データベースシステム 98－3
（１９９４．５．２７）

分散処理環境における生物分類樹データベースの統合化を目ざして

*北上 始、**舘野義男、**五條堀 孝

*広島市立大学、**国立遺伝学研究所

要旨

三大国際DNAデータバンク（日、米、欧）で利用されている生物分類樹データベースは、全て、コンピュータを用いた生物学の研究に有用な電子化辞書である。しかしながら、それらの生物分類樹データベースは、無矛盾に統合化されていない。もし、それらが統合化されれば、統合化された電子化辞書を用いて、生物研究結果の間の比較や既存の研究結果から将来の研究方向の選択などに利用することができる。また、形態学上のデータから計算された生物分類樹と分子データから推論された分子進化系統樹との間を比較するのにも有効である。ここでは、生物分類樹データベースの無矛盾な統合化のために、既存の生物分類樹データベースを無矛盾にする方法について述べられている。データベースの矛盾は、生物学が、近年、急速に発展していることにより生じている。即ち、この急速な発達により、生物分類樹の再構成が頻繁に行われてるが、現存のデータベースにはそれが十分に行われていないのである。この矛盾解消のために、近傍検索によるエラー診断、統合性制約による矛盾ノード抽出、エラー修正ツールなどについて述べられている。また、分散環境における矛盾抽出方法についても述べられている。以上は、全て、関係データベース管理システムを用いて実現されている。

TOWARD UNIFICATION OF TAXONOMY DATABASES IN A DISTRIBUTED COMPUTER ENVIRONMENT

Hajime Kitakami*, Yoshio Tateno** and Takashi Gojobori**

* Hiroshima City University
151-5 Ozuka, Numata-Chou, Asa-minami-Ku, Hiroshima-Shi 731-31, Japan
** National Institute of Genetics
1111 Yata, Mishima-Shi, Shizuoka-Ken 411, Japan

Abstract

All the taxonomy databases constructed with the DNA databases of the international DNA data banks are powerful electronic dictionaries which aid in biological research by computer. The taxonomy databases are, however not consistently unified with a relational format. If we can achieve consistent unification of the taxonomy databases, it will be useful in comparing many research results, and investigating future research directions from existent research results. In particular, it will be useful in comparing relationships between phylogenetic trees inferred from molecular data and those constructed from morphological data. The goal of the present study is to unify the existent taxonomy databases and eliminate inconsistencies (errors) that are present in them. Inconsistencies occur particularly in the restructuring of the existent taxonomy databases, since classification rules for constructing the taxonomy have rapidly changed with biological advancements. A repair system is needed to remove inconsistencies in each data bank and mismatches among data banks. This paper describes a new methodology for removing both inconsistencies and mismatches from the databases on a distributed computer environment. The methodology is implemented in a relational database management system, SYBASE.

1. Introduction

The increasing number of genome projects in the world have led to the further development of biological sciences and an advancement of the technology involved. In particular, the comparative and evolutionary studies of different genomes have become very important. This has promoted a restructuring of taxonomies constructed from various biological data. So far, such restructuring is carried out by a specialist who is interested in specific species or genes. Taxonomy data are usually very complicated because of the nature of its tree structure. It is thus important to make a taxonomy database which can be maintained in and searched through a computer system. The taxonomy database is a kind of electronic dictionary for searching automatically for any taxonomy information on the computer system. If the electronic dictionary can be used freely by biologists, it will contribute not only to genome research but also to other biological sciences. Above all, the evolutionary studies would gain tremendous benefits from a taxonomy database.

A typical taxonomy database is constructed at each of the international DNA data banks [1,2,3,4,5,6,7] which are EMBL (European Molecular Biology Laboratories) Data Library, GenBank-NCBI (National Center for Biotechnology Information)/GSDB-LANL (Genome Sequence Database constructed at Los Alamos National Laboratory) and DDBJ-NIG (DNA Data bank of Japan constructed at the National Institute of Genetics). These data banks shown in Figure 1 have been collaborating in many areas through mutual exchanges of data over the international computer network.

