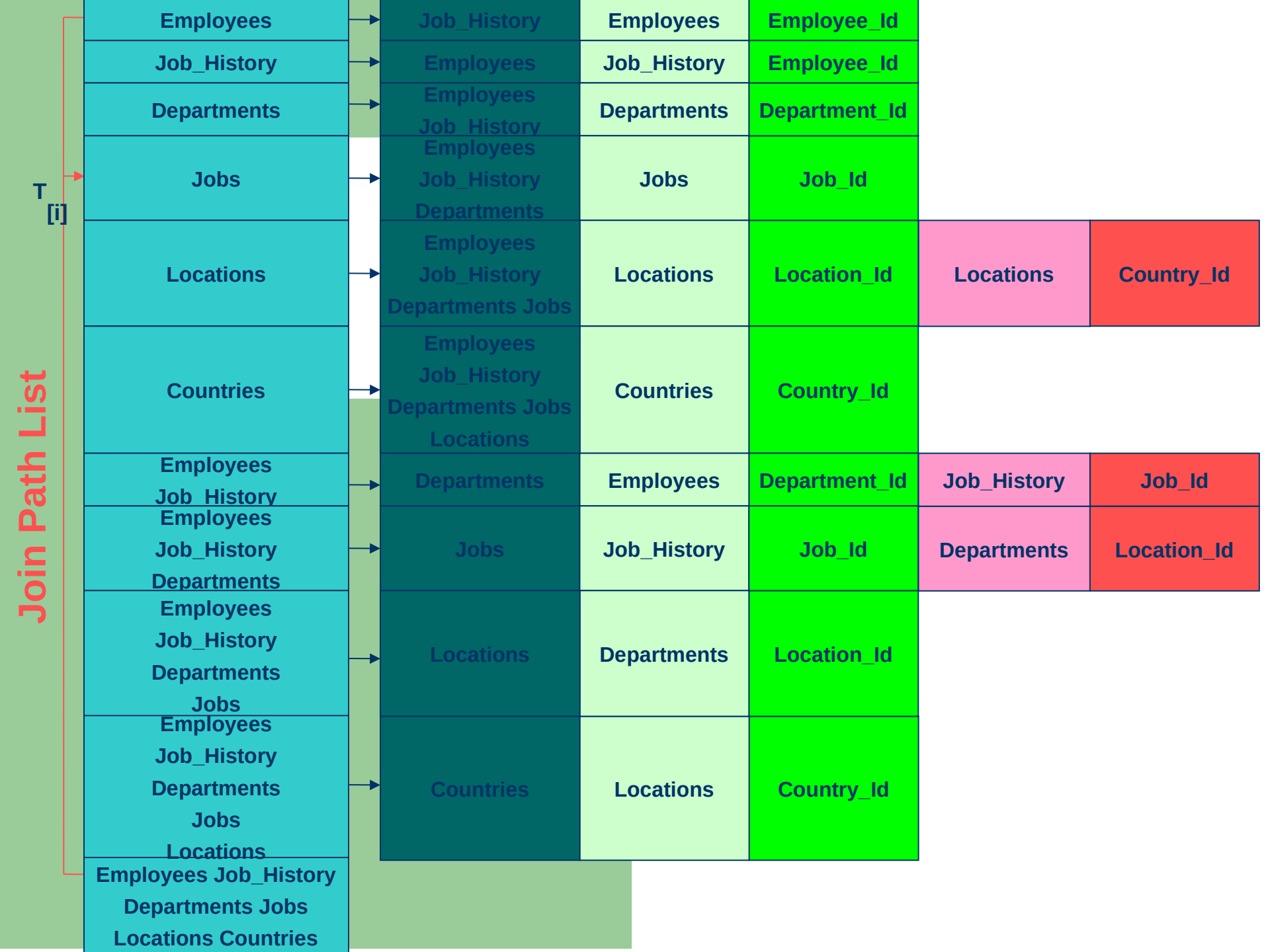
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

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

buf.Table =  $T_i$

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

Table buf 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  $T_i$  is the table from which comes Key( $T_{[i]}$ ) then

buf.Table =  $T_i$

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

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

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

buf.Table =  $T_i$

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

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

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

buf.Table =  $T_i$

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

Table buf 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  $\neq$  Key( $T_{[i]}$ ) ) and (buf.Key not in InheritedKey( $T_{[i]}$ )) then I  
InheritedKey( $T_{[i]}$ ) += buf.key

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

buf.Table =  $T_i$

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

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	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

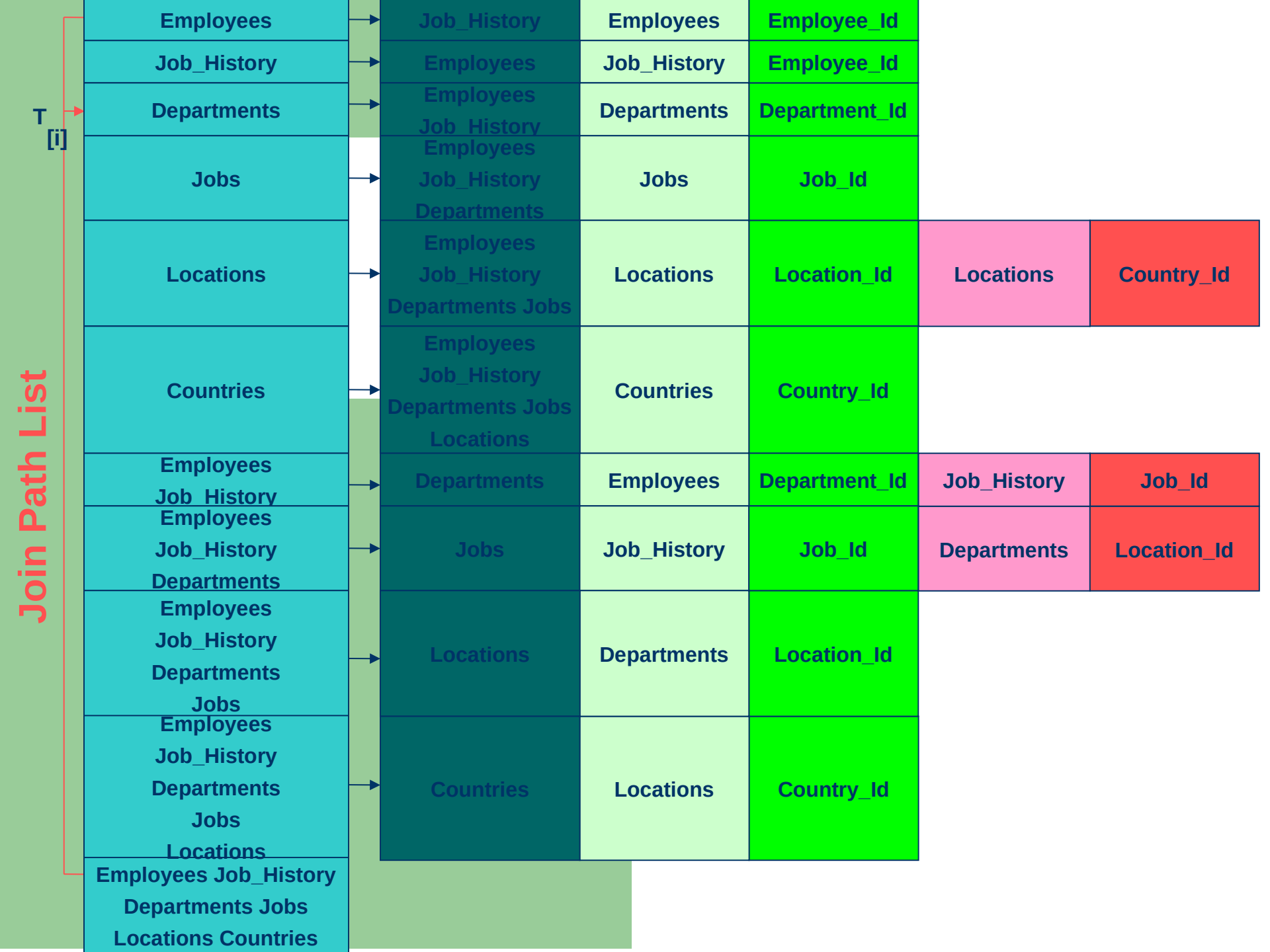
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

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

buf.Table =  $T_i$

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

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

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

buf.Table =  $T_i$

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

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

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

buf.Table =  $T_i$

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

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

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

buf.Table =  $T_i$

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

Table buf 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

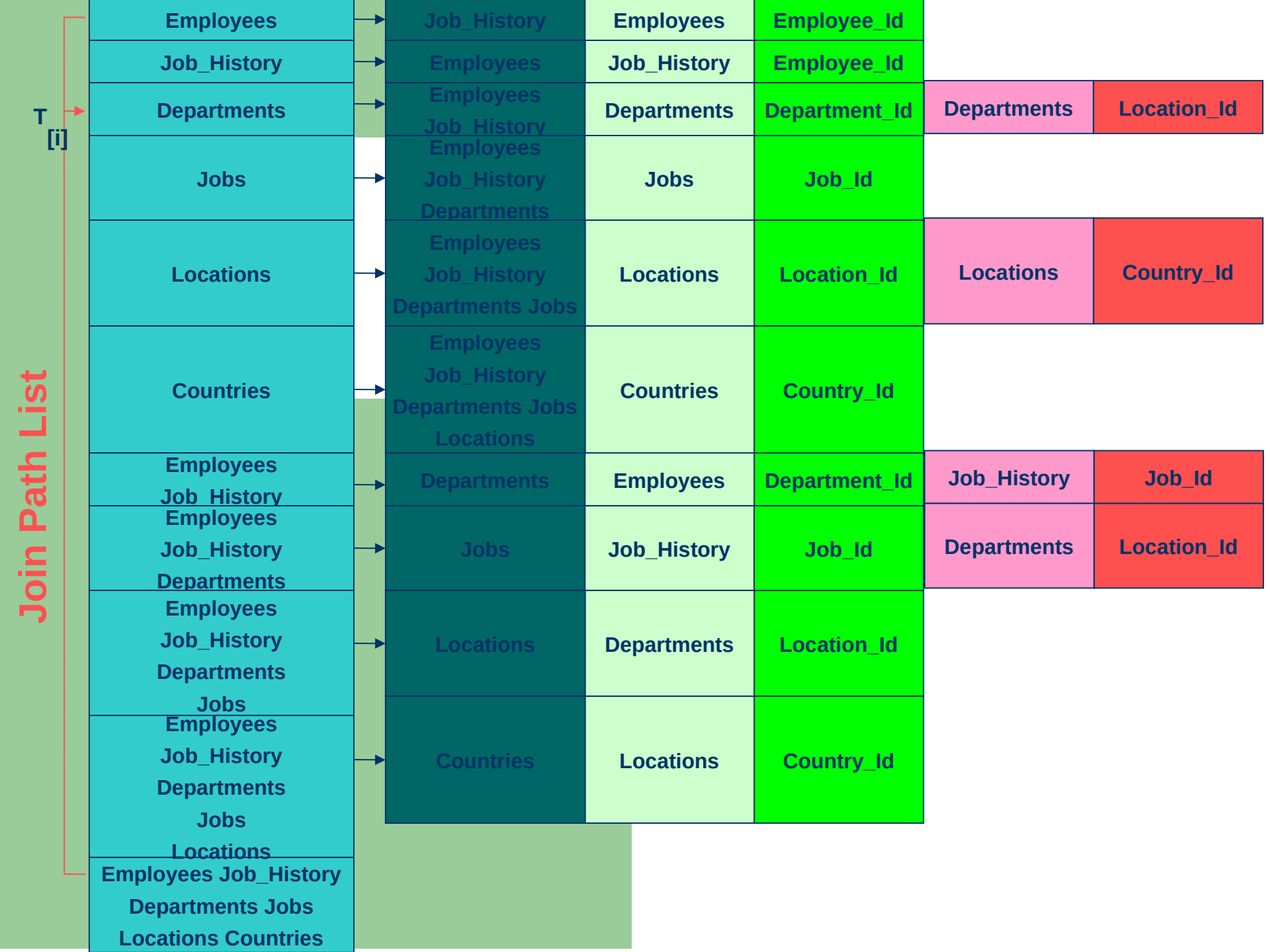
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

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

buf.Table =  $T_i$

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

Table buf Key

Locations	Country_Id
Departments	Location_Id
Job_History	Job_Id
Employees	Department_Id

$T_{[i]}$



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

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

buf.Table =  $T_i$

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



Join Path List

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

create a structure buf with 2 fields: Table and Key  
for all the tables in JoinPathList going downward do

→ take one  $T_{[i]}$  at a time

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

take one  $T_k$  at a time

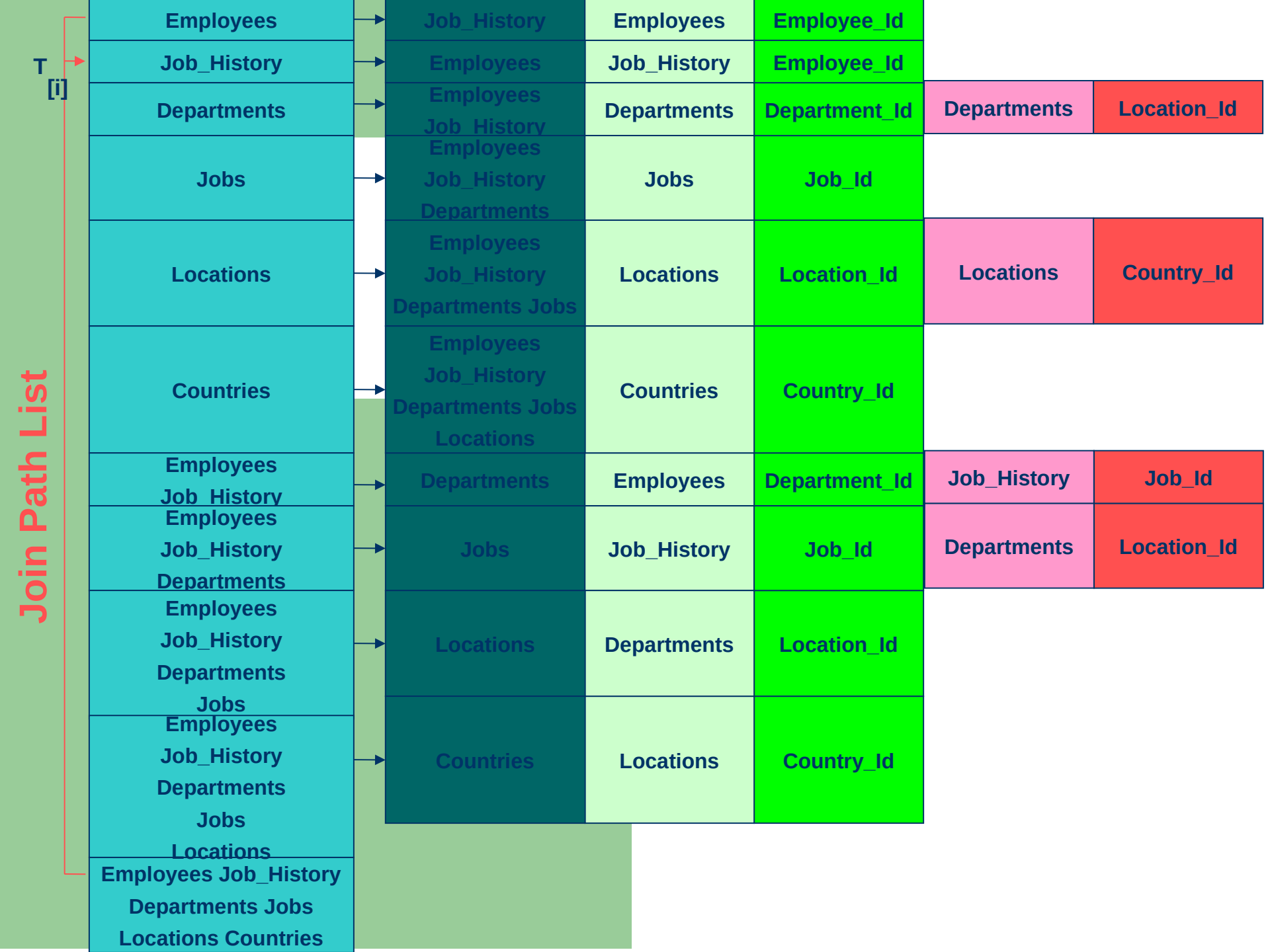
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

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

buf.Table =  $T_i$

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

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

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

buf.Table =  $T_i$

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

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

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

buf.Table =  $T_i$

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

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

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

buf.Table =  $T_i$

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

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

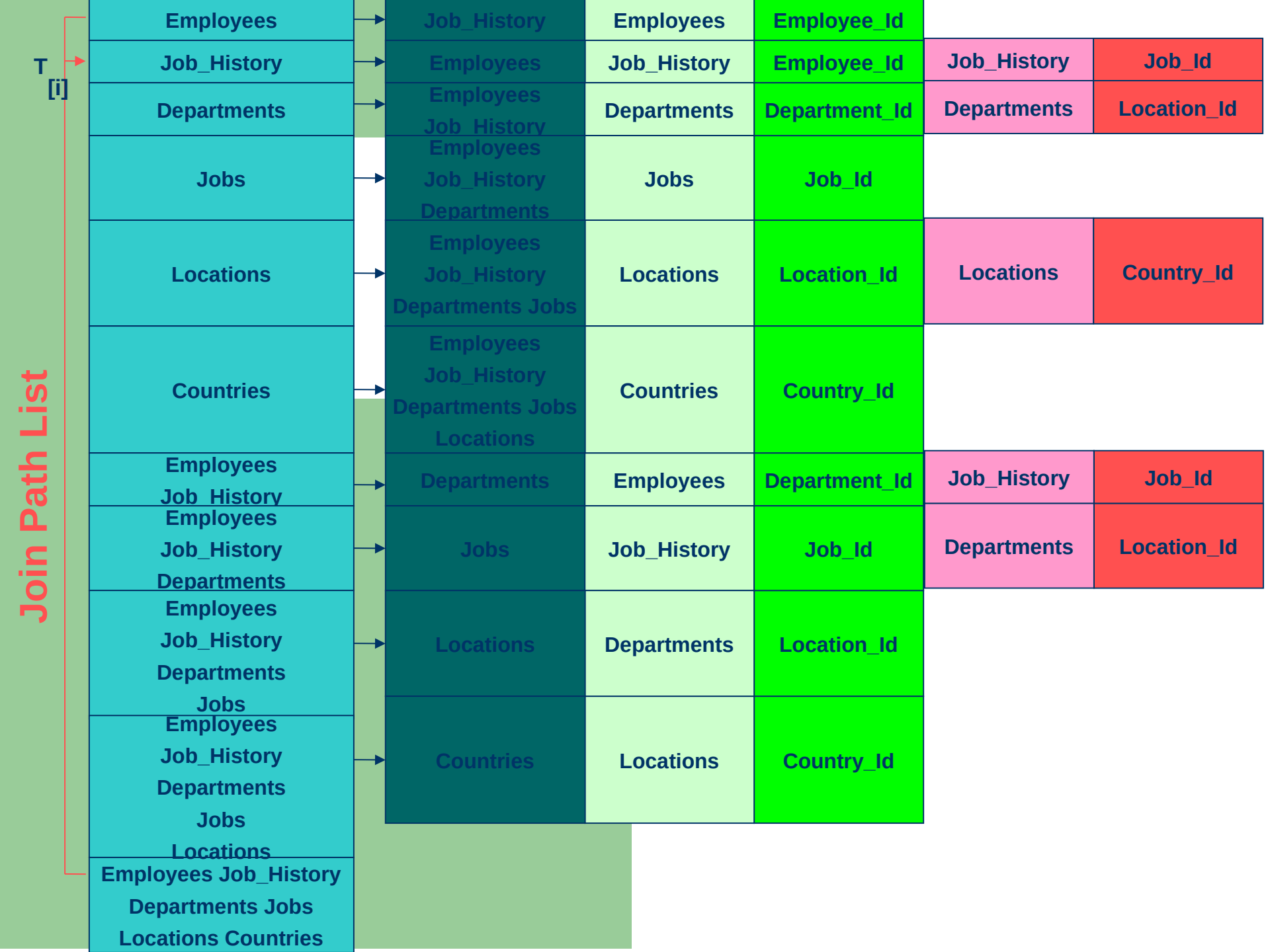
if (buf.key  $\neq$  Key( $T_{[i]}$ ) ) and (buf.Key not in InheritedKey( $T_{[i]}$ )) then I  
InheritedKey( $T_{[i]}$ ) += buf.key

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

buf.Table =  $T_i$

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

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

buf.Table =  $T_i$

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

Join Path List

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

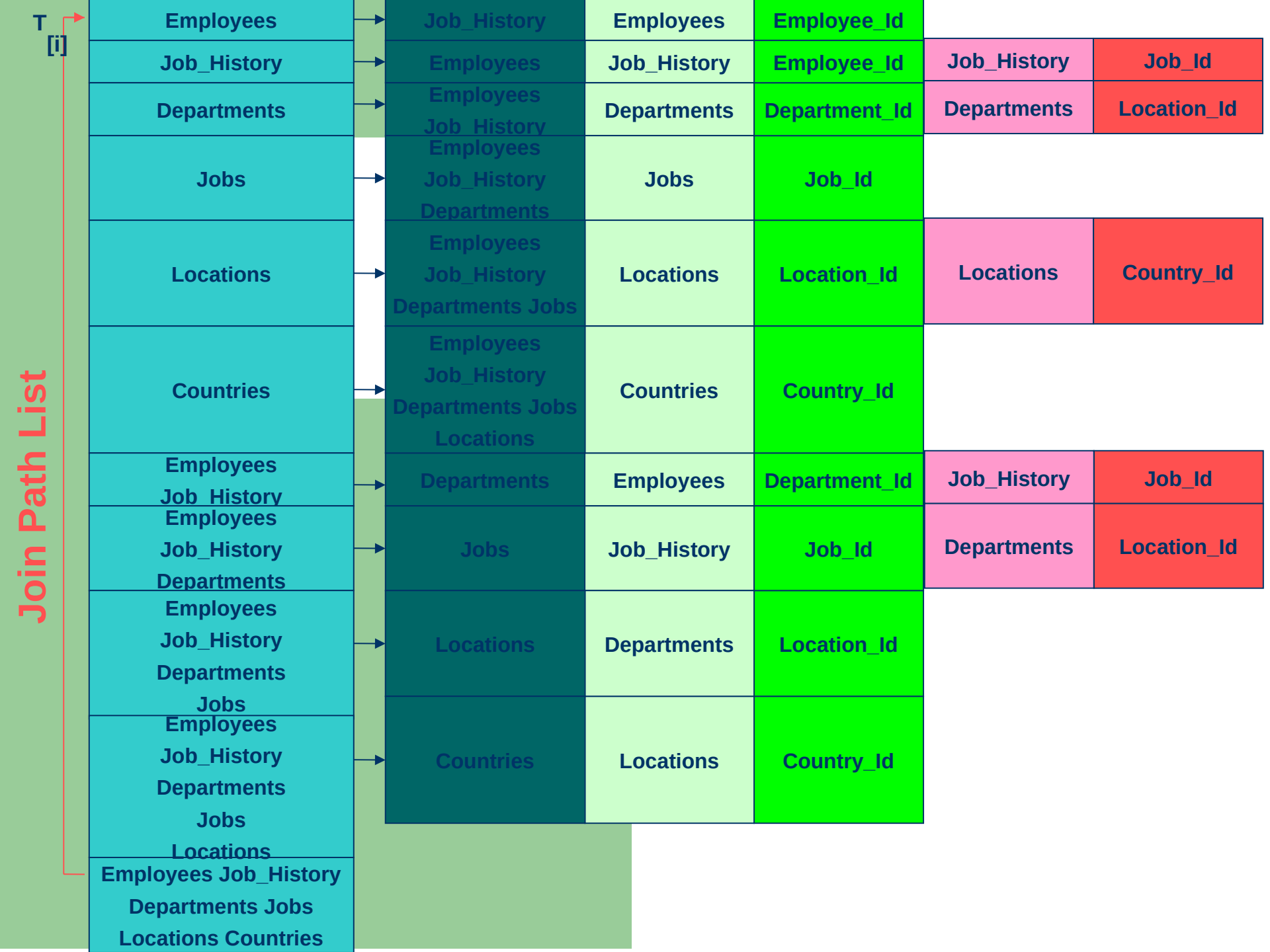
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

