Generating Hypertext Views on Databases

Falquet, Gilles; Prince, Ian James; Guyot, Jacques

Generating Hypertext Views on Databases

Gilles Falquet, Jacques Guyot, Ian Prince

C.U.I - Université de Genève, 24, Rue Général Dufour
CH-1211 Genève, Suisse
Tél: +41 22 705 77 70 - Fax: +41 22 320 29 27
e-mail: {falquet, guyot, prince}@cui.unige.ch
http://cuiwww.unige.ch/db-research/hyperviews/

Resumé:

Cet article présente un langage et un sytème de construction de vue hypertexte (hypervue) surune base de données. Nous étudions tout d'abord différentes manières de représenter les objets d'un base de données sous forme de composants d'un hypertexte: représentation directe des tuples, représentation d'ensembles de tuples et représentation de tuples associés. Nous montrons ensuite comment ces approches sont intégrées dans un langage déclaratif de définition d'hypervues. Ce langage permet de spécifier comment le contenu des différents types de noeuds ainsi que les liens hypertextes sont formés à partir du contenu de la base de données. Nous donnons également des indications sur la conception de la structure hypertextuelle en fonction des relations sémantiques représentés dans le schéma de la base de données.

Le prototype de traducteur qui a été réalisé génère des hypervues pour le système Worldwide Web (W3) à partir d'une base relationnelle. Nous décrivons les principaux composants du système réalisé et montrons comment le processus de traduction peut rémédier à certains défauts du modèle W3 d'hypertexte.

Mots clés: bases de données, sémantique des données, vues, hypertextes, génération d'hypertextes

---

Abstract:

This paper presents a language and a system to construct a hypertextual view (a hyperview) of the content of a database. We first study different approaches to map database objects to hypertext components: tuple level mapping, tuple sets mapping, and associated tuples mapping. Then we present a declarative hyperview definition language which integrates these approaches. With this language one can specify for each node type and each link type how to construct it from the database contents. We also show how the semantic relationship represented in the database schema can be used to design the hypertext structure.

A prototype translator has been implemented to generate a Worldwide Web (W3) hyperview of a relational database, the main components of this generation system are presented. We also show how the translation process can overcome some shortcoming of the W3 hypertext model.

Keywords: databases, database semantics, views, hypertexts, hypertext generation.

---

1. Introduction

It is well acknowledged that accessing a database with traditional query languages, like SQL, is too complex for non specialist or casual users. This is particularly true when the intended answer requires information that is spread across several database tables (or classes). In such a situation the user must be able to select a correct logical access path within the database schema and to express this path as a query language expression. The former task may be particularly difficult when the database schema comprises hundreds of tables and thousands of attributes (often bearing misleading names).

The generally adopted solution consists in providing a set of views or access forms to the user. However forms enable users to perform queries quickly and safely, they limit the user's interaction with the database to a set of predefined actions. This is particularly annoying when the user needs information that exists somewhere in the database but none of the predefined forms has been devised to get it. This approach also raises problems related to information distribution over large heterogeneous networks. Since forms are in fact database client applications, each new user must obtain and run these applications on his local system.

Database browsing or navigation tools can remedy such problems by allowing the user to explore the database contents with a single navigation tool.

Several exploration tools have already been proposed for databases, in particular for object-oriented databases (see [SIGMOD 92] for papers on this topic) (see [Isakowitz & al] for a hypermedia methodology based on E-R model).

We chose to take another perspective that consists in presenting the database content as a hypertext. This hypertext can be actually stored in a hypertext system or virtual, in which case, nodes and links are constructed on demand through database queries. Different users with different operating environments can then access this hypertext with a simple hypertext browsing tool, provided it recognizes the common hypertext markup language. In addition, the generated hypertext nodes can be linked to other external hypertext nodes and thus participate in a global hypertext.

Our first prototype has been used to build a W3 [Berners-Lee 94] hypertext view of the Oracle dictionary [Bobrowski 92] which comprises more than one hundred tables.

In the next section we present some features of the relational data model on which the hypertext construction is based. Note that since we use a semantic point of view, our results are also applicable to object-oriented models. In section 3 we analyse several ways of mapping database entities to hypertext components. Section 4 presents a hypertext definition language intended to specify how to construct hypertext nodes and links from the database contents. The implementation technique using W3 and Oracle is shown in section 6. Section 7 contains a comparison with other approaches. The conclusion discusses the benefits of hypertext generation techniques for W3-based hypertexts.

2. Semantic Links in the Relational Model of Data

2.1 Relations, Attributes and Tuples

In the relational model of data [Codd 70] real world entities are represented by relation tuples. The structure of these tuples is given by relation schemes which are sets of attribute names together with their domain (or data type). A tuple of a relation R takes a value for each attribute of R's scheme, this value must belong to the attribute's domain.