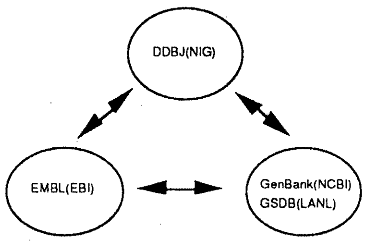


Figure 1. International DNA data banks

The aim of this paper is to present a method for producing a consistent taxonomic database that can integrate the taxonomy databases supplied by these data banks into a single unified taxonomy database. Before integration, it is necessary for each taxonomy database to be consistent within itself and any pair of taxonomy databases should also be consistent with each other. We propose a new methodology for finding inconsistencies and constructing a consistent taxonomy database as a step toward the unified taxonomy database in a distributed computer environment. The methodology is implemented in the relational database management system, SYBASE [8].

In summary, section 2 describes the basic concept of a recursive join which is a search engine of the implemented system. In section 3, we describe a new neighborhood search subsystem which is needed for clarification of error structures in the taxonomy tree. In section 4, we describe the definition of integrity constraints for detecting errors. In section 5, we describe the system configuration for implementing the new methodology. In section 6, we discuss related work for comparing the system with another system in constructing the unified taxonomy database. The final section is a summary of our research results.

2. Recursive Join

A taxonomy has a tree structure in which each taxonomic unit is connected with some immediately subordinate unit. Let us denote the former and latter units by "parent" node and "child" nodes, respectively. The relationship between a child node "X" and a parent node "Y" can be represented by a binary relation "(X,Y)", where "X" and "Y" are defined by the same domain "D".

Let $R=\{ (a,b) \mid a \in D, b \in D \}$ and $S=\{ (b,c) \mid b \in D, c \in D \}$ be two binary relations. The join of "R" and "S", denoted as $R \triangle S$, is defined as $\{ (a,c) \mid \exists b \ ((a,b) \in R, (b,c) \in S) \}$.

Let $R_0=\{ (a_0,b_0) \mid a_0 \in D,\ b_0 \in D \}$, $R_1=R_0$, $R_i=R_{i-1} \Delta R$. The recursive join of "R" with an initial relation "R_0", denoted as "[R]", is defined as $\{ \cup_{i>0} R_i \}$. Let us consider $R=\{ (a,b) \mid a \in D,\ b \in D \}$, where "a" is a child node of "b". The recursive join "[R]" computes all lineages from each node stored in the initial relation "R_0" to the root node. We call this "lineage processing". The other hand is the case of $R=\{ (a,b) \mid a \in D,\ b \in D \}$, where "a" is the parent node of "b". In this case, the recursive join "[R]" computes all progeny from each node stored in the initial relation "R_0" to the root node. We call it "posterity processing".

The recursive join [9] is represented by the following algorithm:

```
[R] = $R_0$ ;  R'= $R_0$ ;
while ( R' $\neq \emptyset$) do
    R' = R' $\Delta$ R  ;
    [R] = [R] $\cup$ R'  ;
end_while
```

In this paper, the previous binary relation is implemented as follows:

taxonomy(tx_id, tx_tl_id, tx_tx_idp, tx_nodename, _)

taxlevel(tl_id, tl_levname, _)

Each node of the taxonomy tree is stored in the "taxonomy" table and the level or ranking of the node in the taxonomy tree is stored in the "taxlevel" table. "tx_id" of the "taxonomy" table specifies the identifier for each node. "tx_tl_id" specifies the pointer to "tl_id" of the "taxlevel" table. "tx_tx_idp" specifies the pointer to a parent node in the "taxonomy" table. "tx_nodename" specifies the name of the node. "tl_id" of the "taxlevel" table specifies the identifier for each level name.

3. Neighborhood Search

In constructing a taxonomy database, it is very important to make it consistent. However, we do not have definitions clear enough to create inconsistency checking in the database design phase, because the biological information change due to rapid advancements in biology. After the design phase, we can incrementally clarify definitions related to the inconsistency checking in the database administration phase. The inconsistent (error) nodes stored in the "taxonomy" table can be incrementally detected in the administration phase. We can repair these inconsistencies by using the neighborhood search functions which visualize neighborhood nodes. The functions instruct us to conduct correct revisions in perspective. Three of the functions including the recursive join algorithm are as follows:

(1) lineage search function

This function searches for a path from a given node to the root node, which does not have any parent node in the tree structure. The path is a set of nodes found by searching in the "taxonomy" table. Appendix-1 shows an example of the program that was implemented in the control flow language of SYBASE called the "stored procedure". Our approach includes the object-oriented database concept [10], since the procedure is a kind of method for hiding the table structures of the "taxonomy" table. If we apply an artificial intelligence approach, we can obtain another kind of a program implemented in prolog [11]. This approach shown in Appendix-2 is made up with a smaller program than that is the previous approach. If we need more complex and higher processing to search and integrate taxonomy databases in the future, the artificial intelligence approach would be preferable in implementing more efficient processing.