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

buf.Table =  $T_i$

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

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

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

buf.Table =  $T_i$

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

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

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

buf.Table =  $T_i$

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

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

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

buf.Table =  $T_i$

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

Table buf 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

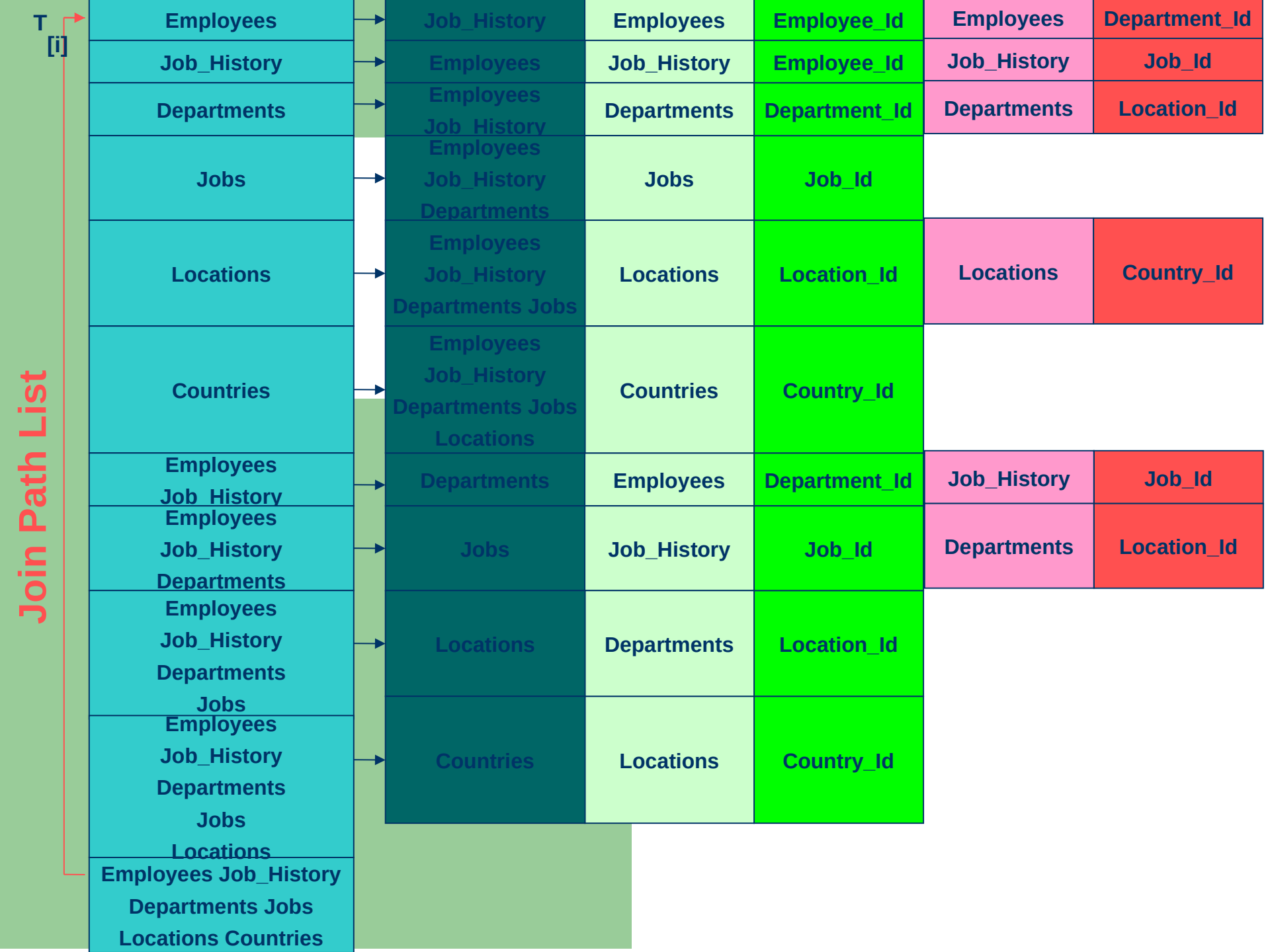
for every buf.Table =  $T_k$  do

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

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

buf.Table =  $T_i$

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

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

buf.Table =  $T_i$

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





Suppose we want the join sequence of the tables ordered by Employees.Name and Departments.Department\_Id, applying the algorithm, the JoinPathList becomes:

Nodes List		Adjacent Table	Table	Key	Inherited Table	Key
Join Path List	0	Employees	Job_History	Employees	Employee_Id	Employees Name 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	Employees Name Job_History Job_Id
Example A	7	Employees Job_History Departments	Jobs	Job_History	Job_Id	Employees Name
						Departments Department_Id
						Departments 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



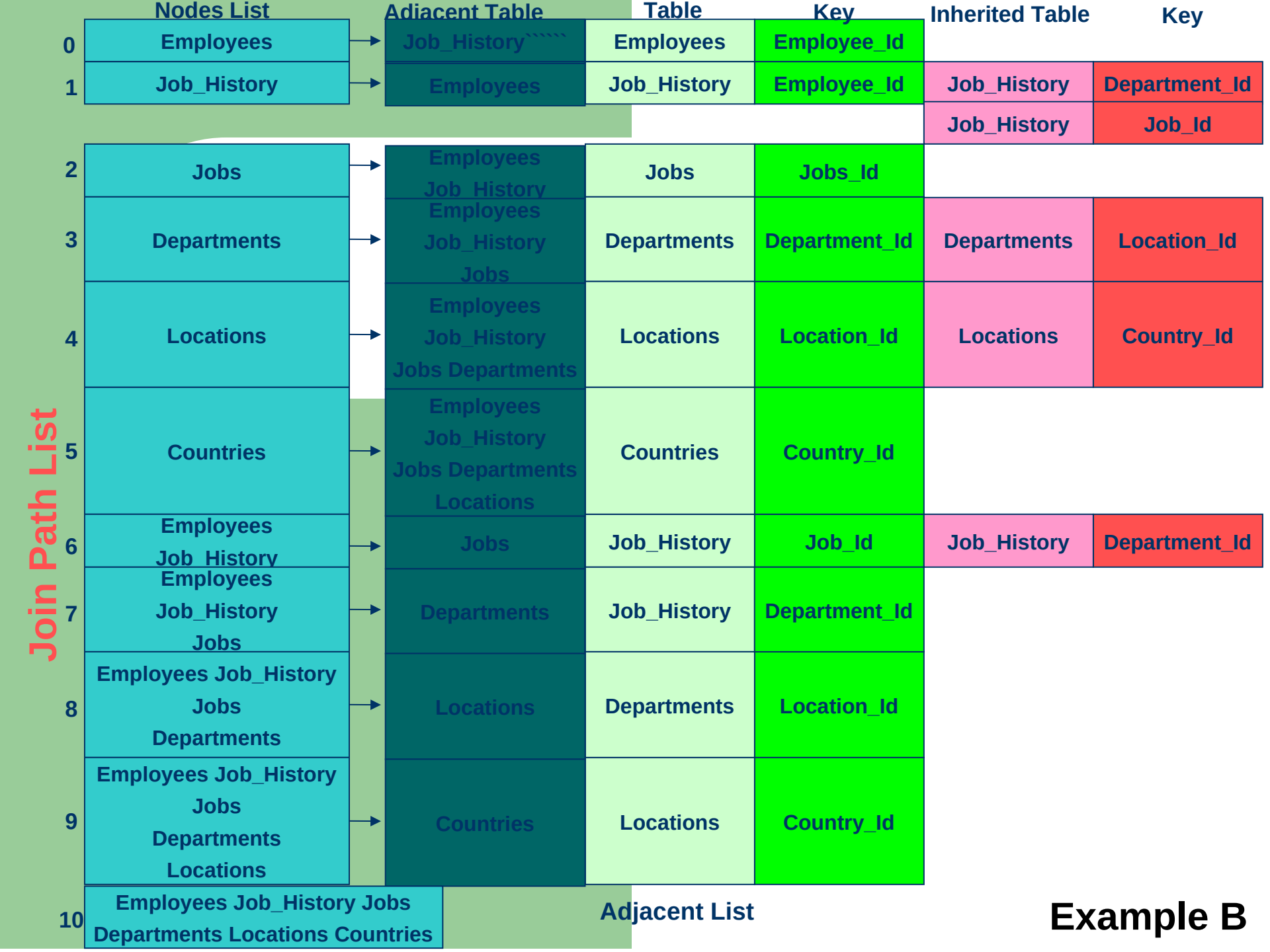
Employees

Name

Departments

Department\_Id

# Example A





# Create B<sup>+</sup>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<sup>+</sup>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<sup>+</sup>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, Inherited Key> in  
        the adjacent list of the node

Give a general name for the  $B^{\infty}$ Tree.

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

About the last virtual table, its index has no keys, it works because we consider pairs of  $\langle \text{keys}, \text{Data Pointers} \rangle$  as key, 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.

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.

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.

The  $B^{\infty}$ Tree is formed from  $(2*n-1)$  indexes.

We can use one index that includes all these indexes by including an index number and treated like a key, so instead of  $\langle \text{keys}, \text{Data Pointers} \rangle$  treated as a Key, we can use  $\langle \text{Index PathJoinList entry number}, \text{Keys}, \text{Data Pointers} \rangle$  as a key.

# **IMPORTANT NOTE**

**The data pointers to the tables in join are in order with the Base Tables of the last Virtual Table and not as declared in the create constructor, this is due to the fact that Path is not always in the same order as the tables declared in the constructor.**

**So to know the order of data pointers in respect to the tables in join, the property “BaseTables” in “BjoinTreeU.pas” should be called to get it.**

**The function GetDataRefByTableName(BaseTable: string; DataRef: array of DataPointerType): DataPointerType; give the data reference to the row in base table. See Test Index (Button5) in SQLProject.**

# Comparison

In the same fashion when using an ordinary B<sup>+</sup>Tree and one row get inserted, the system check the definition of the B<sup>+</sup>Tree and get the necessary keys from the row to insert them, with B<sup>×</sup>Tree the system check the JoinPathList to get the keys and the inherited keys.

# 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 ( $T_{[i]}$ , $Keys_{[i]}$ , InheritedKeys $_{[i]}$ , [DP $_i$ ])

- Call AddKey (B<sup>+</sup>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<sup>+</sup>Tree( $T_{[k]}$ ),  $Keys_{[i]}$ )
  - While found( $keys_{[i]}$ ) do
    - ReturnKeys(B<sup>+</sup>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<sup>+</sup>Tree( $T_{[k]}$ ),  $Keys_{[i]}$ )

# Employees table

DP start from 0

	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

DP start from 0

	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

DP start from 0

	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

DP start from 0

	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

DP start from 0

	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

DP start from 0

	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<sub>i</sub>, Row R<sub>m</sub>, DataRef DP<sub>i</sub>)

locate the entry of T<sub>i</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from row R<sub>m</sub> get the columns constituting the keys & Inherited Keys

call AddJoinKey(T<sub>i</sub>,Keys<sub>i</sub>,InheritedKeys<sub>i</sub>,DP<sub>i</sub>)

Base Table

Employees
0

T<sub>i</sub>

DP<sub>i</sub>  
0



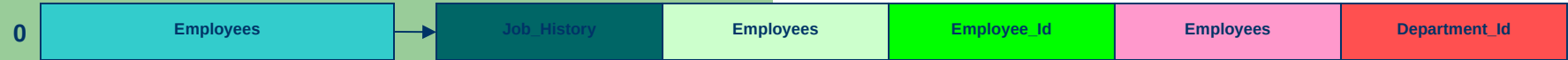
DataRef

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

Row R<sub>m</sub>

insertRoutine(in BaseTable T<sub>i</sub>, Row R<sub>m</sub>, DataRef DP<sub>i</sub>)

locate the entry of T in JoinPathList  
from its adjacentList, locate the definition of Keys & Inherited Keys  
from row R<sub>m</sub> get the columns constituting the keys & Inherited Keys  
call AddJoinKey(T<sub>i</sub>,Keys<sub>i</sub>,InheritedKeys<sub>i</sub>,DP<sub>i</sub>)



insertRoutine(in BaseTable T<sub>i</sub>, Row R<sub>m</sub>, DataRef DP<sub>i</sub>)

locate the entry of T in JoinPathList  
from its adjacentList, locate the definition of Keys & Inherited Keys  
from row R<sub>m</sub> get the columns constituting the keys & Inherited Keys  
call AddJoinKey(T<sub>i</sub>,Keys<sub>i</sub>,InheritedKeys<sub>i</sub>,DP<sub>i</sub>)



insertRoutine(in BaseTable T<sub>i</sub>, Row R<sub>m</sub>, DataRef DP<sub>i</sub>)

locate the entry of T in JoinPathList  
from its adjacentList, locate the definition of Keys & Inherited Keys  
from row R<sub>m</sub> get the columns constituting the keys & Inherited Keys  
call AddJoinKey(T<sub>i</sub>,Keys<sub>i</sub>,InheritedKeys<sub>i</sub>,DP<sub>i</sub>)

DP<sub>i</sub> 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

Keys

Row R<sub>m</sub>

Inherited Keys



```
insertRoutine(in BaseTable Ti, Row Rm, DataRef DPi)
```

locate the entry of T<sub>i</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from row R<sub>m</sub> get the columns constituting the keys & Inherited Keys

```
call AddJoinKey(Ti, Keysi, InheritedKeysi, DPi)
```



EMPLO
YEE_ID
101

Keys

Employees
0

T<sub>i</sub>

0
---

DP<sub>i</sub>

DEPART
MENT_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  $_{[i]}$ )

EMPLO YEE_ID
101

Keys  
 $_{[i]}$

Employees
0

$T_{[i]}$

0
---

[DP] $_i$

DEPART MENT_ID
FIN

Inherited Keys  
 $_{[i]}$

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

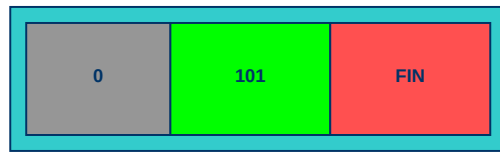
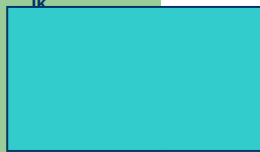
from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

B+Tree(T<sub>0</sub>)



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

0



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

0	Employees	Job_History	Employees	Employee_Id	Employees	Department_Id
---	-----------	-------------	-----------	-------------	-----------	---------------

### Adjacent Table

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

1	Job_History	Employees	Job_History	Employee_Id	Job_History	Job_Id
---	-------------	-----------	-------------	-------------	-------------	--------

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

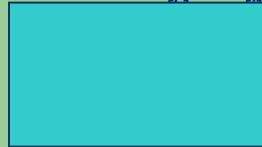
from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

B+Tree(T<sub>1</sub>)



Found: FALSE

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

Found: FALSE

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

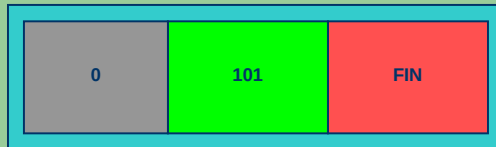
from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

B+Tree(T<sub>0</sub>)



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

$T_i$

	EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
$DP_i$ 0 ↑ DataRef	101	12/16/1998	12/15/1999	AC_AUD	ACC
	Row $R_m$				



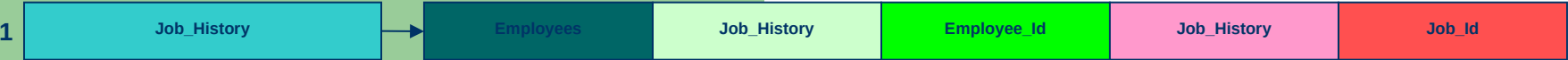
insertRoutine(in BaseTable T<sub>i</sub>, Row R<sub>m</sub>, DataRef DP<sub>i</sub>)

locate the entry of T<sub>i</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from row R<sub>m</sub> get the columns constituting the keys & Inherited Keys

call AddJoinKey(T<sub>i</sub>,Keys<sub>i</sub>,InheritedKeys<sub>i</sub>,DP<sub>i</sub>)



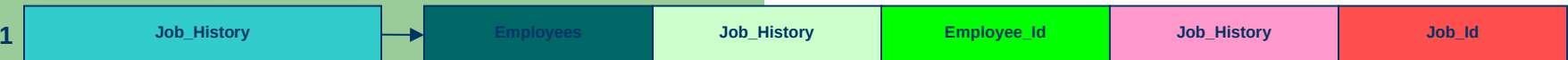
insertRoutine(in BaseTable T<sub>i</sub>, Row R<sub>m</sub>, DataRef DP<sub>i</sub>)

locate the entry of T<sub>i</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from row R<sub>m</sub> get the columns constituting the keys & Inherited Keys

call AddJoinKey(T<sub>i</sub>,Keys<sub>i</sub>,InheritedKeys<sub>i</sub>,DP<sub>i</sub>)



Keys

Inherited Keys

insertRoutine(in BaseTable T<sub>i</sub>, Row R<sub>m</sub>, DataRef DP<sub>i</sub>)

locate the entry of T<sub>i</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from row R<sub>m</sub> get the columns constituting the keys & Inherited Keys

call AddJoinKey(T<sub>i</sub>,Keys<sub>i</sub>,InheritedKeys<sub>i</sub>,DP<sub>i</sub>)

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

Keys

Row R<sub>m</sub>

Inherited Keys

```
insertRoutine(in BaseTable Ti, Row Rm, DataRef DPi)
```

locate the entry of T<sub>i</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from row R<sub>m</sub> get the columns constituting the keys & Inherited Keys

call AddJoinKey(T<sub>i</sub>, Keys<sub>i</sub>, InheritedKeys<sub>i</sub>, DP<sub>i</sub>)



EMPLOYEE_ID
101

Keys

Job_History
1

T<sub>i</sub>

0
---

DP<sub>i</sub>

JOB_ID
AC_AUD

Inherited Keys



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

EMPLOYEE_ID
101

Keys<sub>[i]</sub>

Job_History
1

T<sub>[i]</sub>

0
---

[DP]<sub>i</sub>

JOB_ID
AC_AUD

Inherited Keys<sub>[i]</sub>

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

B+Tree(T<sub>1</sub>)

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

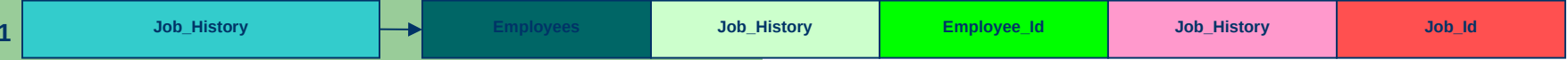
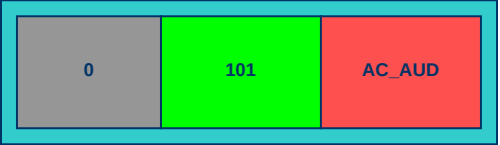
locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

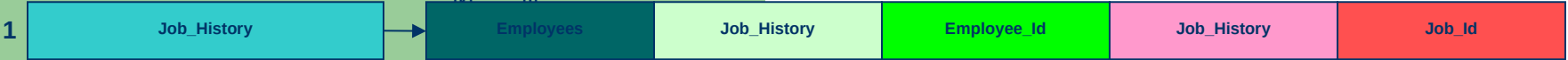
from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

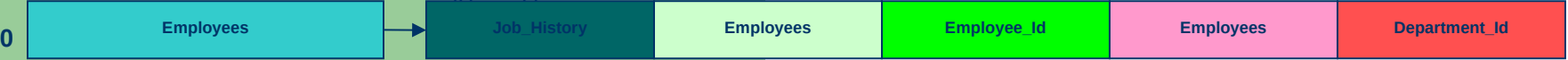


```
addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DPi])
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DPi])
    locate the entry of T[i] in JoinPathList
    from its adjacentList, locate the table T[k] adjacent to it
        locate the entry of T[k] in JoinPathList
        Found = findKey(B+Tree(T[k]), Keys[i])
        while found do
            returnKeys(B+Tree(T[k]), Keys[k], InheritedKeys[k], [DPk])
            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], [DPik])
            Found = nextKey(B+Tree(T[k]), Keys[i])
```



Adjacent Table

```
addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DPi])
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DPi])
    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], [DPk])
            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], [DPik])
            Found = nextKey(B+Tree(T[k]), Keys[i])
```



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

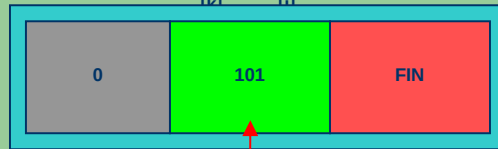
from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[ik]</sub>), Keys<sub>[i]</sub>)

B+Tree(T<sub>0</sub>)



Found: TRUE

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[ik]</sub>), Keys<sub>[i]</sub>)

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])
```

EMPLO YEE_ID
101

Keys<sub>[k]</sub>

0
---

[DP]<sub>k</sub>

DEPART MENT_ID
FIN

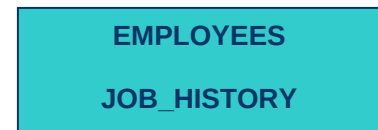
Inherited

Keys<sub>[i]</sub>

```

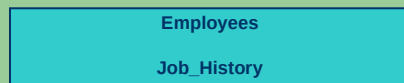
addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DPi])
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DPi])
    locate the entry of T[i] in JoinPathList
    from its adjacentList, locate the table T[k] adjacent to it
        locate the entry of T[k] in JoinPathList
        Found = findKey(B+Tree(T[k]), Keys[i])
        while found do
            returnKeys(B+Tree(T[k]), Keys[k], InheritedKeys[k], [DPk])
            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], [DPik])
            Found = nextKey(B+Tree(T[k]), Keys[i])

```



$T_{ik}$

6



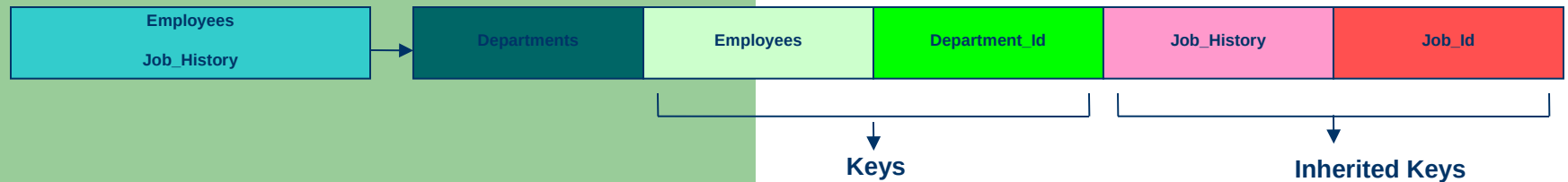


```

addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DPi])
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DPi])
    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], [DPk])
            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], [DPik])
            Found = nextKey(B+Tree(T[k]), Keys[i])

```

6



```

addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DPi])
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DPi])
    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

```

```

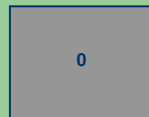
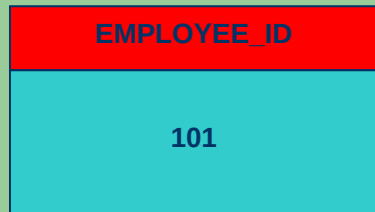
            returnKeys(B+Tree(T[k]), Keys[k], InheritedKeys[k], [DPk])
            locate the entry of T[ik] in JoinPathList
            from its adjacentList, locate the definition of Keys & Inherited Keys
            from keys[i], InheritedKeys[i], Keys[k] & InheritedKeys[k] get the keys & Inherited keys of T[ik]
            call AddJoinKey(T[ik], Keys[ik], InheritedKeys[ik], [DPik])
            Found = nextKey(B+Tree(T[k]), Keys[i])

```

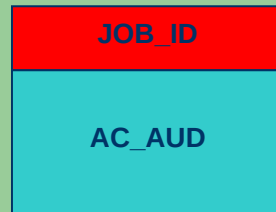


Keys

inherited Keys

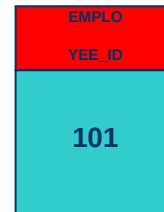


[DP<sub>i</sub>]  
i

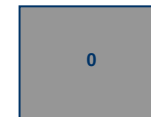


Inherited

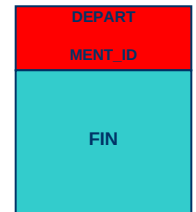
Keys<sub>[i]</sub>



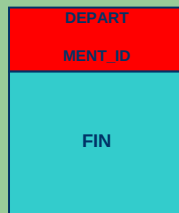
Keys<sub>[k]</sub>



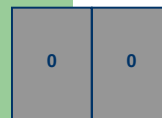
[DP<sub>k</sub>]  
k



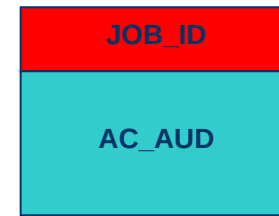
Inherited Keys<sub>[k]</sub>



Keys<sub>[ik]</sub>

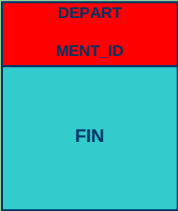


[DP<sub>ik</sub>]  
ik

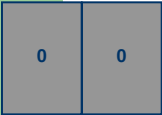


Inherited Keys<sub>[ik]</sub>

```
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<sub>[ik]</sub>



[DP]<sub>ik</sub>



Inherited Keys<sub>[ik]</sub>



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  $_{[i]}$ )

DEPART
MENT_ID
FIN

Keys  $_{[i]}$

Employees
Job_History

$T_{[i]}$

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

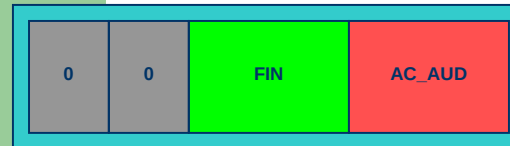
from its adjacentList, locate the definition of Keys & Inherited Keys

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  $_{[i]}$ )

B+Tree( $T_6$ )



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>k</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>k</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)



Keys

Inherited Keys

Adjacent Table

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

B+Tree(T<sub>2</sub>)



Found: FALSE

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

Found: FALSE

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

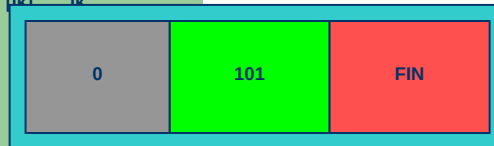
from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

B+Tree(T<sub>0</sub>)



Found: FALSE

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

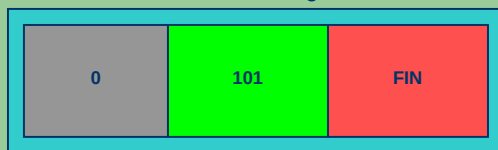
Found: FALSE

```

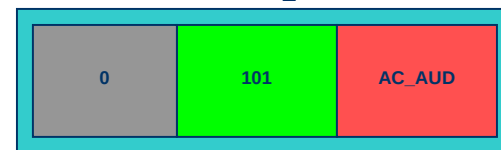
addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DPi])
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DPi])
    locate the entry of T[i] in JoinPathList
    from its adjacentList, locate the table T[k] adjacent to it
        locate the entry of T[k] in JoinPathList
        Found = findKey(B+Tree(T[k]), Keys[i])
        while found do
            returnKeys(B+Tree(T[k]), Keys[k], InheritedKeys[k], [DPk])
            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], [DPik])
            Found = nextKey(B+Tree(T[k]), Keys[i])