Relations can be either explicitly stored in the database or derived (computed) from other relations. In current relational database management systems stored relations are usually called tables and derived relations are called views. In this paper we will stay with the term relation to designate either a table or a view. We will also use the term tuple and attribute while these are often called rows and columns in commercial DBMSs.

Throughout the paper we will base all our examples on the Oracle database dictionary, which is a database comprising approximately 120 tables.

2.2 Inter Relations References

One important characteristic of the relational model is that references between tuples are based on attribute values, as opposed to object-oriented or hypertext models which use immutable object identifiers. However, relations usually possess a primary key which is composed of one or more attributes the values of which uniquely identify each tuple of the relation. For instance the attribute username is a primary key of relation **`USERS(username, user_id, creation_date,...)`** since all users must have a different user name. Primary key values can be used in other relations to refer to a particular tuple of this relation. For instance, each tuple of the relation **`TABLES(table_name, owner, table_space,...)`** refers to a tuple of USERS through the value of its owner attribute. Since the same user may possess several tables, this attribute establishes a "many-to-one" relationship between TABLES and USERS.

References through primary key values, called foreign keys, create many-to-one relationships. In order to implement many-to-many relationships between the tuple of two relations R(r-key, …) and S(s-key, …), the relational model forces to define an "associative" relation T(r-key, s-key, …) which refers to both R and S.

Inter relation references are generally intended to implement different kinds of semantic relationships such as compound-component (part-of), generic-specific (is-a), or general binary relationships. We will see later how the hypertext generation may depend on the kind of semantic relationship considered.

3. Strategies for Mapping Database Contents to Hypertexts Components

Database and hypertext models are based on radically different information representation and processing paradigms [Conklin 87][Nanard Nanard 93]. Hence it is not possible to transform a database into an equivalent hypertext. However, if one focuses on information representation, not considering information retrieval and processing, one can use different strategies to map the contents of a database to hypertext components.

3.1. Direct Tuple Level Mapping

The most straightforward way to map database entities to a hypertext structure consists in mapping each relation tuple t to a hypertext node node(t). In this situation, (a subset of) the attribute values of the tuple form the content of the corresponding node. The key attributes' values provide the identity of the node.

Semantic relationships based on attribute values can give rise to links between the nodes corresponding to the related tuples. For instance, let *t* be a tuple of **`TABLES`** which has value **`"Joe"`** on the attribute owner and let *u* be the tuple of **`USERS`** such that *u*.**`username`** = **`"Joe"`**; this induces a hypertext link from the node(*t*) to node(*u*).

If the target hypertext model does not support reverse traversal of links one must also generate a link from node(u) to node(t).

This approach is simple but it raises two problems:

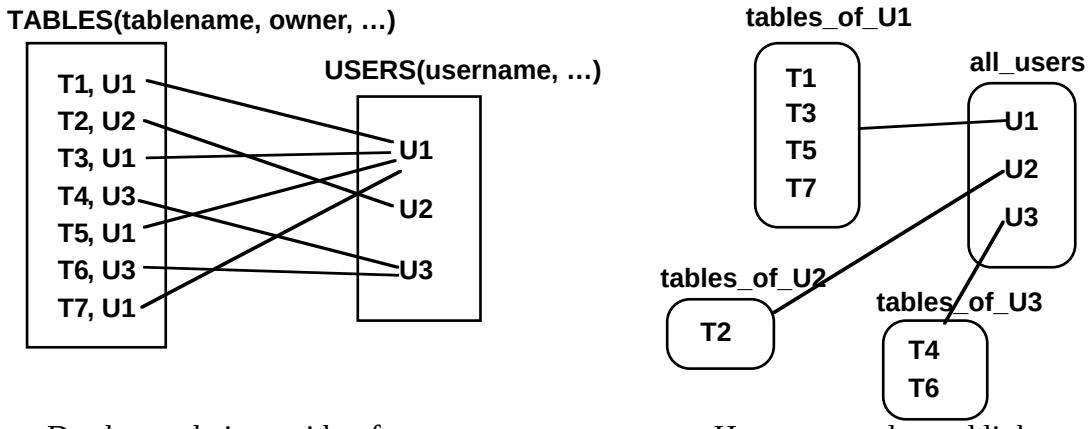
- it may generate a large amount of nodes and links, which can result in user's disorientation. For instance if user u owns 50 tables then there will be 50 links starting from u and leading to 50 nodes (one for each table).
- there is no way to see a set of data as a whole. For instance, to see the names of all the tables of user u it is necessary to navigate to each one of the 50 nodes linked to u.

However, it may be an acceptable solution for relations with many attributes and relatively few tuples.

3.2. Mapping Homogeneous Sets of Tuples to Nodes

In this approach each node is the representation of a set of tuples of a relation, nodes are structured as a set of items, each one of these being the representation of one selected tuple. This structure implies that there are now three kinds of links, namely: links between two nodes; links between an item and a node; and links between two items.