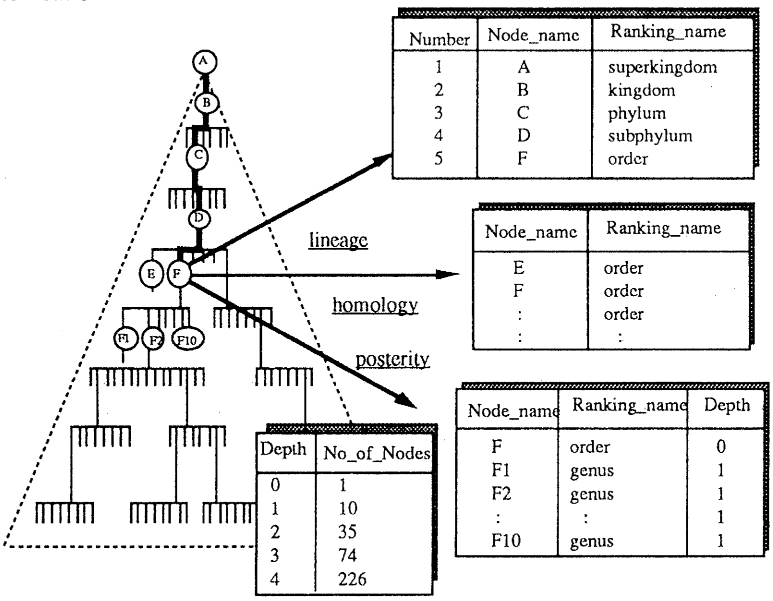
(2) posterity search function

We would like to make this function search for all paths from a given node to its leaf nodes, which do not have any child node in the tree structure. The number of the nodes found by the posterity processing is generally so large that the system can not visualize in any single window system at the same time. Thus we make this function visualize only in a given node and its child node. If we repeatedly use this function, we can find all paths from a given node to the leaf nodes in the tree structure.

In addition, we implemented an option to compute statistics which show the number of nodes for each level, in searching all progeny.

(3) homology search function

This function searches for nodes of the same level as a given node in the tree structure. If we search for a parent node of a given node, we can look for all children nodes of the parent node in the tree structure. This search visualizes homologous nodes for a given node. The homologous nodes include the given node and are in the same classification .



Number	Node_name	Ranking_name
1	A	superkingdom
2	B	kingdom
3	C	phylum
4	D	subphylum
5	F	order

Node_name	Ranking_name
E	order
F	order
:	order
:	:

Node_name	Ranking_name	Depth
F	order	0
F1	genus	1
F2	genus	1
:	:	1
F10	genus	1

Depth	No_of_Nodes
0	1
1	10
2	35
3	74
4	226

Figure 2. Neighborhood search

Figure 2 shows a representation of the neighborhood search. If we use the three functions, we can see a partial tree structure which spans both up-and-down and left-and-right nodes for a given node. We can define the neighborhood area inferred from any node using the neighborhood search functions. The neighborhood area can cover several nodes visualized by use of the functions. If we change the given node to an other node in the area, we can find the new area defined by the change. If we repeatedly apply the neighborhood search, we can move to any area in the original tree structure.

In addition, we also have a search function which looks for the accession numbers of the data entry to be connected with the taxonomy databases.

4. Integrity Constraints

It is important to maintain high quality in a database, including both facts (data) and inference rules. This enables us to construct a database system to carry out consistency checking using integrity constraints as defined by experts, with automatic revision of the database to eliminate inconsistencies [12].

The set of facts should be manually revised by biological experts, since the taxonomy database only has a set of facts without inference rules. The existent taxonomy database of DDBJ includes both syntax and semantic errors. The syntax errors can be easily found, if we can clarify structural inconsistencies which destroy the tree structure. The semantic errors can be found, if we can clarify the correct and invariable contents for the database with the advancement of biology. The difficulty lies in that a single data bank can not define the correct contents of the taxonomy database by itself and there are no perfect taxonomy dictionaries which reflect world-wide advancements in biology.