```

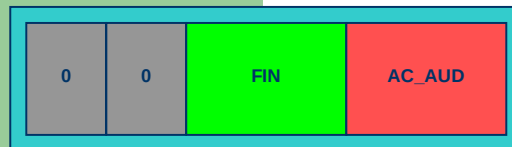
B+Tree(T<sub>0</sub>)



B+Tree(T<sub>1</sub>)



B+Tree(T<sub>6</sub>)



## 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

$DP_i$   
0

↑  
DataRef

Row  $R_m$

insertRoutine(in BaseTable T<sub>i</sub>, Row R<sub>m</sub>, DataRef DP<sub>i</sub>)

locate the entry of T<sub>i</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from row R<sub>m</sub> get the columns constituting the keys & Inherited Keys

call AddJoinKey(T<sub>i</sub>,Keys<sub>i</sub>,InheritedKeys<sub>i</sub>,DP<sub>i</sub>)

4

Locations
-----------

Employees
Job_History
Departments
Jobs

Locations
-----------

Location_Id
-------------

Locations
-----------

Country_Id
------------

insertRoutine(in BaseTable T<sub>i</sub>, Row R<sub>m</sub>, DataRef DP<sub>i</sub>)

locate the entry of T<sub>i</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from row R<sub>m</sub> get the columns constituting the keys & Inherited Keys

call AddJoinKey(T<sub>i</sub>,Keys<sub>i</sub>,InheritedKeys<sub>i</sub>,DP<sub>i</sub>)

4

Locations
-----------

Employees
Job_History
Departments
Jobs

Locations
-----------

Location_Id
-------------

Locations
-----------

Country_Id
------------

insertRoutine(in BaseTable T<sub>i</sub>, Row R<sub>m</sub>, DataRef DP<sub>i</sub>)

locate the entry of T<sub>i</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from row R<sub>m</sub> get the columns constituting the keys & Inherited Keys

call AddJoinKey(T<sub>i</sub>,Keys<sub>i</sub>,InheritedKeys<sub>i</sub>,DP<sub>i</sub>)

DP<sub>i0</sub>

Keys

Inherited Keys

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

Keys

Row R<sub>m</sub>

Keys

```
insertRoutine(in BaseTable Ti, Row Rm, DataRef DPi)
```

locate the entry of T<sub>i</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from row R<sub>m</sub> get the columns constituting the keys & Inherited Keys

```
call AddJoinKey(Ti, Keysi, InheritedKeysi, DPi)
```

LOCATION_ ID
1000

Keys

Locations
4

T<sub>i</sub>

0
---

DP<sub>i</sub>

COUNTRY_ ID
ca

Inherited Keys



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

LOCATION_ ID
1000

Keys<sub>[i]</sub>

Locations
4

T<sub>[i]</sub>

0
---

[DP]<sub>i</sub>

COUNTRY_ ID
ca

Inherited Keys<sub>[i]</sub>

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

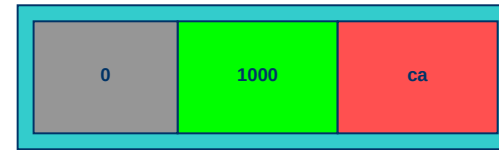
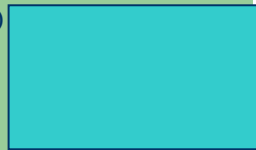
from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

**B+Tree(T<sub>[i]</sub>)**



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

4





addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

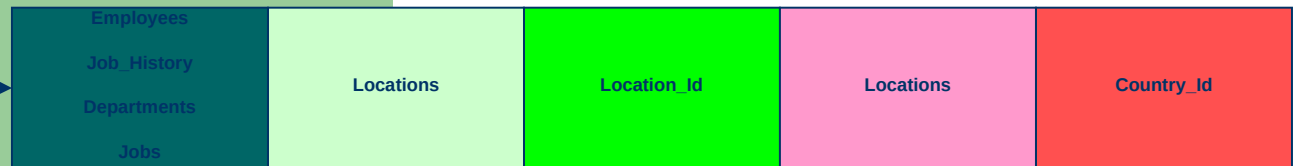
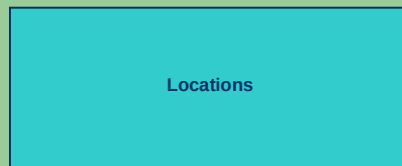
from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

4



Adjacent Table

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

8



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

B+Tree(T<sub>8</sub>)

Found: FALSE

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

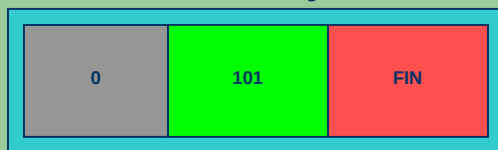
Found: FALSE

```

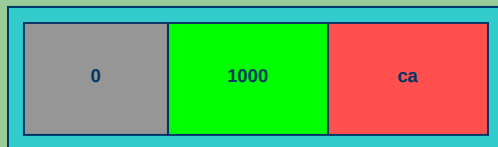
addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DPi])
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DPi])
    locate the entry of T[i] in JoinPathList
    from its adjacentList, locate the table T[k] adjacent to it
        locate the entry of T[k] in JoinPathList
        Found = findKey(B+Tree(T[k]), Keys[i])
        while found do
            returnKeys(B+Tree(T[k]), Keys[k], InheritedKeys[k], [DPk])
            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], [DPik])
            Found = nextKey(B+Tree(T[k]), Keys[i])

```

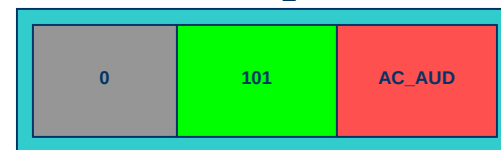
B+Tree(T<sub>0</sub>)



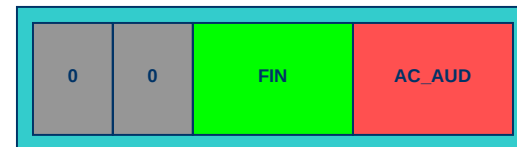
B+Tree(T<sub>4</sub>)



B+Tree(T<sub>1</sub>)



B+Tree(T<sub>6</sub>)



## Inserting first row from table Departments

Base Table

DEPARTMENTS
2

Department_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

$T_i$

$DP_i$ 0	Department_Id	Department_Name	Manager_Id	Location_Id
DataRef ↑	FIN	FINANCE	101	1000
	Row $R_m$			

```
insertRoutine(in BaseTable Ti, Row Rm, DataRef DPi)
```

locate the entry of T<sub>i</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from row R<sub>m</sub> get the columns constituting the keys & Inherited Keys

call AddJoinKey(T<sub>i</sub>,Keys<sub>i</sub>,InheritedKeys<sub>i</sub>,DP<sub>i</sub>)



```
insertRoutine(in BaseTable Ti, Row Rm, DataRef DPi)
```

locate the entry of T<sub>i</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from row R<sub>m</sub> get the columns constituting the keys & Inherited Keys

call AddJoinKey(T<sub>i</sub>,Keys<sub>i</sub>,InheritedKeys<sub>i</sub>,DP<sub>i</sub>)



```
insertRoutine(in BaseTable Ti, Row Rm, DataRef DPi)
```

locate the entry of T<sub>i</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from row R<sub>m</sub> get the columns constituting the keys & Inherited Keys

call AddJoinKey(T<sub>i</sub>,Keys<sub>i</sub>,InheritedKeys<sub>i</sub>,DP<sub>i</sub>)

Keys

Inherited Keys

DP<sub>i</sub>  
0

Department_Id	Department_Name	Manager_Id	Location_Id
FIN	FINANCE	101	1000

Keys

Row R<sub>m</sub>

Inherited Keys

```
insertRoutine(in BaseTable Ti, Row Rm, DataRef DPi)
```

locate the entry of T<sub>i</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from row R<sub>m</sub> get the columns constituting the keys & Inherited Keys

call AddJoinKey(T<sub>i</sub>, Keys<sub>i</sub>, InheritedKeys<sub>i</sub>, DP<sub>i</sub>)



Department_Id
FIN

Keys

DEPARTMENTS
2

T<sub>i</sub>

0
---

DP<sub>i</sub>

Location_Id
1000

Keys





addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

DEPARTMENTS

2

T<sub>[i]</sub>

Deparment\_Id

FIN

Keys<sub>[i]</sub>

0

[DP]<sub>i</sub>

Location\_Id

1000

Inherited Keys<sub>[i]</sub>

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

B+Tree(T<sub>2</sub>)

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

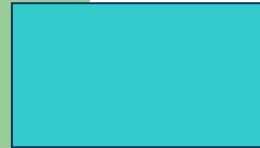
locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)



2



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

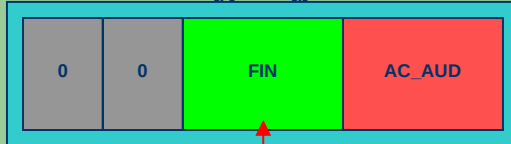
from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

B+Tree(T<sub>6</sub>)



Found: TRUE

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

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])
```

DEPART
MENT_ID
FIN

Keys<sub>[k]</sub>

0	0
---	---

[DP]<sub>k</sub>

JOB_ID
AC_AUD

Inherited Keys<sub>[k]</sub>

```

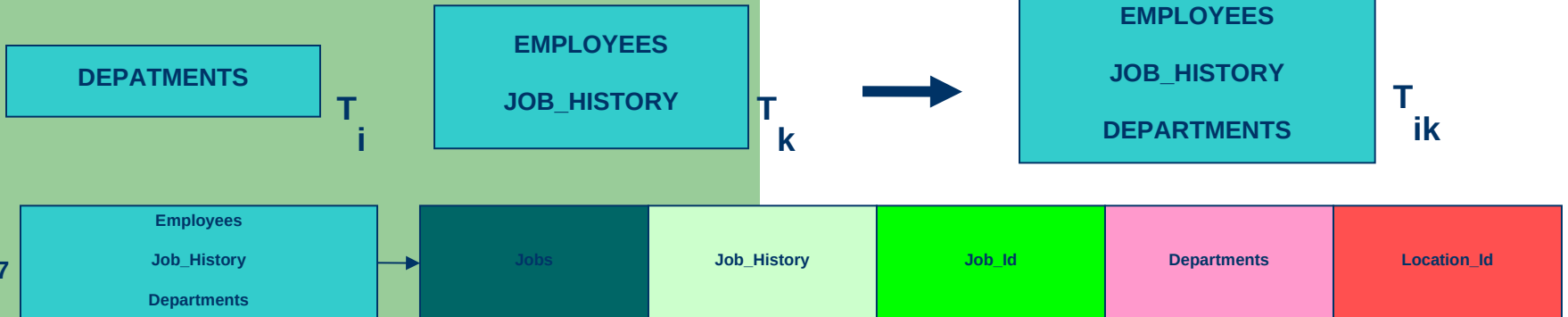
addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DPi])
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DPi])
    locate the entry of T[i] in JoinPathList
    from its adjacentList, locate the table T[k] adjacent to it
        locate the entry of T[k] in JoinPathList
        Found = findKey(B+Tree(T[k]), Keys[i])
        while found do

```

```

            returnKeys(B+Tree(T[k]), Keys[k], InheritedKeys[k], [DPk])
            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], [DPik])
            Found = nextKey(B+Tree(T[k]), Keys[i])

```



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

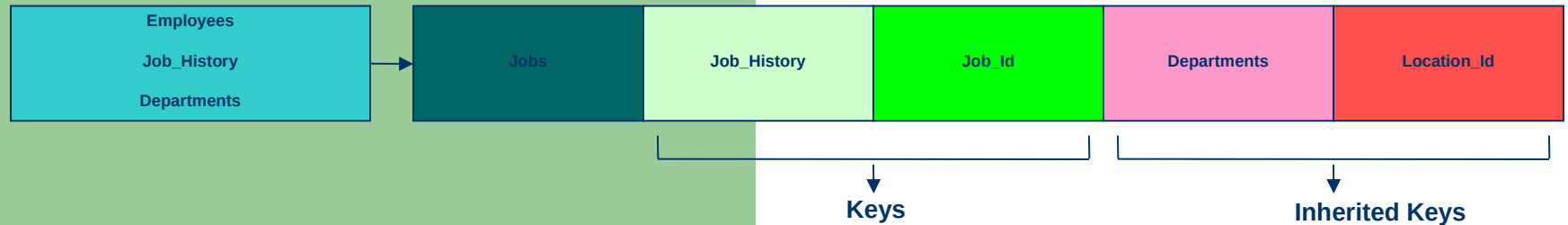
from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

7



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

7

Employees  
Job\_History  
Departments

Jobs

Job\_History

Job\_Id

Departments

Location\_Id

Deparment\_Id

FIN

Keys<sub>[i]</sub>

[DP]<sub>i</sub>

i

Location\_Id

1000

Inherited

Keys<sub>[i]</sub>

JOB\_ID

AC\_AUD

Keys<sub>[ik]</sub>

[DP]<sub>i</sub>

i

Keys

DEPART  
MENT\_ID

FIN

Keys<sub>[k]</sub>

Inherited Keys

JOB\_ID

AC\_AUD

Inherited Keys<sub>[k]</sub>

0

0

[DP]<sub>k</sub>

k

Location\_Id

1000

Inherited Keys<sub>[ik]</sub>

0 0 0

[DP]<sub>ik</sub>

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<sub>ik</sub>

JOB_ID
AC_AUD

Keys<sub>[ik]c</sub>

0	0	0
---	---	---

[DP]<sub>ik</sub>

Location_Id
1000

Inherited Keys<sub>[i]</sub>



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

JOB_ID
AC_AUD

Keys<sub>[i]</sub>

EMPLOYEES
JOB_HISTORY
DEPARTMENTS

T<sub>[i]</sub>

0	0	0
---	---	---

[DP]<sub>i</sub>

Location_Id
1000

Inherited Keys<sub>[i]</sub>

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

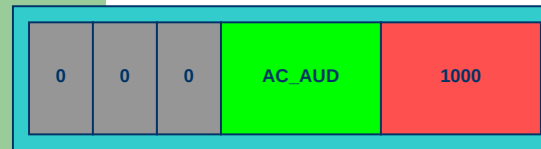
from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

B+Tree(T<sub>7</sub>)



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

7



Keys

Inherited Keys

```

addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DPi])
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DPi])
    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], [DPk])
            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], [DPik])
            Found = nextKey(B+Tree(T[k]), Keys[i])

```

7



**Adjacent Table**

```

addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DPi])
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DPi])
    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], [DPk])

```

```

    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]

```

```

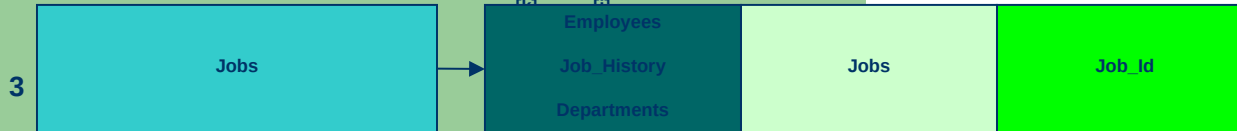
    call addJoinKey(T[ik], Keys[ik], InheritedKeys[ik], [DPik])

```

```

    Found = nextKey(B+Tree(T[k]), Keys[i])

```



```

addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DPi])
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DPi])
    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], [DPk])

```

```

    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], [DPik])

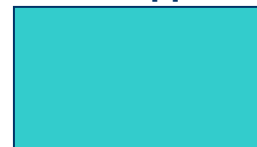
```

```

    Found = nextKey(B+Tree(T[k]), Keys[i])

```

B+Tree(T<sub>3</sub>)



Found: FALSE

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

Found: FALSE

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

```

addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DPi])
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DPi])
    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], [DPk])

```

```

            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], [DPik])

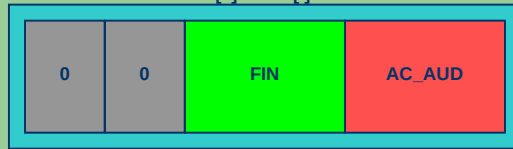
```

```

            Found = nextKey(B+Tree(T[k]), Keys[i])

```

B+Tree(T<sub>6</sub>)



Found: FALSE

```

addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DPi])
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DPi])
    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], [DPk])

```

```

            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], [DPik])

```

```

            Found = nextKey(B+Tree(T[k]), Keys[i])

```

Found: FALSE

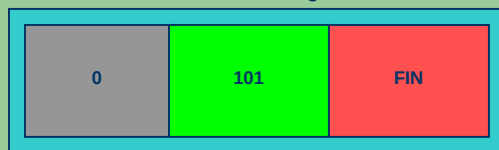


```

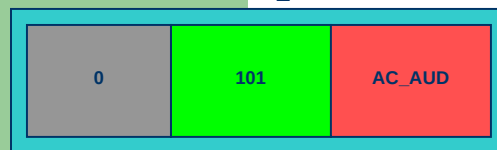
addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DPi])
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DPi])
    locate the entry of T[i] in JoinPathList
    from its adjacentList, locate the table T[k] adjacent to it
        locate the entry of T[k] in JoinPathList
        Found = findKey(B+Tree(T[k]), Keys[i])
        while found do
            returnKeys(B+Tree(T[k]), Keys[k], InheritedKeys[k], [DPk])
            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], [DPik])
            Found = nextKey(B+Tree(T[k]), Keys[i])

```

B+Tree(T<sub>0</sub>)



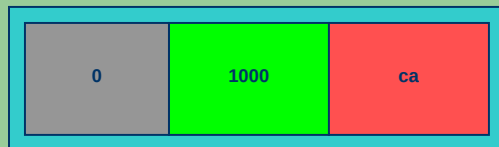
B+Tree(T<sub>1</sub>)



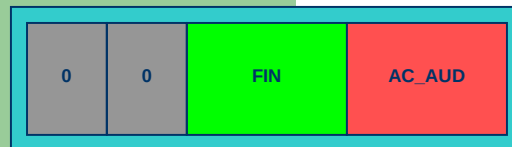
B+Tree(T<sub>2</sub>)



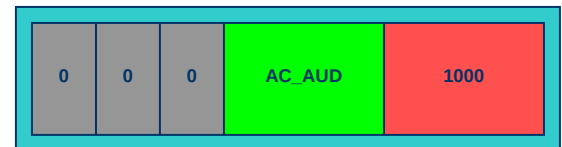
B+Tree(T<sub>4</sub>)



B+Tree(T<sub>6</sub>)



B+Tree(T<sub>7</sub>)



## Inserting first row from table Jobs

Base Table

JOB_ID
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
$DP_i$ 0 ↑ DataRef	AC_AUD	Accounting Auditor	30000	60000
	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 $_i$ ,  $DP_i$ )

3

Jobs
------



Employees
Job_History
Departments

Jobs
------

Job_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 $_i$ ,  $DP_i$ )

3

Jobs
------



Employees
Job_History
Departments

Jobs
------

Job_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<sub>i0</sub>

JOB_ID
--------

JOB_TITLE
-----------

MIN_SALARY
------------

MAX_SALARY
------------

AC_AUD
--------

Accounting Auditor
--------------------

30000
-------

60000
-------

Keys

Row  $R_m$

Keys

Inherited Keys

```
insertRoutine(in BaseTable Ti, Row Rm, DataRef DPi)
```

locate the entry of T<sub>i</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from row R<sub>m</sub> get the columns constituting the keys & Inherited Keys

```
call AddJoinKey(Ti, Keysi, InheritedKeysi, DPi)
```



JOB_ID
AC_AUD

Keys

JOBS
3

T<sub>i</sub>

0
---

DP<sub>i</sub>



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

JOB_ID
AC_AUD

Keys<sub>[i]</sub>

JOBS
3

T<sub>[i]</sub>

0
---

[DP]<sub>i</sub>

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

B+Tree(T<sub>3</sub>)

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

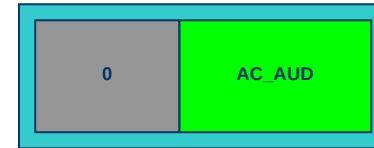
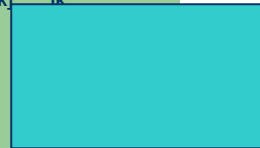
locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

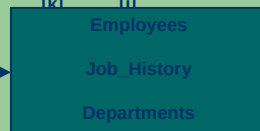
from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)



3



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

3

Jobs
------



Employees		
Job_History		
Departments		

Adjacent Table

7

Employees
Job_History
Departments



Jobs	Job_History	Job_Id	Departments	Location_Id
------	-------------	--------	-------------	-------------



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

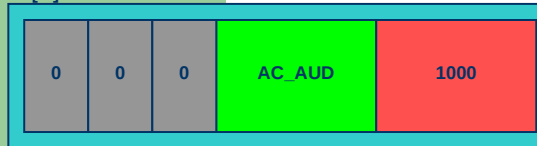
from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

B+Tree(T<sub>7</sub>)



Found: TRUE

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

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<sub>[k]</sub>

0	0	0
---	---	---

[DP]<sub>k</sub>

Location_Id
1000

Inherited Keys<sub>[i]</sub>

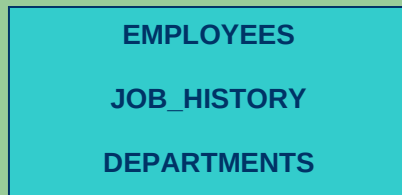
```

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])

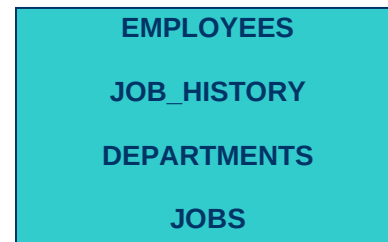
```



T<sub>i</sub>

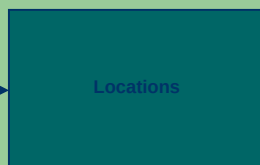


T<sub>k</sub>



T<sub>ik</sub>

8

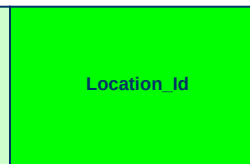
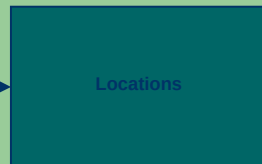


```

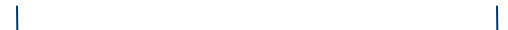
addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DPi])
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DPi])
    locate the entry of T[i] in JoinPathList
    from its adjacentList, locate the table T[k] adjacent to it
        locate the entry of T[k] in JoinPathList
        Found = findKey(B+Tree(T[k]), Keys[i])
        while found do
            returnKeys(B+Tree(T[k]), Keys[k], InheritedKeys[k], [DPk])
            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], [DPik])
            Found = nextKey(B+Tree(T[k]), Keys[i])

```

8



↓  
Keys



↓  
Inherited Keys

```

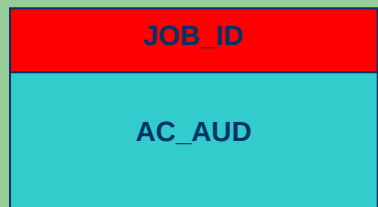
addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DP]i)
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DP]i)
    locate the entry of T[i] in JoinPathList
    from its adjacentList, locate the table T[k] adjacent to it
        locate the entry of T[k] in JoinPathList
        Found = findKey(B+Tree(T[k]), Keys[i])
        while found do

```

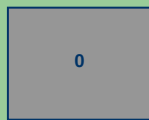
```

            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)
            Found = nextKey(B+Tree(T[k]), Keys[i])

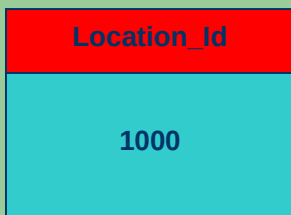
```



Keys<sub>[i]</sub>

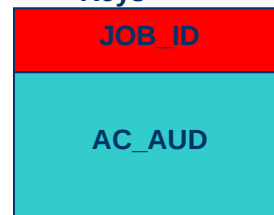


[DP]<sub>i</sub>

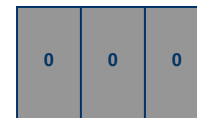


Keys<sub>[ik]</sub>

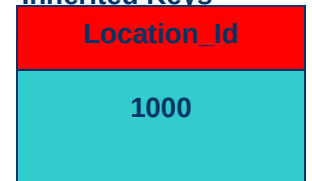
Inherited Keys<sub>[i]</sub>



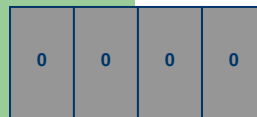
Keys<sub>[k]</sub>



[DP]<sub>k</sub>



Inherited Keys<sub>[k]</sub>



[DP]<sub>ik</sub>

Inheited Keys<sub>[ik]</sub>

```
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<sub>[ik]</sub>

EMPLOYEES			
JOB_HISTORY			
DEPARTMENTS			
JOBS			

T<sub>ik</sub>

0	0	0	0
---	---	---	---

[DP]<sub>ik</sub>

Inherited Keys<sub>[ik]</sub>

→ addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

Location_Id
1000

Keys<sub>[i]</sub>

EMPLOYEES
JOB_HISTORY
DEPARTMENTS
JOBS

T<sub>[i]</sub>

0	0	0	0
---	---	---	---

[DP]<sub>i</sub>

Inherited Keys<sub>[i]</sub>

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

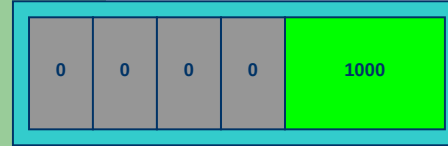
from its adjacentList, locate the definition of Keys & Inherited Keys

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  $_{ik}$ ])

call addKey(B+Tree( $T_{[i]}$ ), Keys  $_{[i]}$ , InheritedKeys  $_{[i]}$ , [DP  $_{ik}$ ])

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  $_{ik}$ ])

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

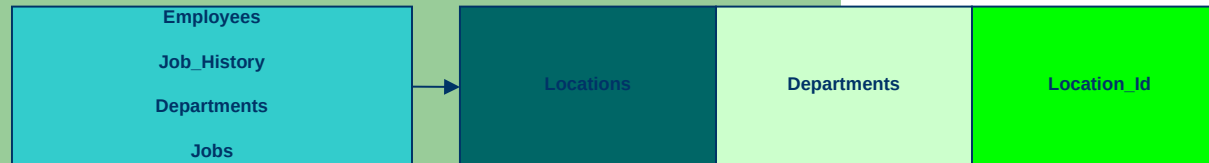
from its adjacentList, locate the definition of Keys & Inherited Keys

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  $_{[i]}$ )

8



```

addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DPi])
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DPi])
    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], [DPk])
            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], [DPik])
            Found = nextKey(B+Tree(T[k]), Keys[i])

```



Adjacent  
Table

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

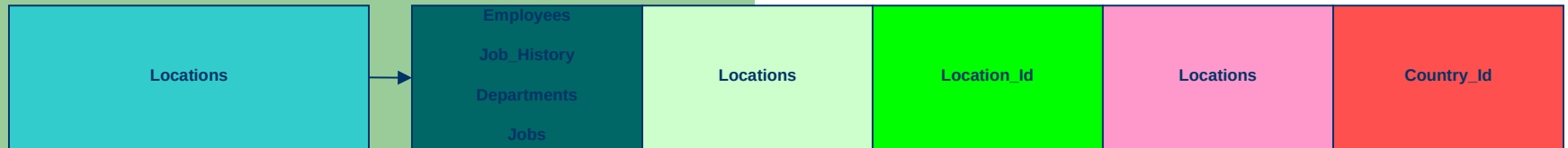
from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

4



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

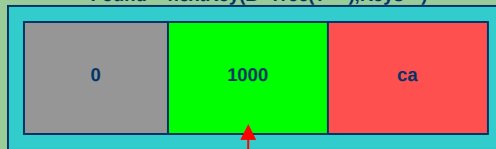
from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

B+Tree(T<sub>4</sub>)



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

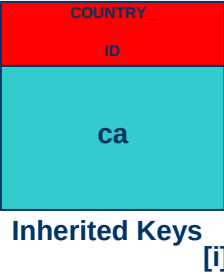
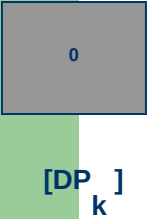
Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

Found: TRUE

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])
```



```

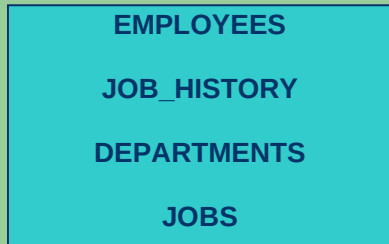
addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DPi])
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DPi])
    locate the entry of T[i] in JoinPathList
    from its adjacentList, locate the table T[k] adjacent to it
        locate the entry of T[k] in JoinPathList
        Found = findKey(B+Tree(T[k]), Keys[i])
        while found do

```

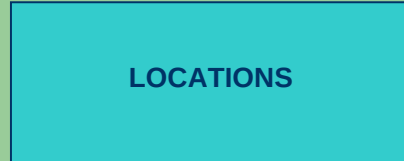
```

    returnKeys(B+Tree(T[k]), Keys[k], InheritedKeys[k], [DPk])
    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], [DPik])
    Found = nextKey(B+Tree(T[k]), Keys[i])

```



$T_i$

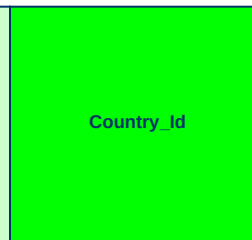
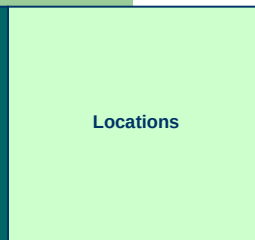


$T_k$



$T_{ik}$

9



```

addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DPi])
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DPi])
    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], [DPk])

```

```

            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], [DPik])

```

```

            Found = nextKey(B+Tree(T[k]), Keys[i])

```

9



Keys

Inherited Keys

```

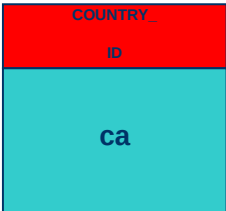
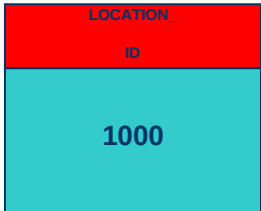
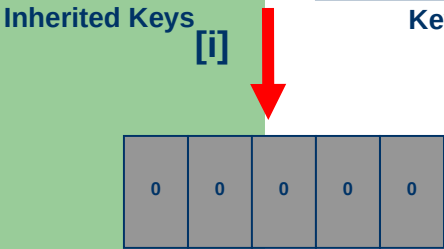
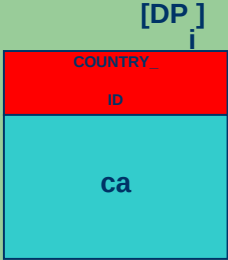
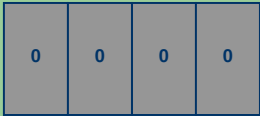
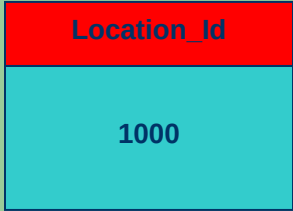
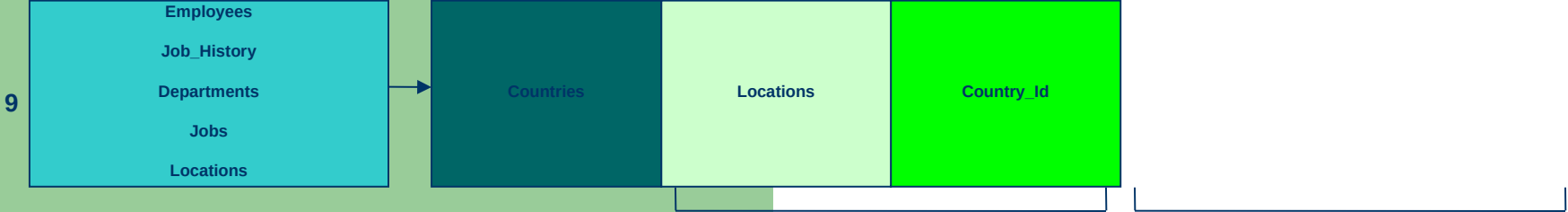
addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DP]i)
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DP]i)
    locate the entry of T[i] in JoinPathList
    from its adjacentList, locate the table T[k] adjacent to it
        locate the entry of T[k] in JoinPathList
        Found = 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)
            Found = nextKey(B+Tree(T[k]), Keys[i])

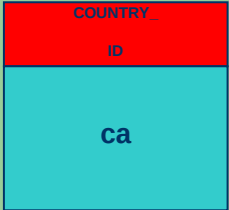
```



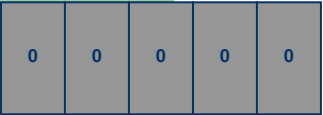


```
addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DP]i)
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DP]i)
    locate the entry of T[i] in JoinPathList
    from its adjacentList, locate the table T[k] adjacent to it
        locate the entry of T[k] in JoinPathList
        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])
```



Keys<sub>[ik]</sub>



[DP]<sub>ik</sub>



T<sub>ik</sub>

Inherited Keys<sub>[ik]</sub>



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  $_{[k]}$ ,  $[DP]_k$ )

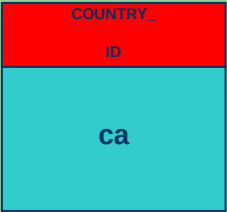
locate the entry of  $T_{[ik]}$  in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

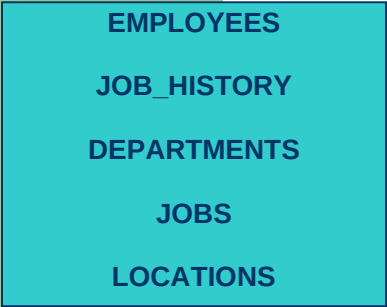
from keys  $_{[i]}$ , InheritedKeys  $_{[i]}$ , Keys  $_{[k]}$  & InheritedKeys  $_{[k]}$  get the keys & Inherited keys of  $T_{[ik]}$

call addJoinKey( $T_{[ik]}$ , Keys  $_{[ik]}$ , InheritedKeys  $_{[ik]}$ ,  $[DP]_{ik}$ )

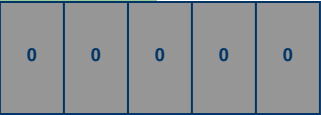
Found = nextKey(B+Tree( $T_{[k]}$ ), Keys  $_{[i]}$ )



Keys  $_{[i]}$



$T_{[i]}$



$[DP]_i$

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

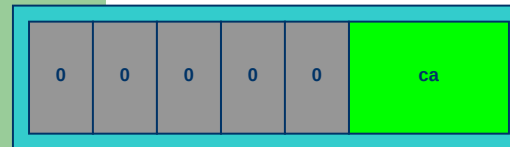
from its adjacentList, locate the definition of Keys & Inherited Keys

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  $_{[i]}$ )

B+Tree( $T_9$ )



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

9

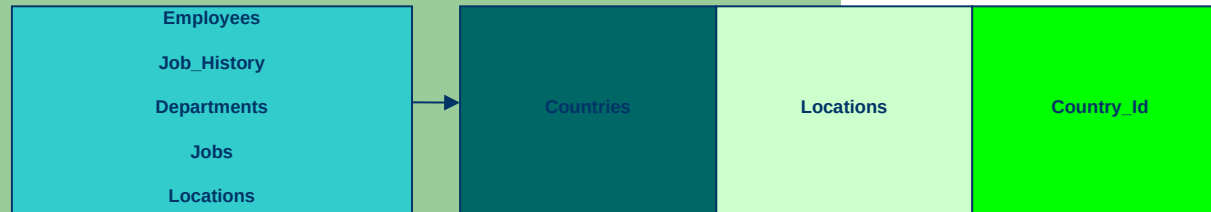


```

addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DPi])
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DPi])
    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], [DPk])
            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], [DPik])
            Found = nextKey(B+Tree(T[k]), Keys[i])

```

9



Adjacent Table

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

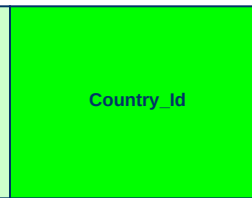
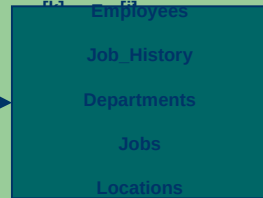
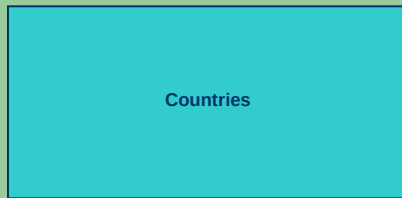
from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

5



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

B+Tree(T<sub>5</sub>)



Found: FALSE

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

Found: FALSE

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

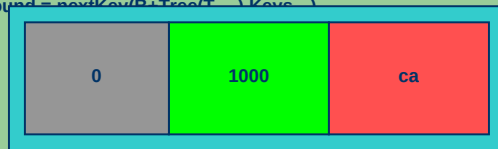
from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

B+Tree(T<sub>4</sub>)



Found: FALSE

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

Found: FALSE



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

B+Tree(T<sub>7</sub>)

0	0	0	AC_AUD	1000
---	---	---	--------	------

Found: FALSE

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

Found: FALSE

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

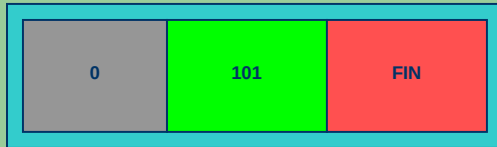
from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

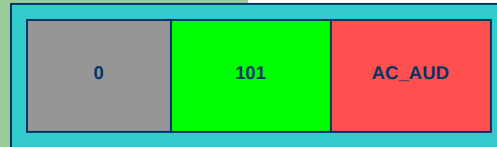
call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

B+Tree(T<sub>0</sub>)



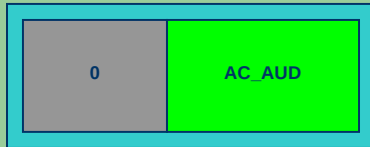
B+Tree(T<sub>1</sub>)



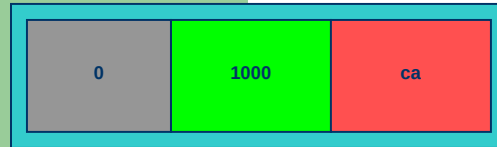
B+Tree(T<sub>2</sub>)



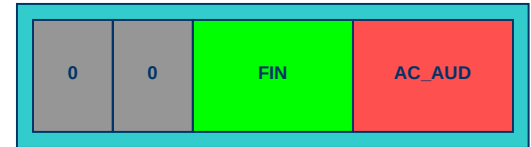
B+Tree(T<sub>3</sub>)



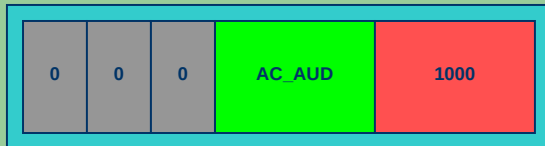
B+Tree(T<sub>4</sub>)



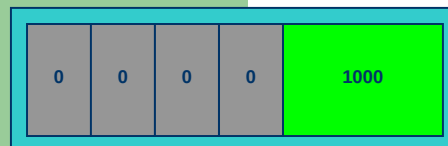
B+Tree(T<sub>6</sub>)



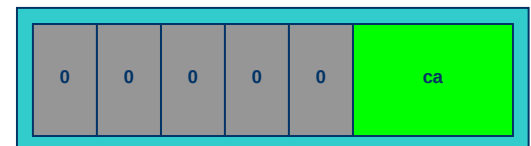
B+Tree(T<sub>7</sub>)



B+Tree(T<sub>8</sub>)



B+Tree(T<sub>9</sub>)



## 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  $T$  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

$DP_i$   
0  
↑  
DataRef

Row  $R_m$

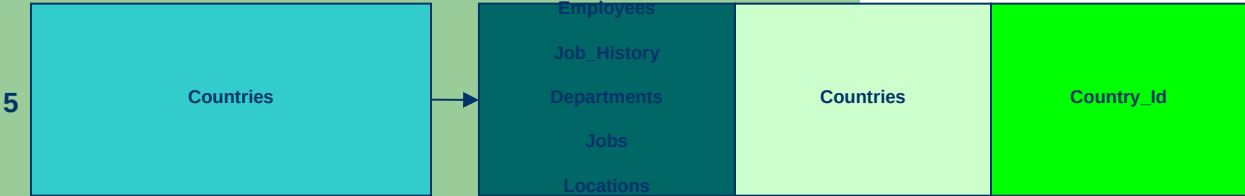
insertRoutine(in BaseTable T<sub>i</sub>, Row R<sub>m</sub>, DataRef DP<sub>i</sub>)

locate the entry of T<sub>i</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from row R<sub>m</sub> get the columns constituting the keys & Inherited Keys

call AddJoinKey(T<sub>i</sub>,Keys<sub>i</sub>,InheritedKeys<sub>i</sub>,DP<sub>i</sub>)



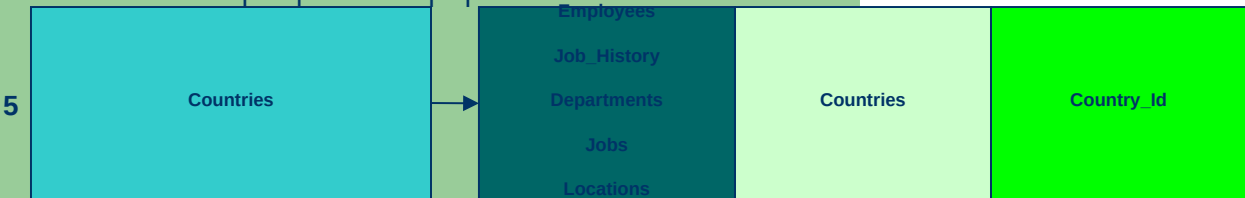
insertRoutine(in BaseTable T<sub>i</sub>, Row R<sub>m</sub>, DataRef DP<sub>i</sub>)

locate the entry of T<sub>i</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from row R<sub>m</sub> get the columns constituting the keys & Inherited Keys

call AddJoinKey(T<sub>i</sub>,Keys<sub>i</sub>,InheritedKeys<sub>i</sub>,DP<sub>i</sub>)



insertRoutine(in BaseTable T<sub>i</sub>, Row R<sub>m</sub>, DataRef DP<sub>i</sub>)

locate the entry of T<sub>i</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from row R<sub>m</sub> get the columns constituting the keys & Inherited Keys

call AddJoinKey(T<sub>i</sub>,Keys<sub>i</sub>,InheritedKeys<sub>i</sub>,DP<sub>i</sub>)

Country_Id		Country_Name	
ca		Canada	

Keys

Inherited Keys

Keys

Row R<sub>m</sub>

```
insertRoutine(in BaseTable Ti, Row Rm, DataRef DPi)
```

locate the entry of T<sub>i</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from row R<sub>m</sub> get the columns constituting the keys & Inherited Keys

call AddJoinKey(T<sub>i</sub>, Keys<sub>i</sub>, InheritedKeys<sub>i</sub>, DP<sub>i</sub>)



Country_Id
ca

Keys

COUNTRIES
5

T<sub>i</sub>

0
---

DP<sub>i</sub>

Inherited Keys



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

COUNTRIES
5

T<sub>[i]</sub>

Country_Id
ca

Keys<sub>[i]</sub>

0
---

[DP]<sub>i</sub>

Inherited Keys<sub>[i]</sub>



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

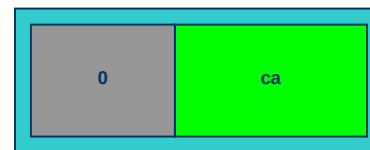
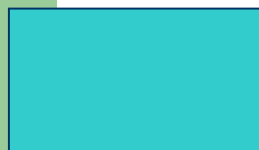
from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

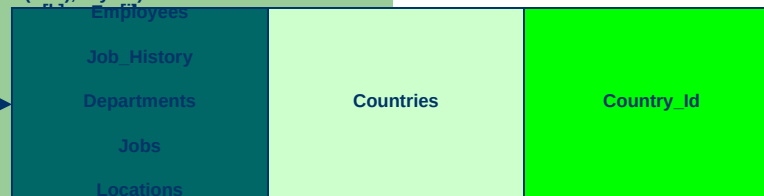
call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

B+Tree(T<sub>5</sub>)



5



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

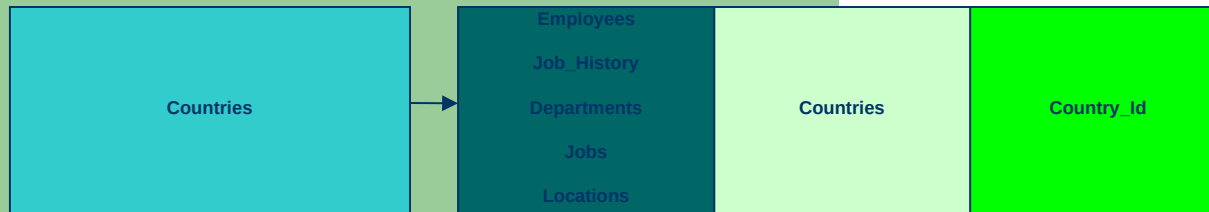
from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

5



Adjacent Table

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

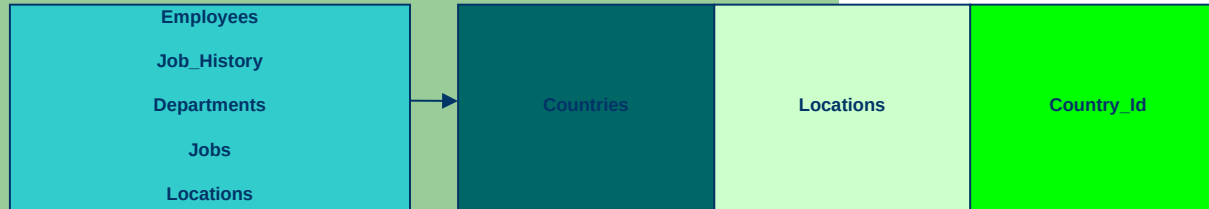
from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

9

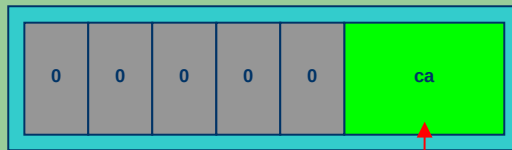


```

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])

```

B+Tree(T<sub>9</sub>)



Found: TRUE

```

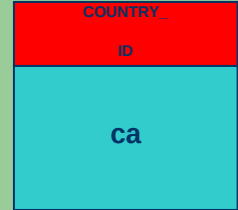
addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DP]i)
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DP]i)
    locate the entry of T[i] in JoinPathList
    from its adjacentList, locate the table T[k] adjacent to it
        locate the entry of T[k] in JoinPathList
        Found = 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])

```

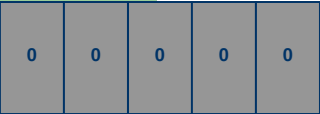
Found: TRUE

```
addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DPi])
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DPi])
    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], [DPk])
            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], [DPik])
            Found = nextKey(B+Tree(T[k]), Keys[i])
```



Keys<sub>[k]</sub>



[DP<sub>k</sub>]

Inherited Keys<sub>[i]</sub>

```

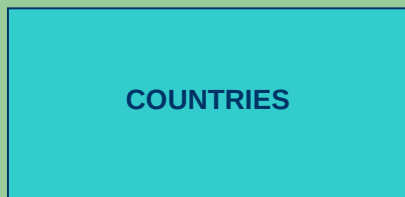
addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DPi])
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DPi])
    locate the entry of T[i] in JoinPathList
    from its adjacentList, locate the table T[k] adjacent to it
        locate the entry of T[k] in JoinPathList
        Found = findKey(B+Tree(T[k]), Keys[i])
        while found do

```

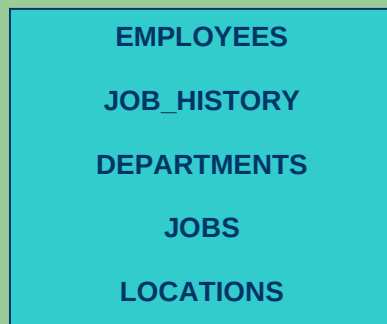
```

            returnKeys(B+Tree(T[k]), Keys[k], InheritedKeys[k], [DPk])
            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], [DPik])
            Found = nextKey(B+Tree(T[k]), Keys[i])

```



T<sub>i</sub>



T<sub>k</sub>



T<sub>ik</sub>

10



```

addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DPi])
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DPi])
    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], [DPk])
            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], [DPik])
            Found = nextKey(B+Tree(T[k]), Keys[i])

```

Employees Job\_History

Departments Jobs

Locations Countries

10

Keys

Inherited Keys

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call AddJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

10

Employees Job\_History

Departments Jobs

Locations Countries

Keys

Inherited Keys

Country\_Id

ca

Keys<sub>[i]</sub>

0

[DP]<sub>i</sub>

Inherited

Keys<sub>[i]</sub>

COUNTRY\_

ID

ca

Keys<sub>[k]</sub>

[DP]<sub>k</sub>

Inherited Keys<sub>[k]</sub>

0	0	0	0	0
---	---	---	---	---

0	0	0	0	0	0
---	---	---	---	---	---

Keys<sub>[ik]</sub>

[DP]<sub>ik</sub>

Inherited Keys<sub>[ik]</sub>



```
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])
```



T<sub>ik</sub>

0	0	0	0	0	0
---	---	---	---	---	---

Keys<sub>[ik]</sub>

[DP]<sub>ik</sub>

Inherited Keys<sub>[ik]</sub>

→ 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

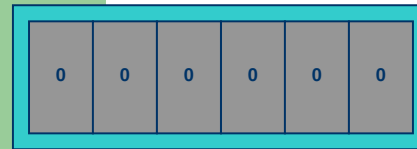
from its adjacentList, locate the definition of Keys & Inherited Keys

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  $_{[i]}$ )

B+Tree( $T_{10}$ )



addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP<sub>i</sub>])

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP<sub>k</sub>])

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP<sub>ik</sub>])

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

10

Employees Job\_History

Departments Jobs

Locations Countries