We are involved in an international collaboration aimed at defining the correct contents of the taxonomy database. This provides a chance to construct a consistent taxonomy database using the system.

Let us define two kinds of integrity constraints for detecting errors and inconsistencies. One is named the structural integrity constraint,which searches for any abnormal partial trees and nodes. The other is the cooperative integrity constraint, which searches for inconsistent nodes among the taxonomy databases of the international data banks. We show a logical sketch of the two kinds of integrity constrains in Appendix-3. The following are some examples of SQL expressions to implement them. Namely, if one of them does not have an empty solution, its integrity constraint is true and is inconsistent with the database. If one of them has an empty solution, the integrity constraint is false and is consistent with the database.

(1) Structural Integrity Constraints

The integrity constraints can detect (a) undefined node names, (b) data duplication, (c) invalid pointers to parent nodes, so that they can detect abnormal partial trees and nodes in the existent taxonomy database.

Undefined Node Name

We should avoid node names represented by blanks or NULL. This situation can be shown using the following query:

```
select  tx_nodename
from    taxonomy
where   tx_nodename in ( " ", NULL )
```

Data Duplication

If a user stores an input node without knowledge of the existence of the same node, data duplication occurs. We must avoid duplication with the exception of "sp.", which means unknown species and represents an unknown node name. We can show the situation using the following query:

```
select   tx_nodename
from     taxonomy
where    tx_nodename ≠ "sp." --------- exception
group by tx_nodename
having   count( tx_nodename ) > 1
```

Invalid Pointers to Parent Nodes

The normal tree structure has invalid (undefined) pointers for only the parent node of the root node and valid pointers for the parent node of another node. An abnormal tree structure including errors, has an invalid pointer for the parent node of nodes other than the root node. We can show the situation using the following two queries:

```
select  tx_nodename
from    taxonomy, taxlevel
where   tx_tl_id = tl_id
and     tl_levname not in ( "kingdom","super_kingdom" )
and     tx_tx_idp = NULL

select  tx_nodename
from    taxonomy x
where   x.tx_tx_idp ≠ NULL
and     not exists( select  tx_id  from  taxonomy
                            where  tx_id = x.tx_tx_idp )
```

(2) Cooperative Integrity Constraint

There are sometimes incorrect spellings, reverse turns for up-and-down relationships between parent and child nodes, and unfinished restructuring (for example, splitting and merging) in the tree structure. We can detect these errors through comparison with the taxonomy databases of the international DNA data banks over international computer networks. Error checking for the DDBJ-taxonomy database is conducted using this method, where it checks whether a pair of child nodes have the same name but one of the parent nodes has a different name. If the database is inconsistent, the following query does not have an empty solution:

```
select  x.tx_nodename
from   taxonomy x, other_taxonomy y
where  x.tx_nodename = y.tx_nodename
and    x.tx_tx_idp     ≠ y.tx_tx_idp,
```

where the "taxonomy" table is one of the tables stored in the DDBJ-taxonomy database and the "other_taxonomy" table is one of the tables stored in the EMBL/GenBank/GSDB-taxonomy database. If the query has one or more solutions, it means that the DDBJ-taxonomy database is inconsistent and error nodes are shown on a display.

Table 1. Computer environments

Databank Name \ Category	Database System	Computer System
DDBJ (NIG)	Sybase	CRAYSMP-42
GenBank (NCBI)	Sybase	Sun690
GSDB (LANL)	Sybase	Sun2000
EMBL (EBI)	ORACLE	Micro VAX 3100/80

5. System Configuration

Figure 3 shows the system configuration for building a consistent taxonomy database. The neighborhood search and database maintenance subsystems are developed by defining stored procedures implemented in both SQL and Control Flow Language (CFL) of SYBASE. The DDBJ-taxonomy database has integrity constraints which are defined by the stored procedure, since it is necessary to construct a consistent database. We need other taxonomy databases to define the cooperative integrity constraint. Other taxonomy databases are acquired automatically from EMBL, GenBank and GSDB over international computer networks. Table 1 shows the computer environments of each international DNA data bank.