```

addJoinKey(in Virtual Table T[i], Keys[i], InheritedKeys[i], [DPi])
    call addKey(B+Tree(T[i]), Keys[i], InheritedKeys[i], [DPi])
    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], [DPk])
            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], [DPik])
            Found = nextKey(B+Tree(T[k]), Keys[i])

```

10

Employees Job_History
Departments Jobs
Locations Countries

Adjacent  
Table

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

Found: FALSE

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

Found: FALSE

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

Found: FALSE

addJoinKey(in Virtual Table T<sub>[i]</sub>, Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

call addKey(B+Tree(T<sub>[i]</sub>), Keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, [DP]<sub>i</sub>)

locate the entry of T<sub>[i]</sub> in JoinPathList

from its adjacentList, locate the table T<sub>[k]</sub> adjacent to it

locate the entry of T<sub>[k]</sub> in JoinPathList

Found = findKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

while found do

returnKeys(B+Tree(T<sub>[k]</sub>), Keys<sub>[k]</sub>, InheritedKeys<sub>[k]</sub>, [DP]<sub>k</sub>)

locate the entry of T<sub>[ik]</sub> in JoinPathList

from its adjacentList, locate the definition of Keys & Inherited Keys

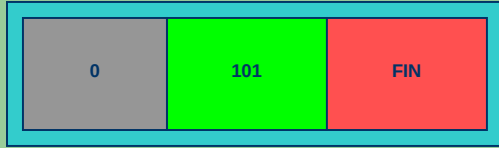
from keys<sub>[i]</sub>, InheritedKeys<sub>[i]</sub>, Keys<sub>[k]</sub> & InheritedKeys<sub>[k]</sub> get the keys & Inherited keys of T<sub>[ik]</sub>

call addJoinKey(T<sub>[ik]</sub>, Keys<sub>[ik]</sub>, InheritedKeys<sub>[ik]</sub>, [DP]<sub>ik</sub>)

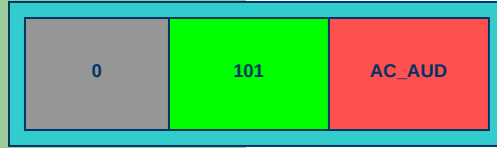
Found = nextKey(B+Tree(T<sub>[k]</sub>), Keys<sub>[i]</sub>)

Found: FALSE

B+Tree( $T_0$ )



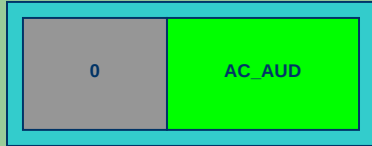
B+Tree( $T_1$ )



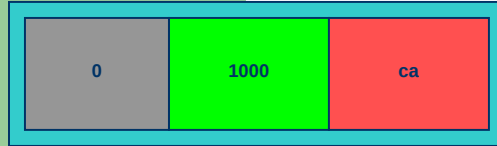
B+Tree( $T_2$ )



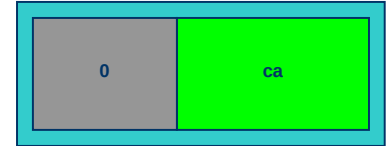
B+Tree( $T_3$ )



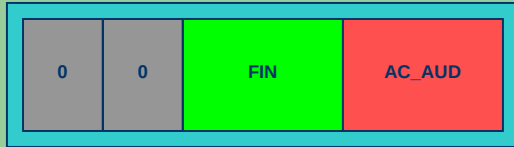
B+Tree( $T_4$ )



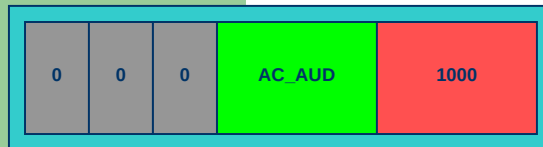
B+Tree( $T_5$ )



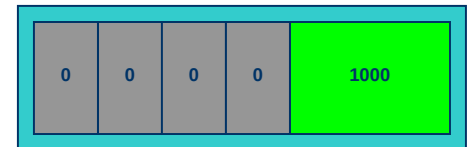
B+Tree( $T_6$ )



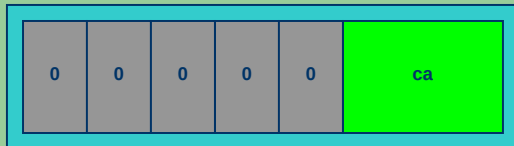
B+Tree( $T_7$ )



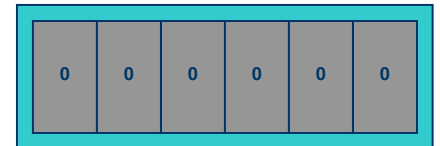
B+Tree( $T_8$ )



B+Tree( $T_9$ )



B+Tree( $T_{10}$ )





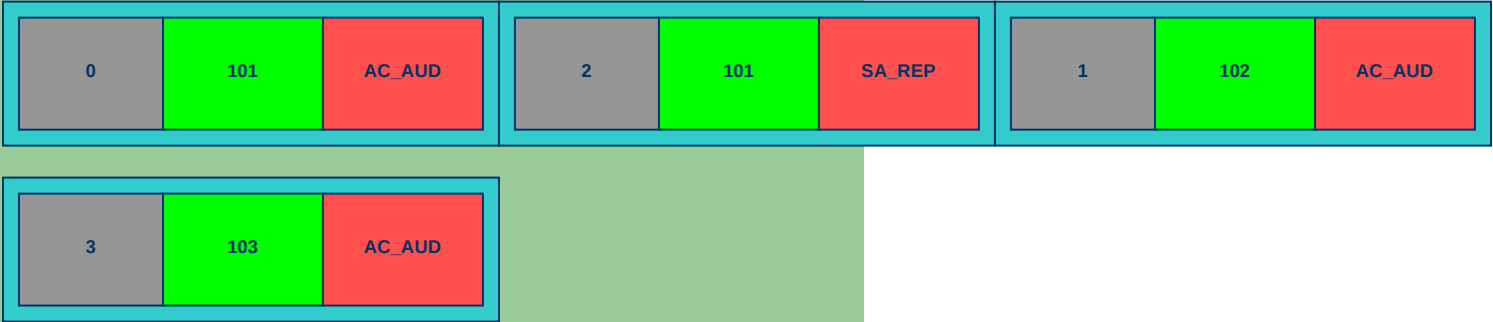
As we can notice from the last index we have an element with 6 data pointers respectively pointing to the 6 base tables forming the virtual table  $T_{10}$ , with all values equal to the first row on each table, those rows are in join together.

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( $T_0$ )



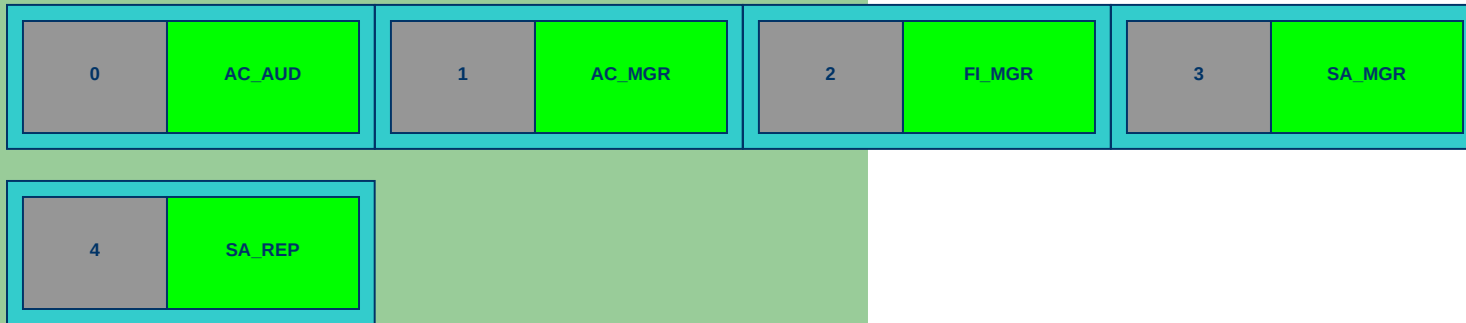
B+Tree( $T_1$ )



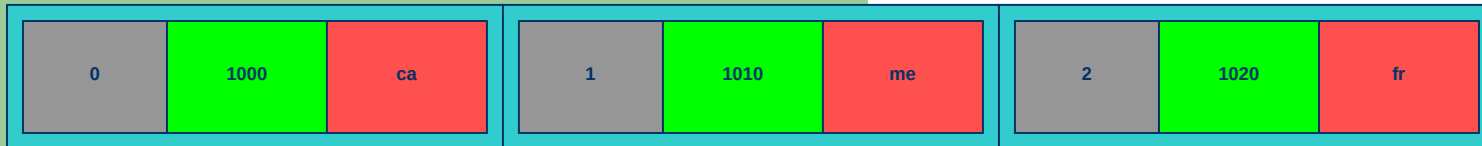
B+Tree( $T_2$ )



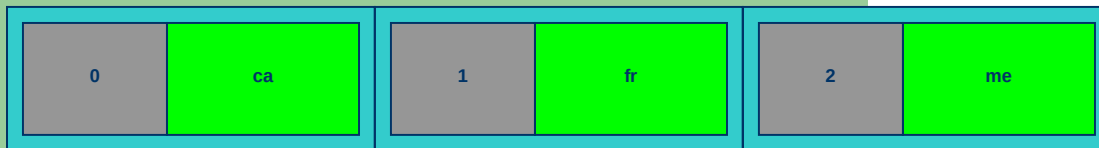
B+Tree(T<sub>3</sub>)



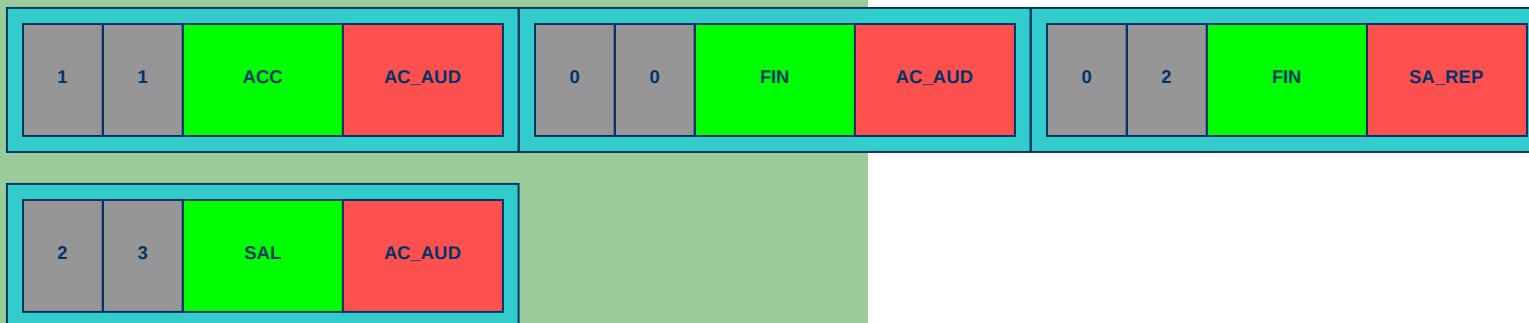
B+Tree(T<sub>4</sub>)



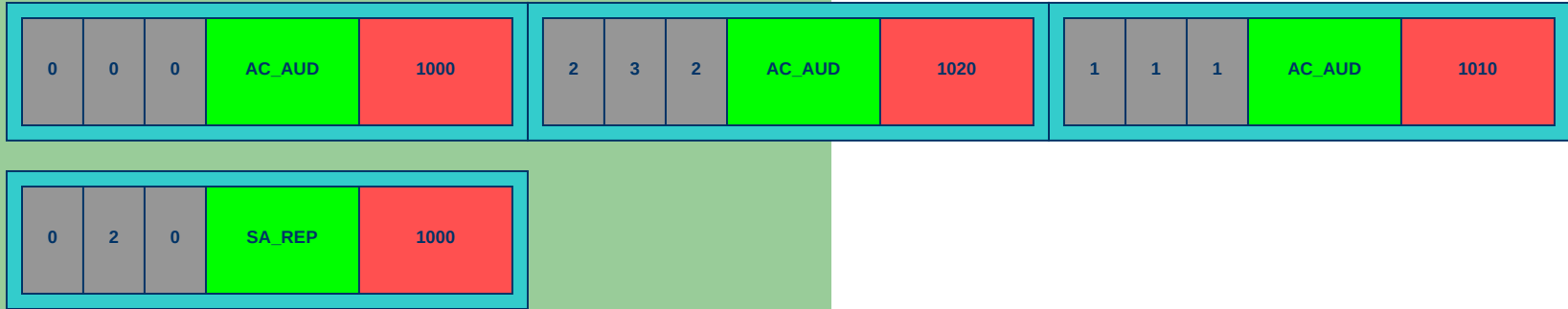
B+Tree(T<sub>5</sub>)



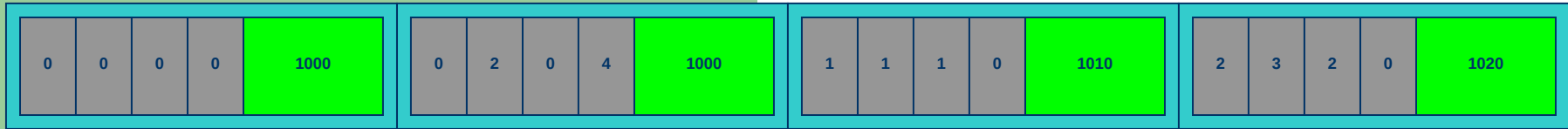
B+Tree(T<sub>6</sub>)



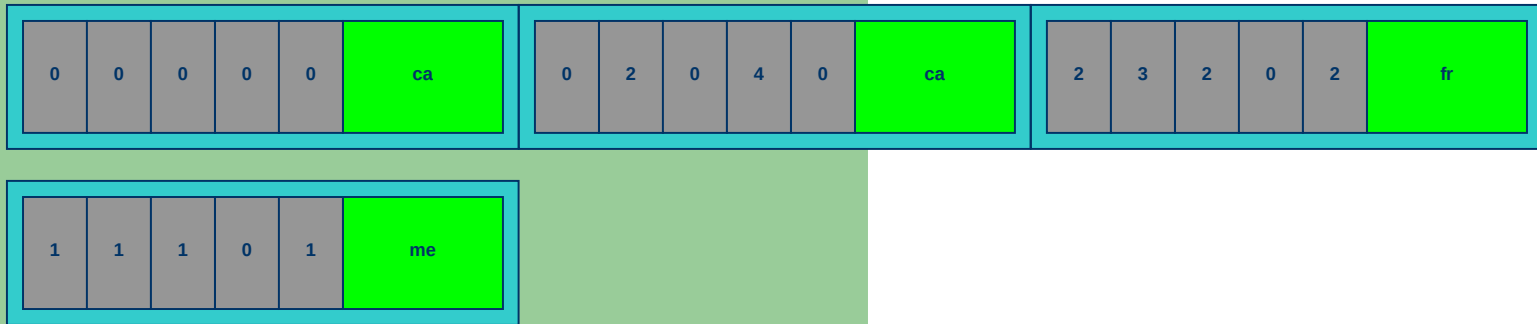
B+Tree(T<sub>7</sub>)



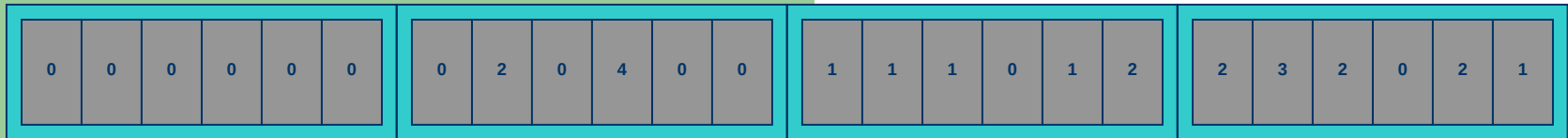
B+Tree(T<sub>8</sub>)



B+Tree(T<sub>9</sub>)



B+Tree(T<sub>10</sub>)



# 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<sup>+</sup>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<sup>+</sup>Tree( $T_{[k]}$ ),  $Keys_{[i]}$ )
  - While found( $keys_{[i]}$ ) do
    - ReturnKeys(B<sup>+</sup>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<sup>+</sup>Tree( $T_{[k]}$ ),  $Keys_{[i]}$ )

# B $\bowtie$ 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  $T_0$  to  $T_i$  and from  $T_i$  to  $T_0$ .

The index number start always with 0.

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

This algorithm is linear, is  $2*n - 1$ .

# Complexity of the algorithm for the creation of JoinPathList.

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

**Proof:**

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

**For  $m = 1$ :**

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

**For  $m = n-1$ :**

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

**For  $m = n$ :**

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

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

So when the algorithm run at certain point should execute:

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

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

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<sup>+</sup>tree mechanism to drive us in the insert for the join.**

**Suppose that the order of the B<sup>+</sup>Trees is m and the number of elements for every B<sup>+</sup>Tree with i as index from the (2\*n-1) B<sup>+</sup>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 a row that need to call recursively all other B<sup>+</sup>Trees, the insert procedure will be running for  $(2*n - 1) - (n - 1) = n$  times.**

**The complexity will be:**

**$\text{Ord}(n * \log_m(l_i * p_i))$**



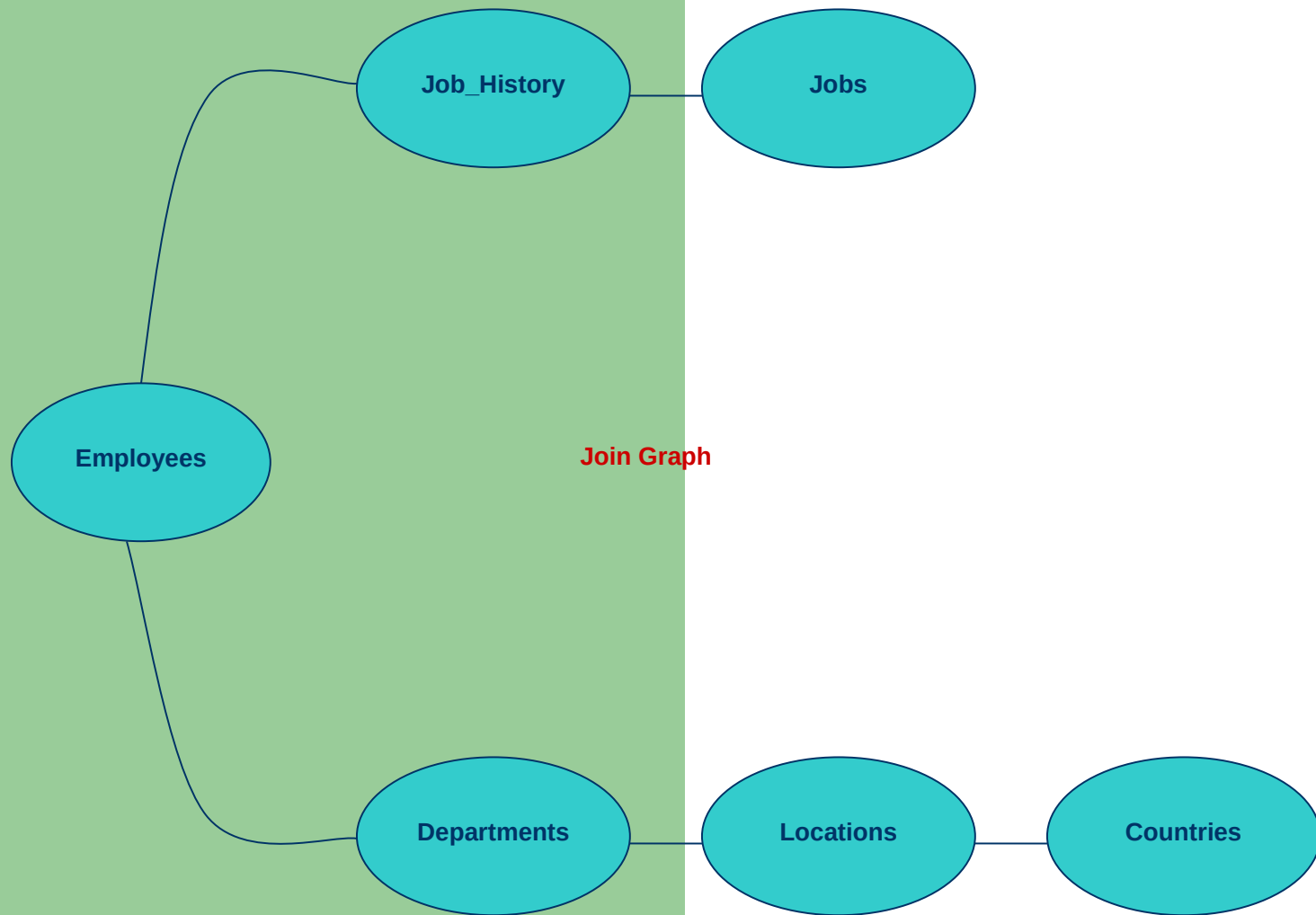
# Complexity of the algorithm for the other operations.

The only B<sup>+</sup>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<sup>+</sup>Tree for find, search, prev, next,... are the same as for normal B<sup>+</sup>Tree with the same number of elements.

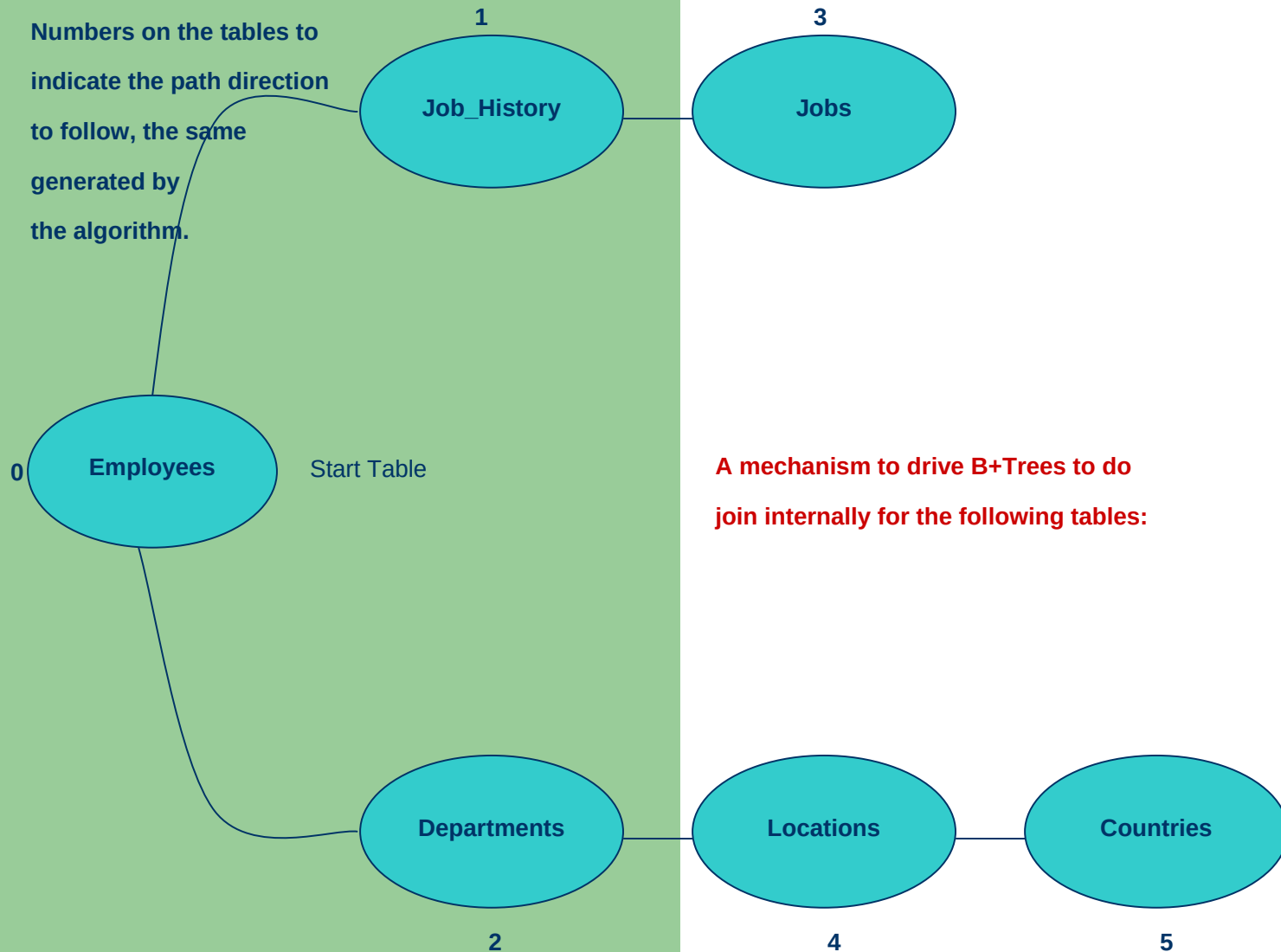
# **Proof of correctness.**

**To prove the correctness of the algorithm  
let see how does the algorithm work for  
example A 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:

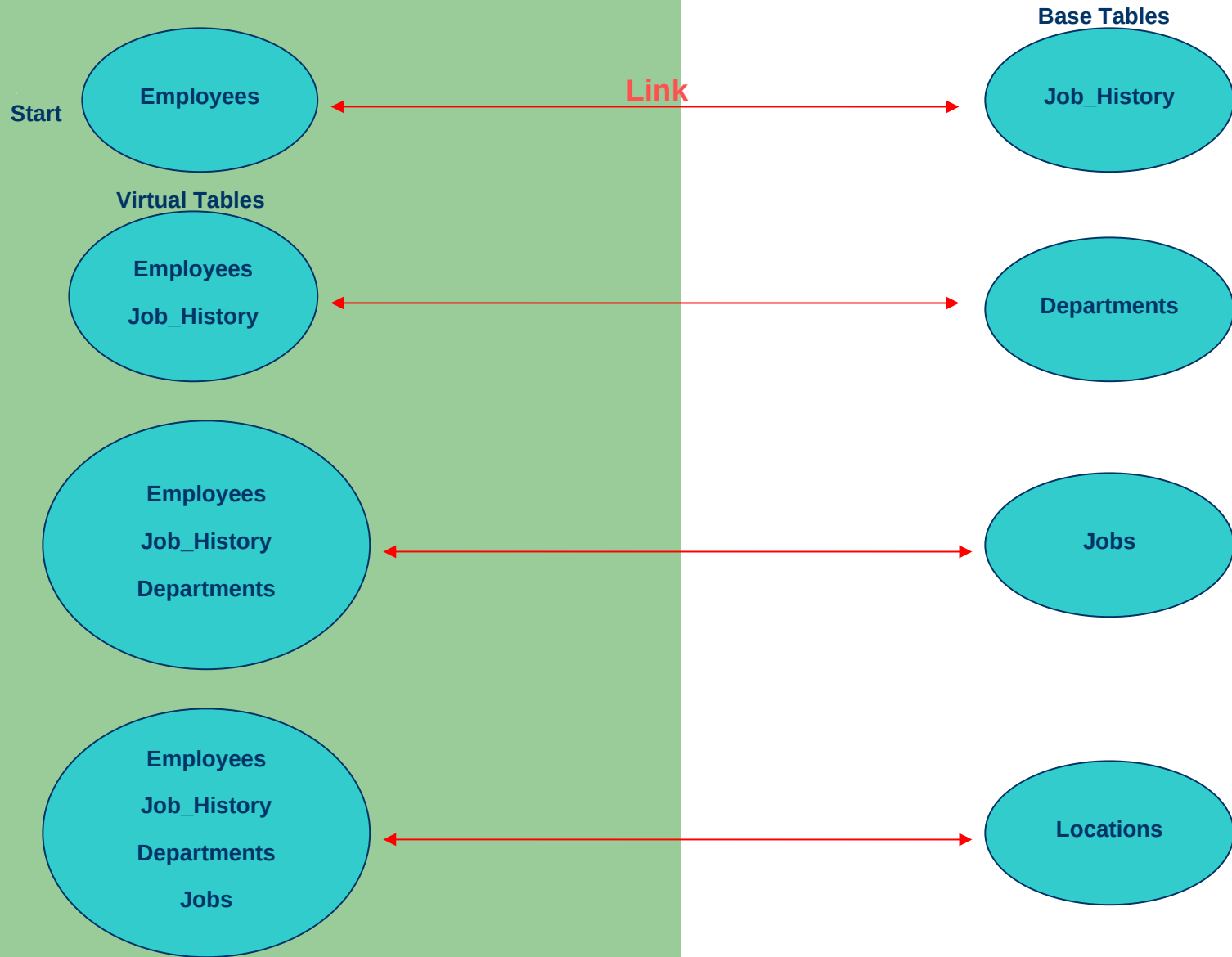
`generateJoinPathList`

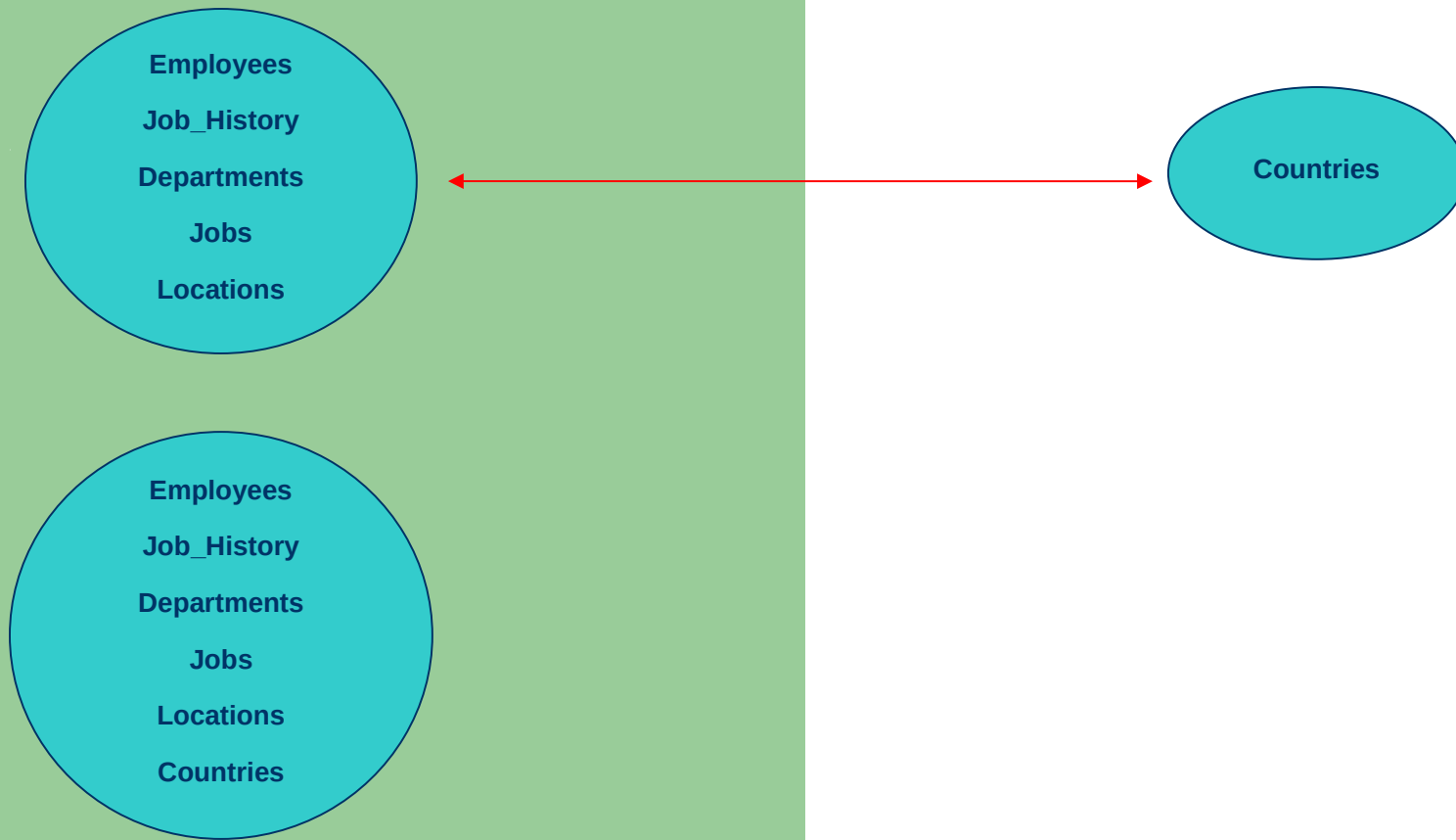


Notice that in the path if we reach one table it is not necessary to continue from it.

This is very important because this makes the tables free from any order, independent selection of the start table.

By grouping comes out: `





There is no restriction on how the rows from base tables get inserted (Any order with any sequence).

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.

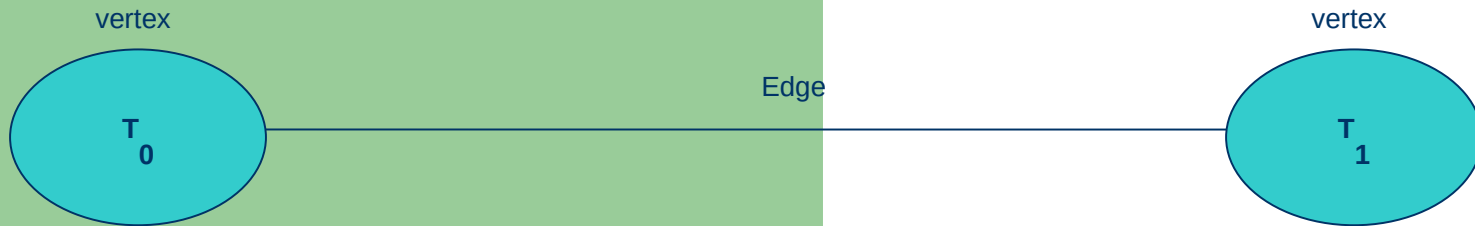
**Notice that what we show before is independent from the number of tables, so that the same reasoning apply to any number of tables and the proof of correctness could be easily proved by induction.**



Let prove the correctness by induction.

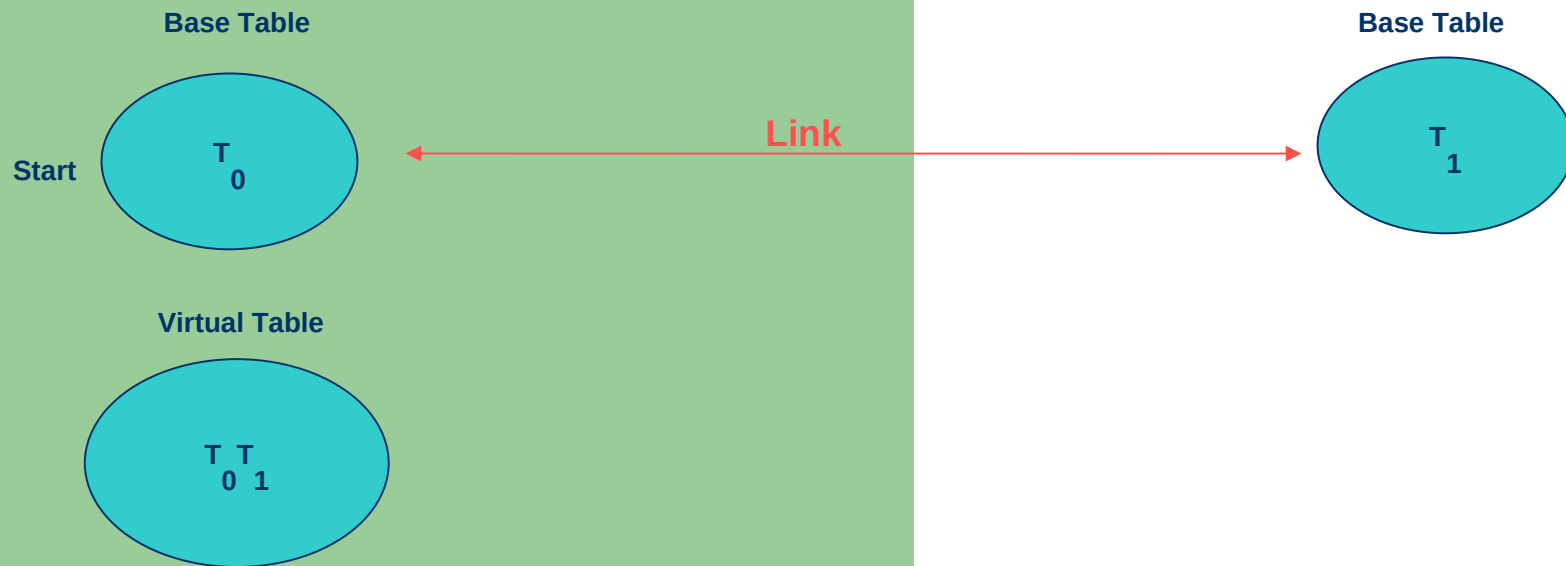
To do it, let see the correctness for 2 tables  $T_0$  and  $T_1$  in join together.

The join graph should be the following:

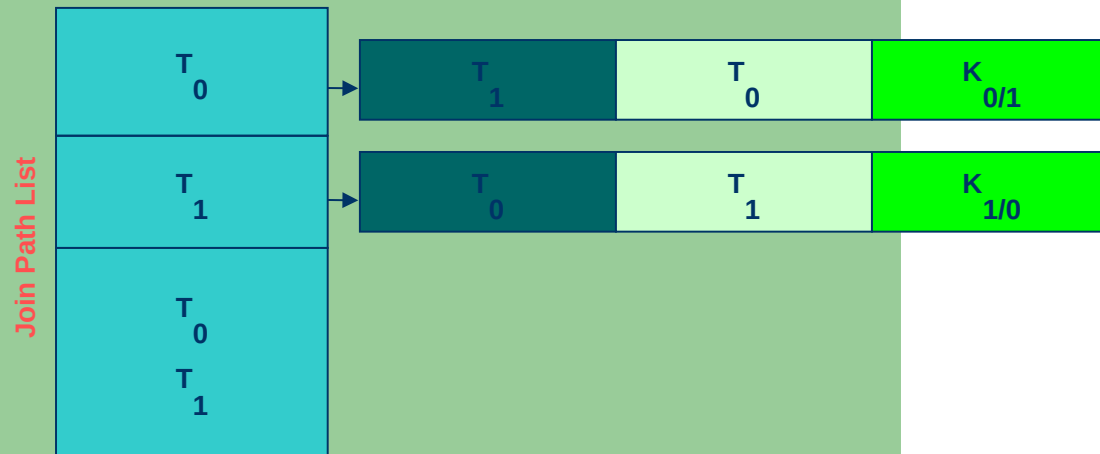


There are just 2 paths between the 2 tables: or from  $T_0$  going toward  $T_1$  or vice versa, let consider the former, the second case is symmetric and after all  $T_0$  and  $T_1$  are of arbitrarily choice.

By grouping comes out:



So, the JoinPathList should be the following:



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.

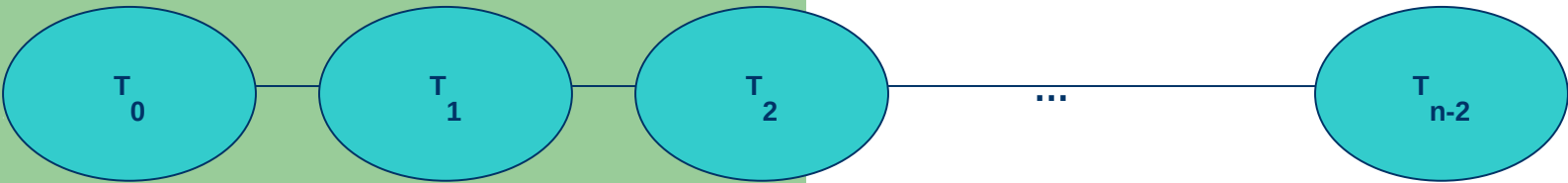
If such a couple of data pointers exist, it comes out because there is 2 keys belonging to the rows pointed by the data pointers and such keys satisfy the join condition, this is in contradiction on what we assume initially.

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  $n$ th 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.

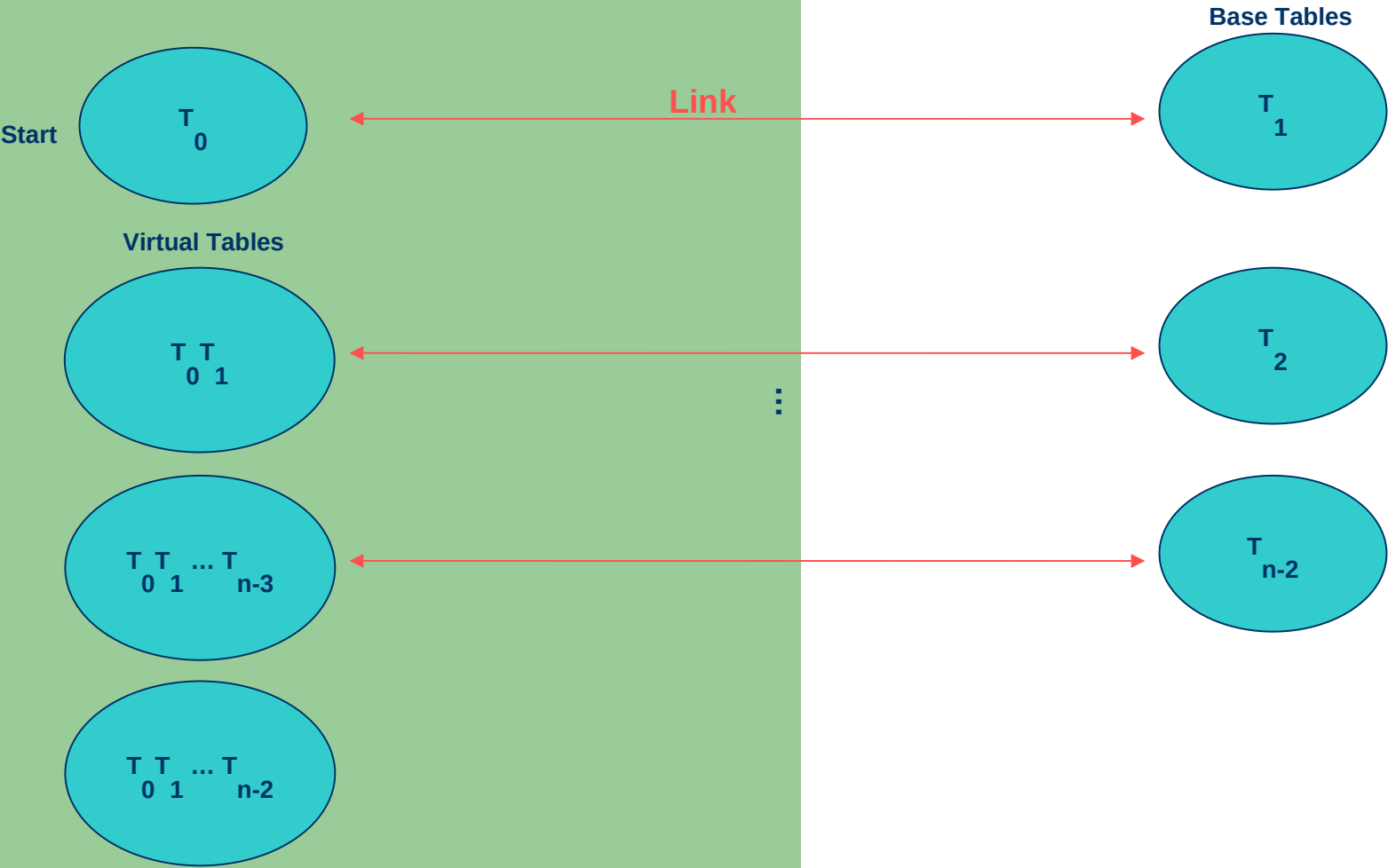
Let see first the Join Graph for the  $(n-1)$  tables and how they went in group and later what happens when we consider the  $n$ th table.

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

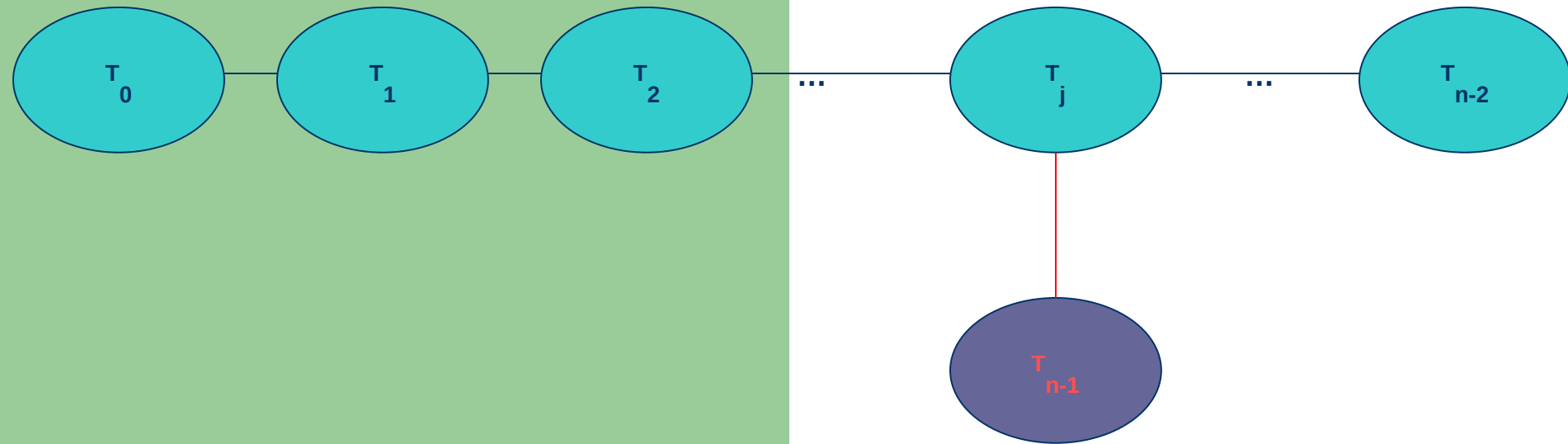


Suppose that the choice of  $T_0 \dots T_{n-2}$  are in the way that the path start from  $T_0$ , continue by  $T_1 \dots T_{n-3}$  till the end to arrive at  $T_{n-2}$ .

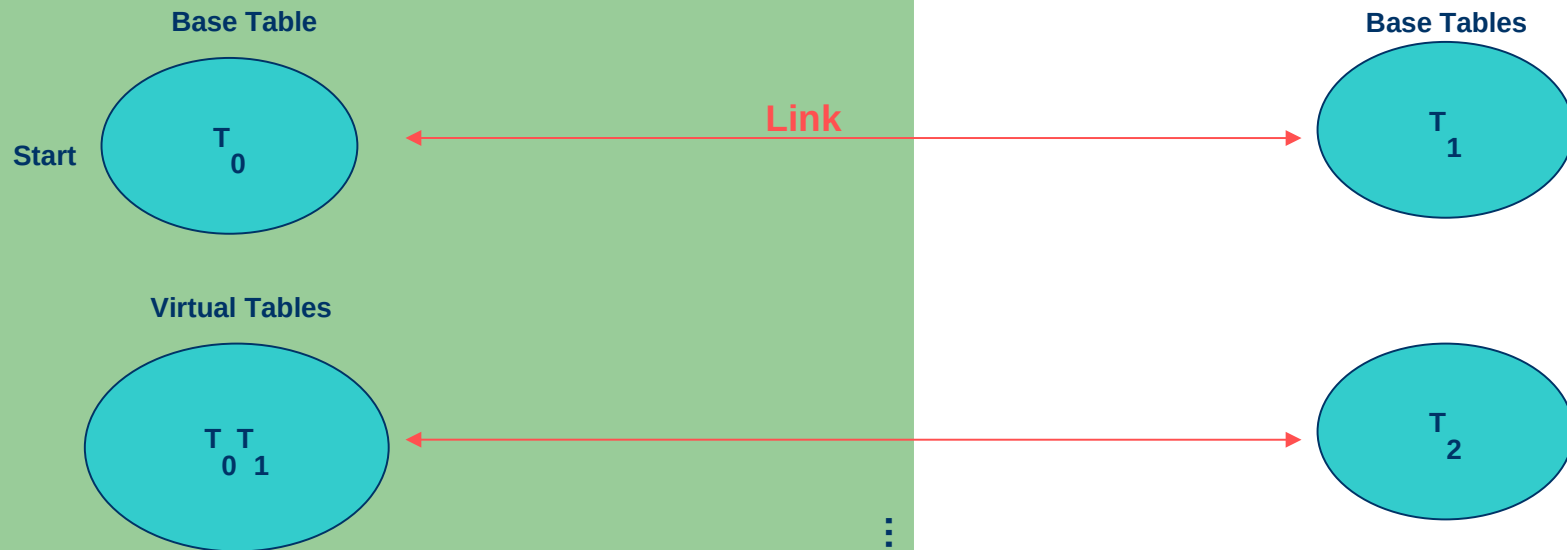
By grouping comes out:

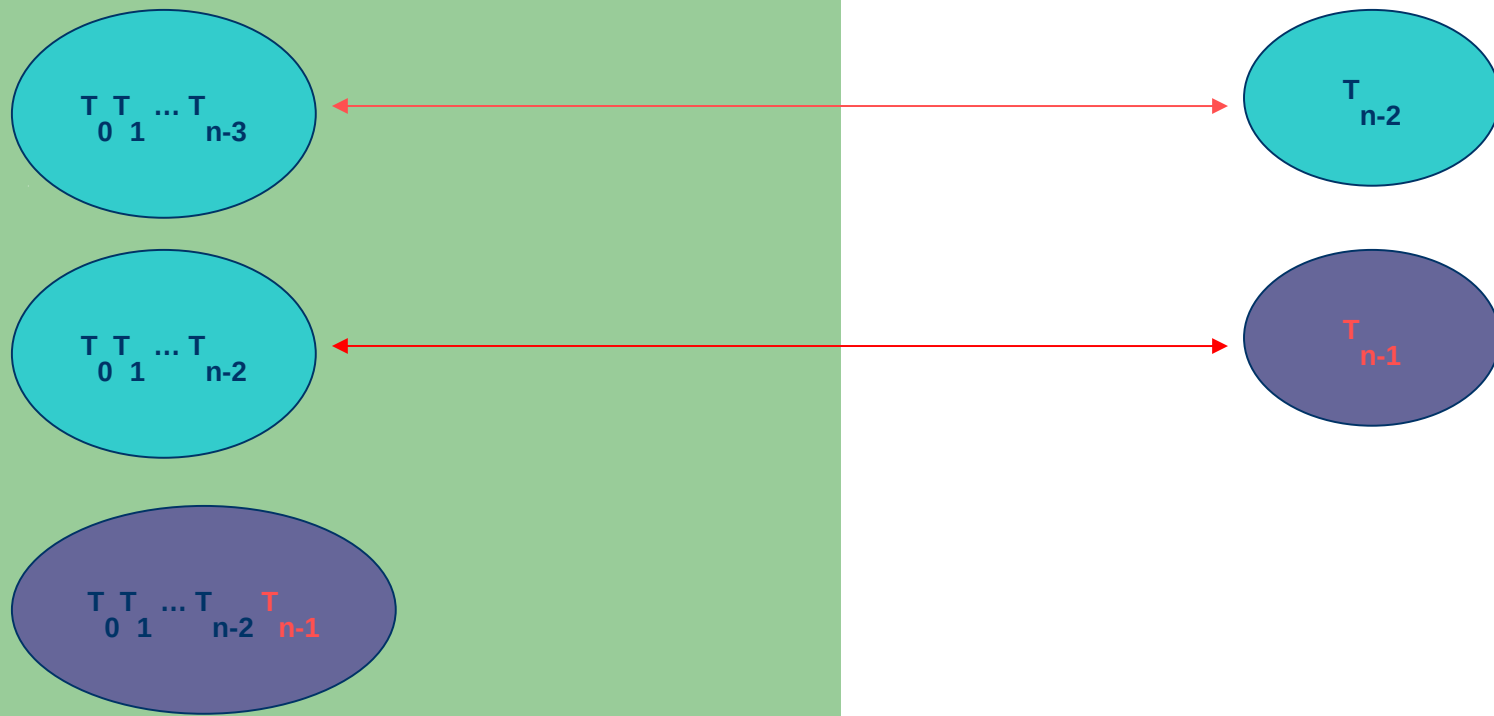


If there is one more table, the join graph would be the following:



The path should look like this:





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 the common key should be from table  $T_j$ .

So we have the following situation:

Table  $T_0 \dots T_{n-2}$  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  $T_0 \dots T_{n-2}$  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  $T_0 \dots T_{n-2}$  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  $T_0 \dots T_{n-2}$  to  $T_j$  and the eventual keys from  $T_0 \dots T_{n-1}$  to some base tables in the base tables  $T_0 \dots T_{n-1}$  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 in the next slide.

# Employees table

DP start from 0

	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

→ 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 (A)	Employees/Supervisor_Id (B)
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 (A)	Employees/Supervisor_Id (B)
0	1

Employees/Employee_Id (A)
Employees/Supervisor_Id (B)

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 (A)	Employees/Supervisor_Id (B)
0	1

Employees/Employee_Id (A)
Employees/Supervisor_Id (B)

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 (A)	Employees/Supervisor_Id (B)
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

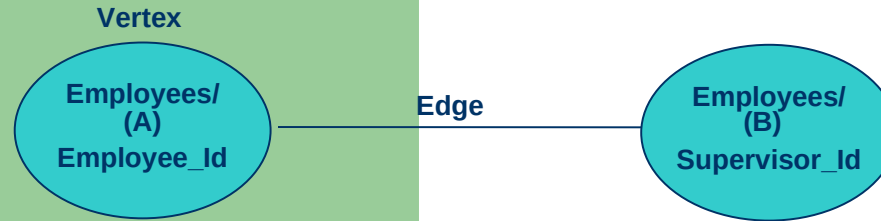


Employees/Employee_Id (A)	Employees/Supervisor_Id
0	1





## 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 T0 ... Tm be the base tables
create 2 dynamic arrays queue and path

insert T0 into path
insert T0 into queue
repeat
    TElement = First Table in queue
    for every Link Item in Adjacent Link of TElement 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 TElement from queue
until queue is empty
```