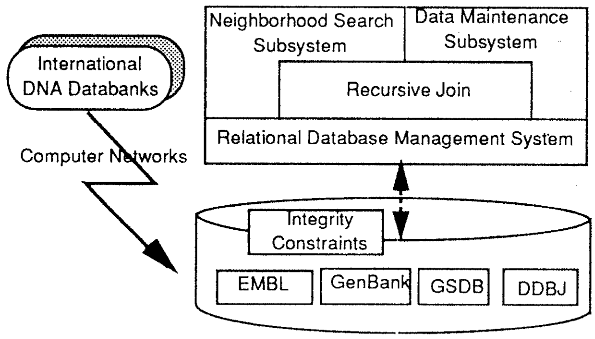

International DNA Databanks
Computer Networks
Neighborhood Search Subsystem
Data Maintenance Subsystem
Recursive Join
Relational Database Management System
Integrity Constraints
EMBL
GenBank
GSDB
DDBJ

Figure 3. System configuration

Figure 4 shows the building process of the taxonomy database. After detecting error nodes using the integrity constraints, we can analyze the error structures for each error node to correctly revise all the error nodes. The structure is clarified by the neighborhood search functions which provide lineage, posterity and homology search processing for the tree structure. After the analysis is completed, we can revise the taxonomy database using data maintenance functions, which provide allocation of new nodes, linkage between two nodes, merging among identical nodes and garbage collection for unused nodes. If we can finish the revision for one error node, we can repeatedly do the same processing for other error nodes.

In addition, we implemented both a write (or save) function to disk and an undo (or restore) function to execute error recovery for data revision. Both of them are implemented in CFL and the transaction management function of SYBASE.

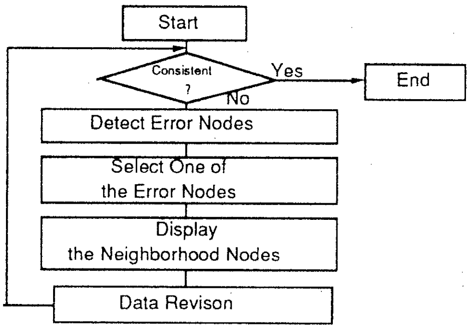


Figure 4. Building process for the taxonomy database

6. Related Work

TaxMan [13] and TAXSON [14] produced by NCBI are building and search systems for the taxonomy database, respectively. AWB (Annotators workbench) [3,4,5] created at LANL and IDEA (Interactive Data Entry and Annotation) [1,2] at EBI are, respectively, building systems for not only the DNA database but also the taxonomy database. AWB and IDEA have a mutual connection between the DNA and taxonomy databases.

The taxonomy database managed by TaxMan is specified in ASN.1 (Abstract Syntax Notation 1) and is stored in a flat-file system. The flat-file system is inadequate for managing the taxonomy database, since we need both flexibility and high performance to manage the taxonomy database. TaxMan has the same neighborhood search functions as our system, but does not have an automatic error detection function using the integrity constraint concept or a powerful merging function among the identical nodes. Moreover, TaxMan does not have a posterity search function with computation of statistics, either. The taxonomy database managed by TAXSON is stored in the relational database system, SYBASE. The TAXSON database is loaded from the TaxMan database; this allows users to use the power of the relational model for integrating the data with other systems and the power of the tree-structured model for maintaining the data itself.

Both AWB and IDEA are stored in the relational database system,but neither has the useful neighborhood search function. They have a function which searches the taxonomy database from the DNA database but another function to do it in the reverse.

7. Conclusions

We presented a new methodology for implementation of a system to build consistency in the taxonomy database. We focused on the existent taxonomy databases and presented methodologies designed for error detection and revision. The error detection mechanisms are represented by the structural and cooperative integrity constraints specified as stored procedures. The procedures are implemented in both SQL and CFL. Moreover, we proposed neighborhood search functions using the recursive join technique. The neighborhood search function provides a method for revising error nodes detected by these constraints in perspective. The error revisions are carried out using the data maintenance subsystem.

The system has been successfully used to repair the DDBJ-taxonomy database by DDBJ-staff since early autumn 1993.