Join Base Tables

Employees/Employee_Id (A)	Employees/Supervisor_Id (B)
---------------------------	-----------------------------

Join Graph



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

```
→ let T0 ... Tm be the base tables
create 2 dynamic arrays queue and path

insert T0 into path

insert T0 into queue

repeat
    TElement = First Table in queue
    for every Link Item in Adjacent Link of TElement 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 TElement from queue
until queue is empty
```

Join Base Tables

Employees/Employee_Id (A)	Employees/Supervisor_Id (B)
---------------------------	-----------------------------

T<sub>0</sub>

T<sub>1</sub>

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

$T_{\text{Element}}$  = First Table in queue

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

        if the Link Item is in the join sequence then

            if path doesn't contain the Link Item then

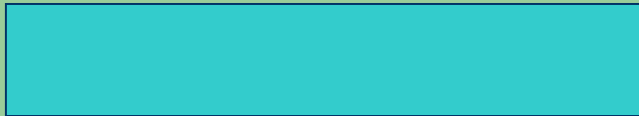
                insert Link Item into path

                insert Link Item into queue

    remove  $T_{\text{Element}}$  from queue

until queue is empty

queue



path



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

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  from queue

until queue is empty

Join Base Tables

Employees/Employee_Id (A)	Employees/Supervisor_Id (B)
---------------------------	-----------------------------

$T_0$

$T_1$

queue

--

path

Employees/Employee_Id (A)
---------------------------

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

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  from queue

until queue is empty

Join Base Tables

Employees/Employee_Id (A)	Employees/Supervisor_Id (B)
---------------------------	-----------------------------

$T_0$

$T_1$

queue

Employees/Employee_Id (A)
---------------------------

path

Employees/Employee_Id (A)
---------------------------

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

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  from queue

until queue is empty

queue

Employees/Employee\_Id (A)

path

Employees/Employee\_Id (A)

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

→  $T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  from queue

until queue is empty

queue



Employees/Employee\_Id (A)

path

Employees/Employee\_Id (A)



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

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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  $T_{\text{Element}}$  from queue

until queue is empty

queue



Employees/Employee\_Id (A)

path

Employees/Employee\_Id (B)

Join Graph

Employees/Employee\_Id (A)

Employees/Supervisor\_Id (B)

Employee\_Id

Employees/Supervisor\_Id (B)

Employees/Employee\_Id (A)

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

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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

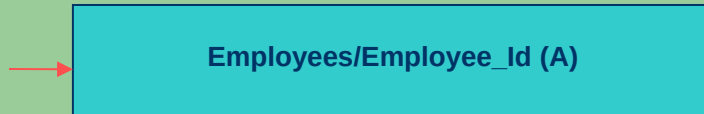
remove  $T_{\text{Element}}$  from queue

until queue is empty

## Join Base Tables



queue



path



## Join Graph



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

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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

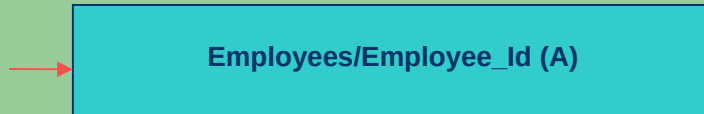
remove  $T_{\text{Element}}$  from queue

until queue is empty

## Join Base Tables



queue



path



## Join Graph



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

```
let T0...Tm be the base tables
create 2 dynamic arrays queue and path

insert T0 into path

insert T0 into queue

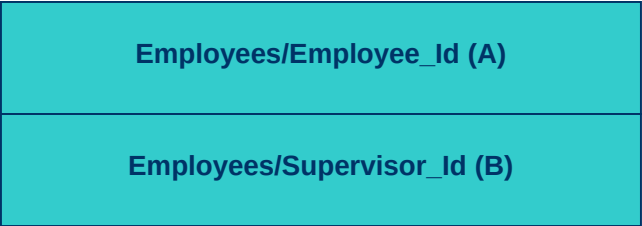
repeat
    TElement = First Table in queue
    for every Link Item in Adjacent Link of TElement from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue

    remove TElement from queue
until queue is empty
```

queue



path

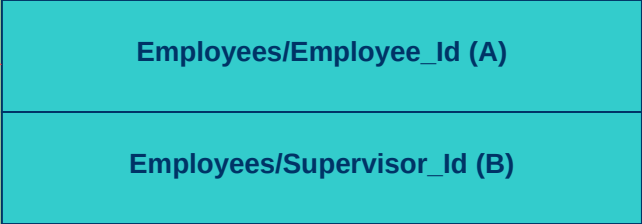


generateJoinPathList (in Join Base Tables, JoinGraph; out JoinPathList)

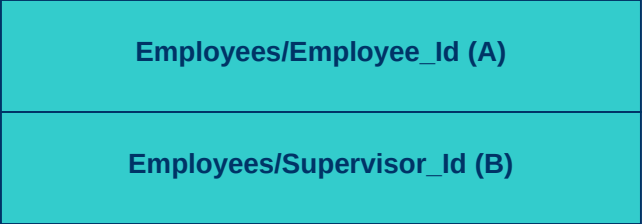
```
let T0...Tm be the base tables
create 2 dynamic arrays queue and path

insert T0 into path
insert T0 into queue
repeat
    TElement = First Table in queue
    for every Link Item in Adjacent Link of TElement 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 TElement from queue
until queue is empty
```

queue



path



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

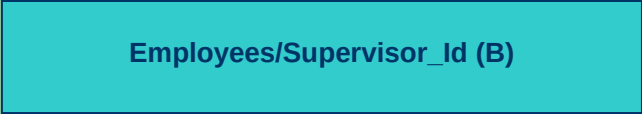
```
let T0 ... Tm be the base tables
create 2 dynamic arrays queue and path

insert T0 into path

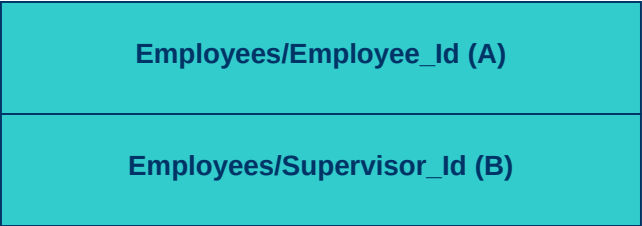
insert T0 into queue

repeat
    TElement = First Table in queue
    for every Link Item in Adjacent Link of TElement 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 TElement from queue
until queue is empty
```

queue



path



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

→  $T_{\text{Element}}$  = First Table in queue

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

if the Link Item is in the join sequence then

if path doesn't contain the Link Item then

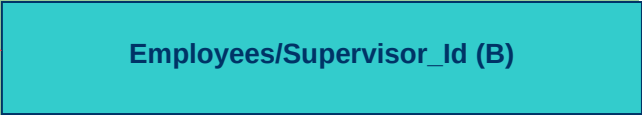
insert Link Item into path

insert Link Item into queue

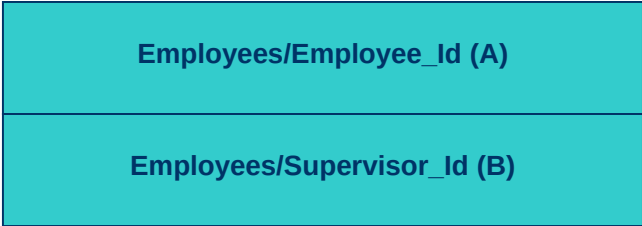
remove  $T_{\text{Element}}$  from queue

until queue is empty

queue



path



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

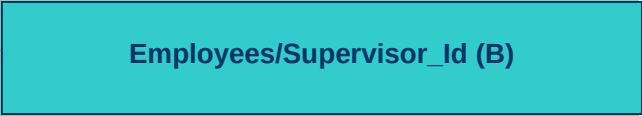
insert Link Item into path

insert Link Item into queue

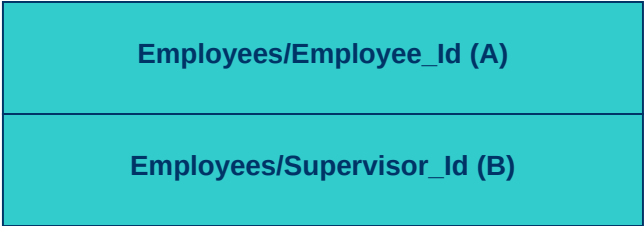
remove  $T_{Element}$  from queue

until queue is empty

queue



path



Join Graph





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

$T_{\text{Element}}$  = First Table in queue

for every Link Item in Adjacent Link of  $T_{\text{Element}}$  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

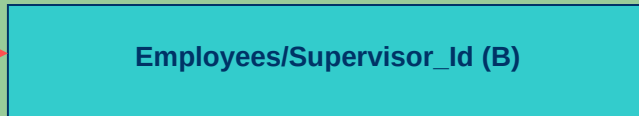
remove  $T_{\text{Element}}$  from queue

until queue is empty

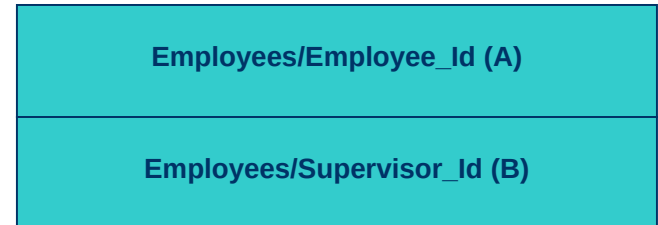
### Join Base Tables



queue



path



### Join Graph



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

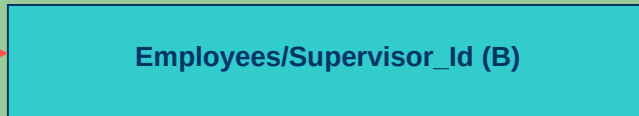
remove  $T_{Element}$  from queue

until queue is empty

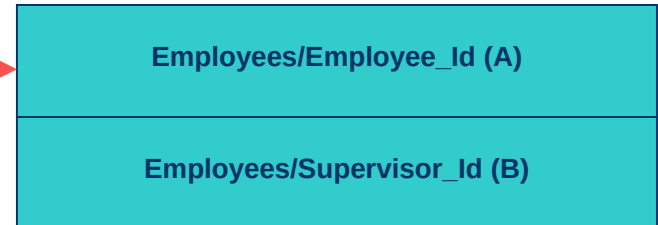
## Join Base Tables



queue



path



## Join Graph



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

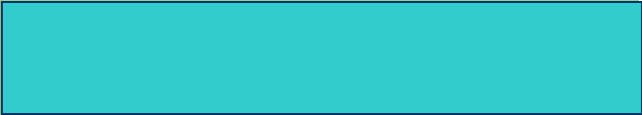
```
let T0 ... Tm be the base tables
create 2 dynamic arrays queue and path

insert T0 into path

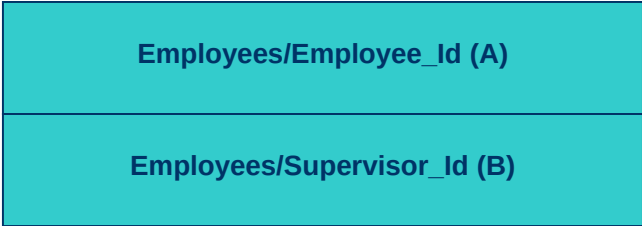
insert T0 into queue

repeat
    TElement = First Table in queue
    for every Link Item in Adjacent Link of TElement 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 TElement from queue
until queue is empty
```

queue



path



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

```
let T0 ... Tm be the base tables
create 2 dynamic arrays queue and path

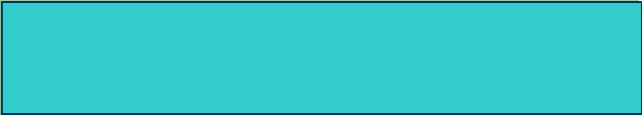
insert T0 into path

insert T0 into queue

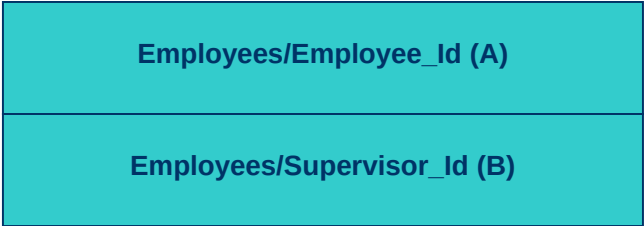
repeat
    TElement = First Table in queue
    for every Link Item in Adjacent Link of TElement from the Join Graph do
        if the Link Item is in the join sequence then
            if path doesn't contain the Link Item then
                insert Link Item into path
                insert Link Item into queue

remove TElement from queue
until queue is empty
```

queue



path



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

create a local buffer buf

insert into buf the first entry from path

for all the remainder entries in path do

take one  $T_i$  at a time

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

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

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

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

$T_{[buf]} + = T_i$

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

Nodes

Employees/Employee\_Id (A)

Employees/Supervisor\_Id (B)

path

Employees/Employee\_Id (A)

Employees/Supervisor\_Id (B)

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

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

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

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

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

$T_{[buf]} + = T_i$

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

Nodes

Employees/Employee\_Id (A)

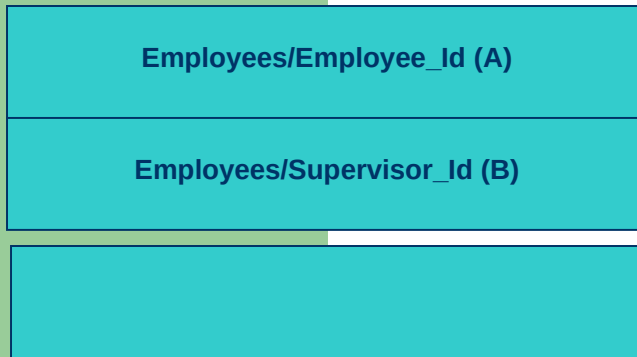
Employees/Supervisor\_Id (B)

path

Employees/Employee\_Id (A)

Employees/Supervisor\_Id (B)

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

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

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

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

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

$T_{[buf]} + = T_i$

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

Nodes

Employees/Employee\_Id (A)

Employees/Supervisor\_Id (B)

path

Employees/Employee\_Id (A)

Employees/Supervisor\_Id (B)

buf

Employees/Employee\_Id (A)

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

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

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

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

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

$T_{[buf]} + T_i$

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

## Nodes

Employees/Employee\_Id (A)

Employees/Supervisor\_Id (B)

path

Employees/Employee\_Id (A)

Employees/Supervisor\_Id (B)

buf

Employees/Employee\_Id (A)



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

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

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

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

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

$T_{[buf]} + = T_i$

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

## Nodes

Employees/Employee\_Id (A)

Employees/Supervisor\_Id (B)

path

$T_i$

buf

Employees/Employee\_Id (A)

Employees/Supervisor\_Id (B)

Employees/Employee\_Id (A)

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



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

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

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

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

$T_{[buf]} + T_i$

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

## Nodes

Employees/Employee\_Id (A)

Employees/Supervisor\_Id (B)

Employees/Employee\_Id (A)

Employees/Employee\_Id (A)

Employees/Supervisor\_Id (B)

Employees/Employee\_Id (A)

path

$T_i$

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

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

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

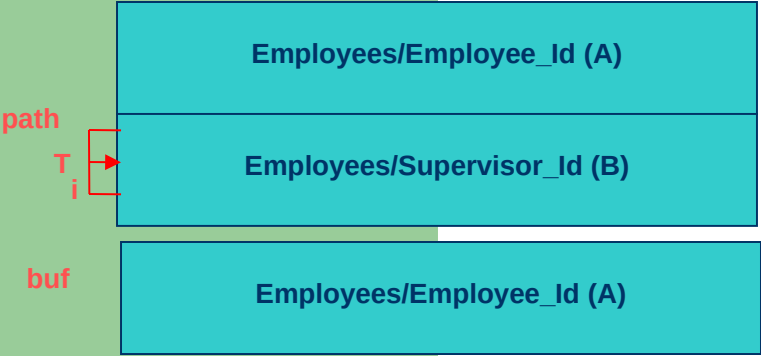
$\text{JoinPathAdjacentList}(T_{[buf]}) = T_i$

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

$T_{[buf]} += T_i$

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

Nodes



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

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

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

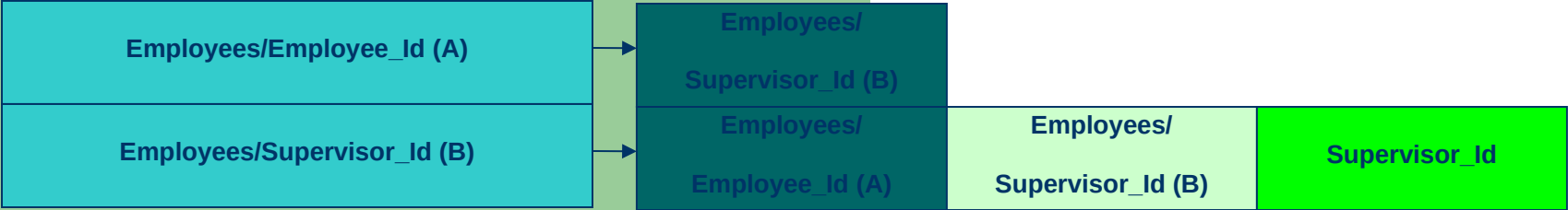
→  $\text{JoinPathAdjacentList}(T_{[buf]}) = T_i$

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

$T_{[buf]} += T_i$

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

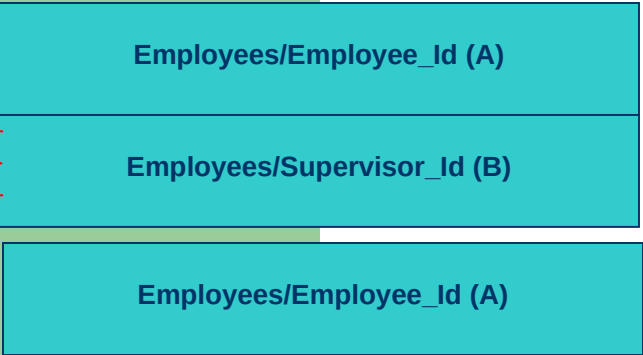
Nodes



path

$T_i$

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

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

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

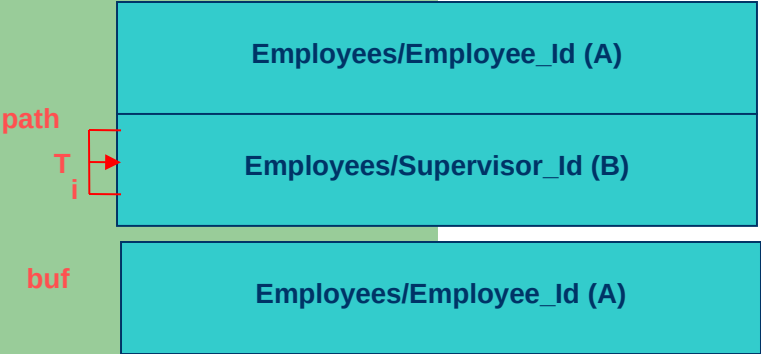
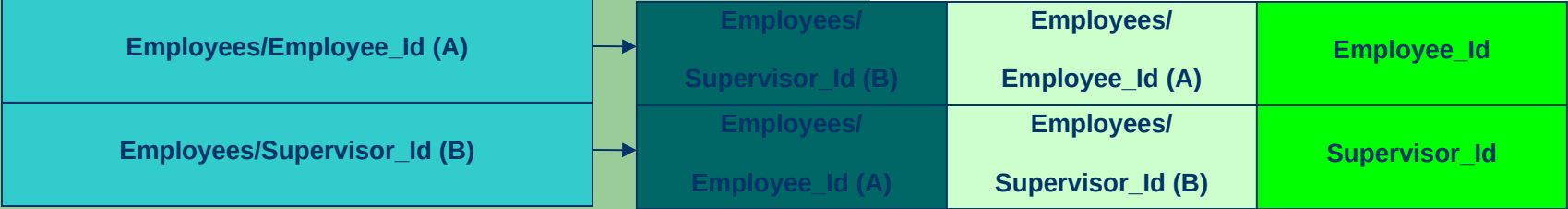
$\text{JoinPathAdjacentList}(T_{[buf]}) = T_i$

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

$T_{[buf]} += T_i$

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

Nodes



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

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

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

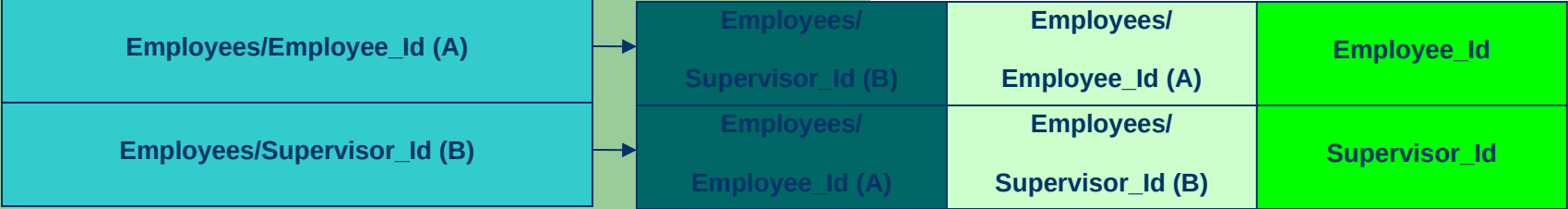
$\text{JoinPathAdjacentList}(T_{[buf]}) = T_i$

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

→  $T_{[buf]} += T_i$

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

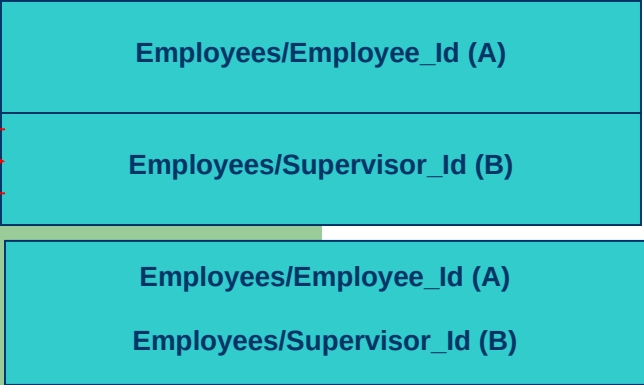
Nodes



path

$T_i$

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

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

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

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

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

$T_{[buf]} += T_i$

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

Nodes

Join Path List

Employees/Employee_Id (A)	Employees/ Supervisor_Id (B)	Employees/ Employee_Id (A)	Employee_Id
Employees/Supervisor_Id (B)	Employees/ Employee_Id (A)	Employees/ Supervisor_Id (B)	Supervisor_Id
Employees/Employee_Id (A)			
Employees/Supervisor_Id (B)			

path

$T_i$

buf

Employees/Employee_Id (A)
Employees/Supervisor_Id (B)
Employees/Employee_Id (A)
Employees/Supervisor_Id (B)



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  $\langle \text{Table}, \text{key} \rangle$  in buf do

take one couple buf<sub>c</sub> at a time

if (buf<sub>c</sub>.Table =  $T_k$ ) then

if (buf<sub>c</sub> != Key( $T_{[i]}$ )) and (buf<sub>c</sub> not in InheritedKey( $T_{[i]}$ )) then

InheritedKey( $T_{[i]}$ ) += buf<sub>c</sub>

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

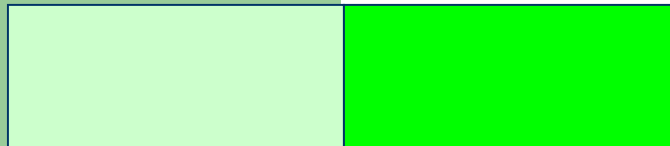
buf.Table +=  $T_l$

buf.key +=  $K_{k_l}$

Table

buf

Key





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  $\langle \text{Table}, \text{key} \rangle$  in buf do

take one couple  $\text{buf}_c$  at a time

if  $(\text{buf}_c.\text{Table} = T_k)$  then

if  $(\text{buf}_c.\text{Key} \neq \text{Key}(T_{[i]}))$  and  $(\text{buf}_c \text{ not in InheritedKey}(T_{[i]}))$  then

$\text{InheritedKey}(T_{[i]}) += \text{buf}_c$

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

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

$\text{buf}.\text{Table} += T_{[i]}$

$\text{buf}.\text{key} += K_{[i]}$

Table

Key

--	--

Join Path List

Employees/ Employee_Id (A)
Employees/ Supervisor_Id (B)
Employees/ Employee_Id (A)  Employees/ Supervisor_Id (B)