We are planning to automatically integrate, the processes from error detection to the error revision using a graphic interface. The implemented system has recursive join processing as a search engine. Since the taxonomy database is increasing in size in an explosive manner, it is important to realize high performance for the recursive join. Parallel processing [15] is one possible solution for high performance.

Acknowledgments

We wish to thank Drs. James Ostell ,Tim Clark and Scott Federhen of NCBI for helpful discussion. We also thank Ms. Maria Mcleod of LANL for comments regarding taxonomy revisions. Thanks also to Drs. Rainer Fuchs and David Hazledine of EMBL Data Library for their help in automatically obtaining the EMBL-taxonomy database with the relational model. Finally, we also thank Ms. Mari Saito of DDBJ and all the other DDBJ-staff involved in developing the taxonomy database system.

References

[1] Patricia Kahn and Graham Cameron: EMBL Data Library, Methods in Enzymology, Vol. 183, 1990.
[2] Desmond G. Higgins, Rainer Fuchs, Peter J.Stoeher and Graham N.Cameron: The EMBL Data Library, Nucleic Acids Research, Vol.20, Oxford University Press, 1992.
[3] Christian Burks, Michael J. Cinkosky, Paul Gilna, et al: GenBank: Current Status and Future Directions, Methods in Enzymology, Vol. 183, 1990.
[4] Michael J. Cinkosky, James W. Fickett, Paul Gilna, and Christian Burks: Electronic Data Publishing and GenBank, Science, Vol. 252, 1991.
[5] Christian Burks, Michael J. Cinkosky,William M. Fischer, Paul Gilna, Jamie E.-D.Hayden, Gifford M. Keen, Michael Kelly, David Kristofferson and Julie Lawrence: GenBank, Nucleic Acids Research, Vol.20, Oxford University Press, 1992.
[6] Research News,Managing the Genome Data Deluge, Science,Vol.262, No.22,1993.
[7] Hajime Kitakami, Yukiko Yamazaki, Kazuho Ikeo, Yoshihiro Ugawa, Tadasu Shini, Naruya Saitou, Takashi Gojobori, and Yoshio Tateno: Building and Search System for a Large-scale DNA Database, Proc. One-Day Colloquim "Molecular Bioinformatics", The Institution of Electrical Engineers (IEE) Press, London, 1994.
[8] SYBASE Transact-SQL User's Guide, Sybase Inc., May 1989.
[9] Francois Bancilhon and Raghu Ramakrishnan: An Amateur's Introduction to the Recursive Query Processing Strategies, Proc. of the ACM SIGMOD '86, Washington D.C., pp.16-52, 1986.
[10] Prabhat K. Andleigh and Michael R. Gretzinger: Distributed Object-Oriented Data-Systems Design,Prentice Hall,Inc.,1992.
[11] Prolog by BIM(BIM_Prolog) Reference Manual, BIM (Belgium), 1990.
[12] Hajime Kitakami, Susumu Kunifuji, Taizo Miyachi, and Koich Furukawa: A Methodology for Implementation of a Knowledge Acquisition System, Proceedings of the IEEE 1984 International Symposium on Logic Programming, IEEE Computer Society,pp.131-142, 1984.
[13] Scott Federhen : TaxMan User's Mannual,to be available at NCBI's ftp site ("ncbi.nlm.nih.gov"),1994.
[14] Tim Clark, Cynthia Chung, and X. Yu: The NCBI Taxonomy Database, NCBI Preliminary Report, 1993.
[15] Yanchung Zang, Xiaofang Zhou, and Maria Orlowska: On Horizontal Fragmentation and Computation of Transitive Closures, Proc. of the International Symposium on Next Generation Database Systems and Their Applications, Kyushu University, pp.49-55, Fukuoka, Japan, 1993.