Employees/ Supervisor_Id (B)	Employees/ Employee_Id (A)	Employee_Id
Employees/ Employee_Id (A) (A)	Employees/ Supervisor_Id (B)	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  $\langle \text{Table}, \text{key} \rangle$  in buf do

take one couple  $\text{buf}_c$  at a time

if  $(\text{buf}_c.\text{Table} = T_k)$  then

if  $(\text{buf}_c.\text{Key} \neq \text{Key}(T_{[i]}))$  and  $(\text{buf}_c \text{ not in InheritedKey}(T_{[i]}))$  then

$\text{InheritedKey}(T_{[i]}) += \text{buf}_c$

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

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

$\text{buf}.\text{Table} += T_{[i]}$

$\text{buf}.\text{key} += K_{[i]}$

Table

Key

--	--

Join Path List

$T_{[i]}$

Employees/ Employee_Id (A)
Employees/ Supervisor_Id (B)
Employees/ Employee_Id (A) Employees/ Supervisor_Id (B)

Employees/ Supervisor_Id (B)	Employees/ Employee_Id (A)	Employee_Id
Employees/ Employee_Id (A)	Employees/ Supervisor_Id (B)	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  $\langle \text{Table}, \text{key} \rangle$  in buf do

take one couple  $\text{buf}_c$  at a time

if  $(\text{buf}_c.\text{Table} = T_k)$  then

if  $(\text{buf}_c.\text{Key} \neq \text{Key}(T_{[i]}))$  and  $(\text{buf}_c$  not in  $\text{InheritedKey}(T_{[i]}))$  then

$\text{InheritedKey}(T_{[i]}) += \text{buf}_c$

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

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

$\text{buf}.\text{Table} += T_l$

$\text{buf}.\text{key} += K_{k_l}$

Table

Key

--	--

Join Path List

$T_{[i]}$

Employees/ Employee_Id (A)
Employees/ Supervisor_Id (B)
Employees/ Employee_Id (A) Employees/ Supervisor_Id (B)

Employees/ Supervisor_Id (B)	Employees/ Employee_Id (A)	Employee_Id
Employees/ Employee_Id (A)	Employees/ Supervisor_Id (B)	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  $\langle \text{Table}, \text{key} \rangle$  in buf do

take one couple buf<sub>c</sub> at a time

if (buf<sub>c</sub>.Table =  $T_k$ ) then

if (buf<sub>c</sub>.Key != Key( $T_{[i]}$ )) and (buf<sub>c</sub> not in InheritedKey( $T_{[i]}$ )) then

InheritedKey( $T_{[i]}$ ) += buf<sub>c</sub>

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 +=  $K_{k_l}$

Table

Key

--	--

Join Path List

$T_{[i]}$

Employees/ Employee_Id (A)
Employees/ Supervisor_Id (B)
Employees/ Employee_Id (A) Employees/ Supervisor_Id (B)

Employees/ Supervisor_Id (B)	Employees/ Employee_Id (A)	Employee_Id
Employees/ Employee_Id (A)	Employees/ Supervisor_Id (B)	Supervisor_Id

$T_k$

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 all the couples of  $\langle \text{Table}, \text{key} \rangle$  in buf do

take one couple  $\text{buf}_c$  at a time

if  $(\text{buf}_c.\text{Table} = T_k)$  then

if  $(\text{buf}_c.\text{key} \neq \text{Key}(T_{[i]}))$  and  $(\text{buf}_c \text{ not in InheritedKey}(T_{[i]}))$  then

InheritedKey( $T_{[i]}$ ) +=  $\text{buf}_c$

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

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

$\text{buf}.\text{Table} += T_l$

$\text{buf}.\text{key} += K_{k_l}$

Table

Key

N/A

--	--

Employees/ Employee_Id (A)	Employees/ Supervisor_Id (B)	Employees/ Employee_Id (A)	Employee_Id
Employees/ Supervisor_Id (B)	Employees/ Employee_Id (A)	Employees/ Supervisor_Id (B)	Supervisor_Id

Join Path List

$T_{[i]}$

$T_k$

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 all the couples of  $\langle \text{Table}, \text{key} \rangle$  in buf do

take one couple buf<sub>c</sub> at a time

if (buf<sub>c</sub>.Table =  $T_k$ ) then

if (buf<sub>c</sub>.Key != Key( $T_{[i]}$ )) and (buf<sub>c</sub> not in InheritedKey( $T_{[i]}$ )) then

InheritedKey( $T_{[i]}$ ) += buf<sub>c</sub>

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 +=  $T_l$

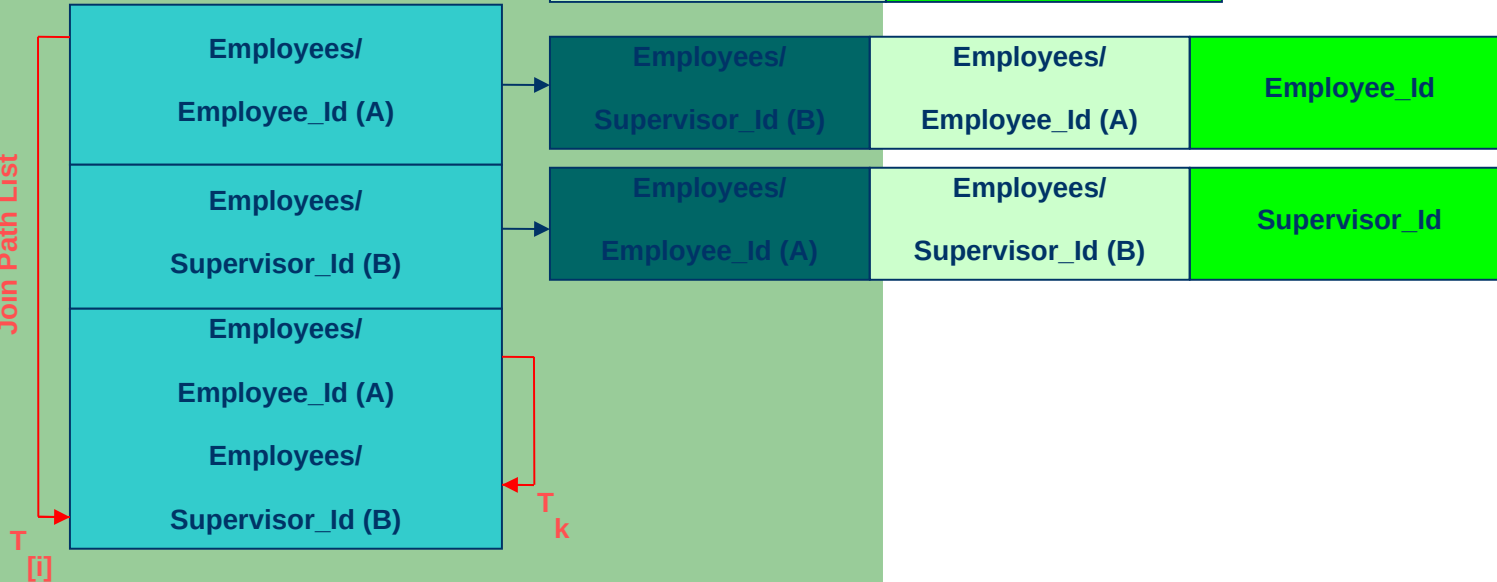
buf.key +=  $K_{k_l}$

Table

Key

--	--

Join Path List



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  $\langle \text{Table}, \text{key} \rangle$  in buf do

take one couple  $\text{buf}_c$  at a time

if  $(\text{buf}_c.\text{Table} = T_k)$  then

if  $(\text{buf}_c.\text{key} \neq \text{Key}(T_{[i]}))$  and  $(\text{buf}_c$  not in  $\text{InheritedKey}(T_{[i]})$ ) then

$\text{InheritedKey}(T_{[i]}) += \text{buf}_c$

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

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

$\text{buf}.\text{Table} += T_l$

$\text{buf}.\text{key} += K_{k_l}$

Table

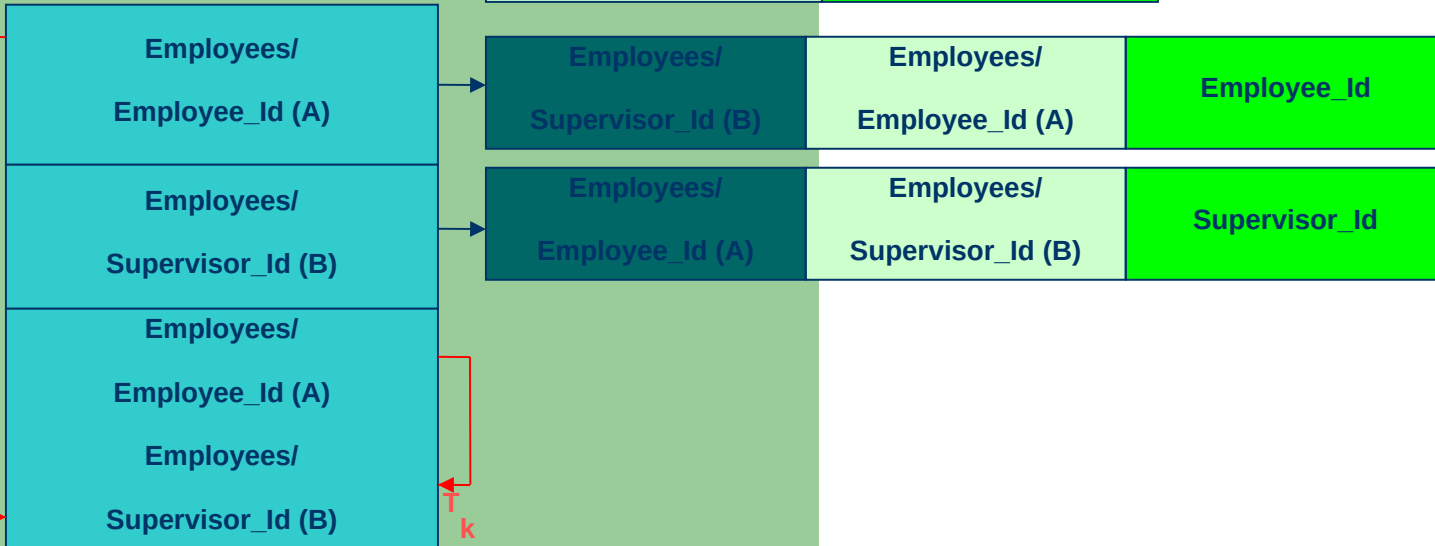
Key

N/A

Join Path List

$T_{[i]}$

$T_k$



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 all the couples of  $\langle \text{Table}, \text{key} \rangle$  in buf do

take one couple  $\text{buf}_c$  at a time

if  $(\text{buf}_c.\text{Table} = T_k)$  then

if  $(\text{buf}_c.\text{key} \neq \text{Key}(T_{[i]}))$  and  $(\text{buf}_c \text{ not in InheritedKey}(T_{[i]}))$  then

InheritedKey( $T_{[i]}$ ) +=  $\text{buf}_c$

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

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

$\text{buf}.\text{Table} += T_{[i]}$

$\text{buf}.\text{key} += K_{[i]}$

Table

Key

--	--

Employees/ Supervisor_Id (B)	Employees/ Employee_Id (A)	Employee_Id
Employees/ Employee_Id (A)	Employees/ Supervisor_Id (B)	Supervisor_Id

Join Path List

Employees/ Employee_Id (A)
Employees/ Supervisor_Id (B)
Employees/ Employee_Id (A) Employees/ Supervisor_Id (B)

$T_{[i]}$



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  $\langle \text{Table}, \text{key} \rangle$  in buf do

take one couple  $\text{buf}_c$  at a time

if  $(\text{buf}_c.\text{Table} = T_k)$  then

if  $(\text{buf}_c.\text{key} \neq \text{Key}(T_{[i]}))$  and  $(\text{buf}_c$  not in  $\text{InheritedKey}(T_{[i]})$ ) then

$\text{InheritedKey}(T_{[i]}) += \text{buf}_c$

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

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

$\text{buf}.\text{Table} += T_l$

$\text{buf}.\text{key} += K_{k_l}$

Table

Key

--	--

Employees/ Employee_Id (A)	Employees/ Supervisor_Id (B)	Employees/ Employee_Id (A)	Employee_Id
Employees/ Supervisor_Id (B)	Employees/ Employee_Id (A)	Employees/ Supervisor_Id (B)	Supervisor_Id

N/A

Join Path List

$T_{[i]}$

Employees/ Employee_Id (A)
Employees/ Supervisor_Id (B)
Employees/ Employee_Id (A) Employees/ Supervisor_Id (B)

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  $\langle \text{Table}, \text{key} \rangle$  in buf do

take one couple  $\text{buf}_c$  at a time

if  $(\text{buf}_c.\text{Table} = T_k)$  then

if  $(\text{buf}_c.\text{key} \neq \text{Key}(T_{[i]}))$  and  $(\text{buf}_c$  not in  $\text{InheritedKey}(T_{[i]})$ ) then

$\text{InheritedKey}(T_{[i]}) += \text{buf}_c$

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

if  $T_i$  is the table from which comes  $\text{Key}(T_{[i]})$  and  $k_i$  is the respective key then

$\text{buf}.\text{Table} += T_i$

$\text{buf}.\text{key} += K_i$

Table

Key

--	--

Employees/ Employee_Id (A)	Employees/ Supervisor_Id (B)	Employees/ Employee_Id (A)	Employee_Id
Employees/ Supervisor_Id (B)	Employees/ Employee_Id (A)	Employees/ Supervisor_Id (B)	Supervisor_Id

Join Path List

$T_{[i]}$

Employees/ Employee_Id (A)
Employees/ Supervisor_Id (B)
Employees/ Employee_Id (A)
Employees/ Supervisor_Id (B)

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  $\langle \text{Table}, \text{key} \rangle$  in buf do

take one couple  $\text{buf}_c$  at a time

if  $(\text{buf}_c.\text{Table} = T_k)$  then

if  $(\text{buf}_c.\text{key} \neq \text{Key}(T_{[i]}))$  and  $(\text{buf}_c$  not in  $\text{InheritedKey}(T_{[i]})$ ) then

$\text{InheritedKey}(T_{[i]}) += \text{buf}_c$

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

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

$\text{buf}.\text{Table} += T_l$

$\text{buf}.\text{key} += K_{k_l}$

Table

Key

--	--

Employees/ Employee_Id (A)	Employees/ Supervisor_Id (B)	Employees/ Employee_Id (A)	Employee_Id
Employees/ Supervisor_Id (B)	Employees/ Employee_Id (A)	Employees/ Supervisor_Id (B)	Supervisor_Id

Join Path List

$T_{[i]}$

Employees/ Employee_Id (A)
Employees/ Supervisor_Id (B)
Employees/ Employee_Id (A)
Employees/ Supervisor_Id (B)

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  $\langle \text{Table}, \text{key} \rangle$  in buf do

take one couple  $\text{buf}_c$  at a time

if ( $\text{buf}_c.\text{Table} = T_k$ ) then

if ( $\text{buf}_c.\text{Key} \neq \text{Key}(T_{[i]})$ ) and ( $\text{buf}_c$  not in  $\text{InheritedKey}(T_{[i]})$ ) then

$\text{InheritedKey}(T_{[i]}) += \text{buf}_c$

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

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

$\text{buf}.\text{Table} += T_l$

$\text{buf}.\text{key} += K_{k_l}$

Table

Key

--	--

Employees/ Employee_Id (A)	Employees/ Supervisor_Id (B)	Employees/ Employee_Id (A)	Employee_Id
Employees/ Supervisor_Id (B)	Employees/ Employee_Id (A)	Employees/ Supervisor_Id (B)	Supervisor_Id

Join Path List

$T_{[i]}$

$T_{[i]}$

$k_l$

Employees/ Employee_Id (A)
Employees/ Supervisor_Id (B)
Employees/ Employee_Id (A)
Employees/ Supervisor_Id (B)

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  $\langle \text{Table}, \text{key} \rangle$  in buf do

take one couple  $\text{buf}_c$  at a time

if  $(\text{buf}_c.\text{Table} = T_k)$  then

if  $(\text{buf}_c.\text{key} \neq \text{Key}(T_{[i]}))$  and  $(\text{buf}_c$  not in  $\text{InheritedKey}(T_{[i]})$ ) then

$\text{InheritedKey}(T_{[i]}) += \text{buf}_c$

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

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

$\text{buf}.\text{Table} += T_l$

$\text{buf}.\text{key} += K_{k_l}$

Table

Key

N/A

--	--

Employees/  
Employee\_Id (A)

Employees/  
Supervisor\_Id (B)

Employees/  
Employee\_Id (A)

Employee\_Id

Employees/  
Supervisor\_Id (B)

Employees/  
Employee\_Id (A)

Employees/  
Supervisor\_Id (B)

Supervisor\_Id

Employees/  
Employee\_Id (A)

Employees/  
Supervisor\_Id (B)

Join Path List

$T_{[i]}$

$T_k$

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 all the couples of  $\langle \text{Table}, \text{key} \rangle$  in buf do

take one couple  $\text{buf}_c$  at a time

if ( $\text{buf}_c.\text{Table} = T_k$ ) then

if ( $\text{buf}_c \neq \text{Key}(T_{[i]})$ ) and ( $\text{buf}_c$  not in  $\text{InheritedKey}(T_{[i]})$ ) then

$\text{InheritedKey}(T_{[i]}) += \text{buf}_c$

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

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

$\text{buf}.\text{Table} += T_l$

$\text{buf}.\text{key} += K_l$

Table

Key

--	--

Employees/ Employee_Id (A)	Employees/ Supervisor_Id (B)	Employees/ Employee_Id (A)	Employee_Id
Employees/ Supervisor_Id (B)	Employees/ Employee_Id (A)	Employees/ Supervisor_Id (B)	Supervisor_Id

Join Path List

$T_{[i]}$

Employees/ Employee_Id (A)
Employees/ Supervisor_Id (B)
Employees/ Employee_Id (A)
Employees/ Supervisor_Id (B)

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  $\langle \text{Table}, \text{key} \rangle$  in buf do

take one couple  $\text{buf}_c$  at a time

if  $(\text{buf}_c.\text{Table} = T_k)$  then

if  $(\text{buf}_c.\text{key} \neq \text{Key}(T_{[i]}))$  and  $(\text{buf}_c$  not in  $\text{InheritedKey}(T_{[i]})$ ) then

$\text{InheritedKey}(T_{[i]}) += \text{buf}_c$

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

if  $T_i$  is the table from which comes  $\text{Key}(T_{[i]})$  and  $k_i$  is the respective key then

$\text{buf}.\text{Table} += T_i$

$\text{buf}.\text{key} += K_i$

Table

Key

--	--

$T_{[i]}$

Join Path List

Employees/ Employee_Id (A)
Employees/ Supervisor_Id (B)
Employees/ Employee_Id (A) Employees/ Supervisor_Id (B)

Employees/ Supervisor_Id (B)	Employees/ Employee_Id (A)	Employee_Id
Employees/ Employee_Id (A)	Employees/ Supervisor_Id (B)	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  $\langle \text{Table}, \text{key} \rangle$  in buf do

take one couple  $\text{buf}_c$  at a time

if  $(\text{buf}_c.\text{Table} = T_k)$  then

if  $(\text{buf}_c.\text{key} \neq \text{Key}(T_{[i]}))$  and  $(\text{buf}_c$  not in  $\text{InheritedKey}(T_{[i]})$ ) then

$\text{InheritedKey}(T_{[i]}) += \text{buf}_c$

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

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

$\text{buf}.\text{Table} += T_l$

$\text{buf}.\text{key} += K_{k_l}$

Table

Key

--	--

$T_{[i]}$

Employees/  
Employee\_Id (A)

Employees/  
Supervisor\_Id (B)

Employees/  
Employee\_Id (A)  
Employees/  
Supervisor\_Id (B)

Employees/  
Supervisor\_Id (B)

Employees/  
Employee\_Id (A)

Employees/  
Employee\_Id (A)

Employees/  
Supervisor\_Id (B)

Employee\_Id

Supervisor\_Id

Join Path List



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  $\langle \text{Table}, \text{key} \rangle$  in buf do

take one couple  $\text{buf}_c$  at a time

if ( $\text{buf}_c.\text{Table} = T_k$ ) then

if ( $\text{buf}_c \neq \text{Key}(T_{[i]})$ ) and ( $\text{buf}_c$  not in  $\text{InheritedKey}(T_{[i]})$ ) then

$\text{InheritedKey}(T_{[i]}) += \text{buf}_c$

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

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

$\text{buf}.\text{Table} += T_l$

$\text{buf}.\text{key} += K_{k_l}$

Table

Key

--	--

$T_{[i]}$

Employees/  
Employee\_Id (A)

Employees/  
Supervisor\_Id (B)

Employees/  
Employee\_Id (A)  
Employees/  
Supervisor\_Id (B)

Employees/  
Supervisor\_Id (B)

Employees/  
Employee\_Id (A)

Employee\_Id

Employees/  
Employee\_Id (A)

Employees/  
Supervisor\_Id (B)

Supervisor\_Id

Join Path List

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  $\langle \text{Table}, \text{key} \rangle$  in buf do

take one couple  $\text{buf}_c$  at a time

if  $(\text{buf}_c.\text{Table} = T_k)$  then

if  $(\text{buf}_c.\text{key} \neq \text{Key}(T_{[i]}))$  and  $(\text{buf}_c \text{ not in InheritedKey}(T_{[i]}))$  then

$\text{InheritedKey}(T_{[i]}) += \text{buf}_c$

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

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

$\text{buf}.\text{Table} += T_l$

$\text{buf}.\text{key} += K_{k_l}$

Table

Key

N/A

--	--

$T_{[i]}$

Employees/  
Employee\_Id (A)

Employees/  
Supervisor\_Id (B)

Employees/  
Employee\_Id (A)  
Employees/  
Supervisor\_Id (B)

Employees/  
Supervisor\_Id (B)

Employees/  
Employee\_Id (A)

Employees/  
Employee\_Id (A)

Employees/  
Supervisor\_Id (B)

Employee\_Id

Supervisor\_Id

Join Path List

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  $\langle \text{Table}, \text{key} \rangle$  in buf do

take one couple  $\text{buf}_c$  at a time

if  $(\text{buf}_c.\text{Table} = T_k)$  then

if  $(\text{buf}_c.\text{key} \neq \text{Key}(T_{[i]}))$  and  $(\text{buf}_c$  not in  $\text{InheritedKey}(T_{[i]})$ ) then

$\text{InheritedKey}(T_{[i]}) += \text{buf}_c$

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

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

$\text{buf}.\text{Table} += T_l$

$\text{buf}.\text{key} += K_{k_l}$

Table

Key

--	--

Employees/ Supervisor_Id (B)	Employees/ Employee_Id (A)	Employee_Id
Employees/ Employee_Id (A)	Employees/ Supervisor_Id (B)	Supervisor_Id

Employees/ Employee_Id (A)
Employees/ Supervisor_Id (B)
Employees/ Employee_Id (A)
Employees/ Supervisor_Id (B)

$T_{[i]}$

Join Path List

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  $\langle \text{Table}, \text{key} \rangle$  in buf do

take one couple buf<sub>c</sub> at a time

if (buf<sub>c</sub>.Table =  $T_k$ ) then

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

InheritedKey( $T_{[i]}$ ) += buf<sub>c</sub>

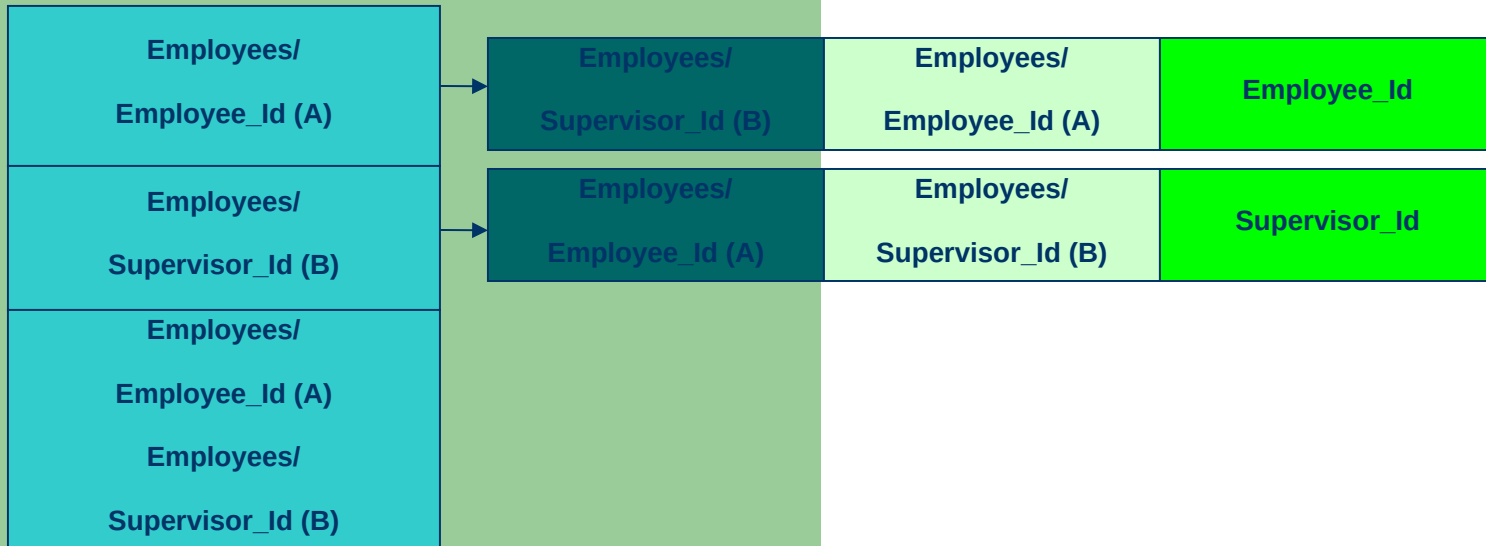
if  $T_{[i]}$  is not a base table then

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 +=  $K_{k_l}$

Join Path List



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