Appendix-1. An Example of Programming for a Lineage Search Implemented in both SQL and CFL

```
create procedure lineage @nodename char(80) as

declare @tuples int
declare @cnt    int

select  @cnt=1
create table tempdb..lineage(
            tx_id char(16),tx_idp char(16),
        level tinyint ,level_name char(32),node_name char(80))
create table tempdb..temp_table(
        tx_id  char(16),tx_idp  char(16),
        level tinyint ,level_name char(32),node_name char(80))
create table tempdb..work_table(
            tx_id char(16),tx_idp char(16),
        level tinyint ,level_name char(32),node_name char(80))

insert tempdb..temp_table
            select tx_id,tx_tx_idp,@cnt,tl_levname,tx_nodename
        from     taxonomy,taxlevel
        where    tx_tl_id=tl_id
        and   tx_nodename=@nodename
insert tempdb..work_table
        select * from tempdb..temp_table
insert tempdb..lineage
        select * from tempdb..temp_table
select @tuples=count(*) from tempdb..temp_table
delete tempdb..temp_table

while @tuples!=0
begin
      select @cnt=@cnt+1
      insert tempdb..temp_table
                select   x.acc_num,y.tx_id,y.tx_tx_idp,
                         @cnt,tl_levname,tx_nodename
                from     tempdb..work_table x,taxonomy y,taxlevel
                where     tx_tl_id=tl_id
                and     y.tx_id=x.tx_idp

      delete tempdb..work_table

      insert tempdb..work_table
                select * from tempdb..temp_table
      insert tempdb..lineage
                select * from    tempdb..temp_table
      select @tuples=count(*)
      from   tempdb..temp_table

      delete tempdb..temp_table
end

select * from tempdb..lineage
drop table tempdb..lineage, tempdb..temp_table, tempdb..work_table
```

Appendix-2. An Example of Programming for a Lineage Search Implemented in BIM-Prolog

```
lineage(NODENAME,[[TX_ID,TX_TX_IDP,LEVELNAME,NODENAME]|Result]):-
        taxonomy(TX_ID,TX_TL_ID,TX_TX_IDP,NODENAME),
        taxlevel(TX_TL_ID,LEVELNAME),
        search(TX_TX_IDP,Result).

search(TX_ID,[[TX_ID,TX_TX_IDP,LEVELNAME,NODENAME]|Result]):-
        taxonomy(TX_ID,TX_TL_ID,TX_TX_IDP,NODENAME),
        taxlevel(TX_TL_ID,LEVELNAME),
        search(TX_TX_IDP,Result).
search(TX_ID,[]).

taxonomy(TX_ID,TX_TL_ID,TX_TX_IDP,NODENAME):-
   retrieve(db_taxonomy(TX_ID,TX_TL_ID,TX_TX_IDP,NODENAME,_,_,_,_,_,_,_,_,_)),!.
taxlevel(TX_TL_ID,LEVELNAME):-
   retrieve(db_taxlevel(TX_TL_ID,LEVELNAME,_,_)),!.
```

The "db_taxonomy" and "db_taxlevel" tables are two tables of the taxonomy database defined in section 2.

Appendix-3. Integrity Constraints

Let ΔB be { DB_1, DB_2, ..., DB_n }, where DB_i is one of the taxonomy databases of the international DNA data banks. Let "lineage(a)" be the lineage, "name(a)" be the name, "level(a)" be the level (or ranking), "parent_id(a)" be the parent identifier, and "id(a)" be the identifier of the node, "a". If one of the databases is consistent with the structural integrity constraints, it is called a self-consistent database. The cooperative integrity constraint is applied only to the self-consistent database in order to check more semantic errors. We can logically define the two kinds of integrity constraints as follows:

1. Structural Integrity Constraints

structural_inconsistent$_1$ <-
  name(node) = $\emptyset$,
  node $\in DB_i$.

structural_inconsistent$_2$ <-
  name(node$_p$) $\neq \emptyset$,
  name(node$_p$) = name(node$_q$),
  node$_p \in DB_i$, node$_q \in$ { DB_i - node$_p$}.

structural_inconsistent$_3$ <-
  parent_id(node$_p$) $\neq \emptyset$,
  parent_id(node$_p$) $\neq$ id(node$_q$),
  node$_p \in DB_i$, node$_q \in DB_i$.

structural_inconsistent$_4$ <-
  name(node$_p$) $\neq$ "root",
  parent_id(node$_p$) = $\emptyset$,
  node$_p \in DB_i$.

2. Cooperative Integrity Constraints

cooperative_inconsistent <-
  name(node$_i$) = name(node$_j$),
  lineage(node$_i$) $\neq$ lineage(node$_j$),
  node$_i \in DB_i$, node$_j \in DB_j$,
  $DB_i \neq DB_j$, $DB_i \in \Delta B$, $DB_j \in \Delta B$.