In general the content of a node will be made of the tuples of a relation which satisfy a given predicate. For instance, the content of a node **`tables_of_Joe`** could be a representation of all the tuples *t* of relation **`TABLES`** which satisfy *t*.**`owner`** = **`"Joe"`**, i.e., it will show all the tables owned by user Joe. Figure 1 below shows the hypertext structure one can create this way to represent the information content of two relations **`TABLES`** and **`USERS`** and their relationship through the foreign key owner



Database relations with references

Hypertext nodes and links

.

Figure 1. Mapping two connected relations to hypertext nodes and links

Compared to the previous approach, this one may greatly reduce the number of nodes and links necessary to represent the database content. In addition, it offers a more synthetic view of data. For instance, a single navigation step from a user item is sufficient to reach all the user's tables.

Another advantage of this approach lies in the possibility to directly represent one-to-many relationships, even in hypertext models (like W3) which do not support multi-headed links.

3.3 Mapping Sets of Related Tuples to Nodes

In the previous section we have shown how to define nodes by selecting sets of tuples from the same relation. We now consider nodes which are made of tuples coming from different relations. In this case the selection criteria is the existence of a relationship between the different tuples represented in a node. In other words, grouping occurs along the inter-relation reference axis instead of the within-relation axis.

This type of node construction is particularly useful to reconstruct complex entities because the relational data model does not provide constructs to represent complex hierarchic entities, i.e. entities which are composed of several other simple or complex entities. Such complex entities must be decomposed and stored as tuples of different relations.

For example, a user subschema is composed of tables, views, procedures, and triggers; a table is itself composed of a set of columns, etc. Such a complex object is represented by

- a tuple *u* of relation **`USERS(username, …)`**
- n_1 tuples of relation **`TABLES(table_name, owner, …)`** which refer to *u* through attribute owner

- m_1 tuples of relation **COLUMNS(column_name, table_name, owner, …)** each one linked to its table through attributes **(table_name, owner)**
- n_2 tuple of relation **VIEW(view_name, owner, …)** which refer to u through attribute owner
- etc.

In this case, the foreign key references (from **TABLES** to **USERS**, from **COLUMNS** to **TABLES**, etc.) represent a part-of semantic link between these relations.

It is natural to group all the tuples that represent a complex entity in a single hypertext node. The node's content can be organized hierarchically to reflect the structural composition of the complex object. Each tuple is mapped to an (sub)item whose level corresponds to its hierarchic level within the complex entity.

The main advantage of this mapping lies in its compact presentation of strongly related information, thus avoiding navigation operations among the different components of the complex entity.

In fact it is possible to construct such hierarchic nodes for any set of interrelated relations, even if they do not represent a "true" complex object, i.e. when the semantic links do not bear a "part-of" semantic. In this case the purpose of the construction is to reduce the number of links at the expense of larger nodes. This aspect will be further developed in section 5.

A similar mechanism is proposed in [Barsalou et al. 91] to view a relational database as a set of complex objects and in [Teuhola 93] to view object-oriented databases.

4. The Hyperview Definition Language

A hyperview specification consists of node types definitions which specify the name of the node type; the table or view from which the node's content is to be drawn; the presentation of the content; links to other nodes. A node definition takes the following form (the complete BNF syntax of the language is given in appendix):

```
node  <node-name>
         display
                  <field>,...
selected  by  <attribute-list>,...
ordered  by  <attribute-list>,...
from  <relation>
```

where each **<field>** is either an attribute or a hyperlink and ",..." denotes the possible repetition of the previous symbol.

Content of a node

The **from <relation>** clause determines the relation (which can be a base relation or a view) from which the node's content is constructed. The node's content is organized as a set of items, each one of them is the representation of a relation tuple.

4.1. Items Definition (attributes and formats)

The **display** clause specifies which attributes to display and the presentation format to use. Each item to display is composed of a number of fields which contain presentations commands and functions and the name of an attribute.

For example, let the table **sys.dba_users** contain the rows:

```
username     user_id     created      profile  ...
-----------------------------------------------
C2DDBA        28           11-OCT-95   DEFAULT  ...
C2DUSER      29           11-OCT-95   DEFAULT  ...
CAO           19           11-OCT-95   DEFAULT  ...
```

and let **users** be a node type defined as follows

```
node  users
   display  bold  (  username  )  break  ,
              " User_id: " bold ( User_id ) ,
              " Created: " bold ( created ) break,
             " Profile: "     bold ( profile ) ,
              " Default TS: " bold ( default_Tablespace )   ,
               " Temporary TS: " bold ( temporary_Tablespace )
newpara
     selected  by  ...
     order  by  ...
      from  "sys.dba_users"
```

A node of type **users** that contains all the table's rows will appear as shown on Figure 2.

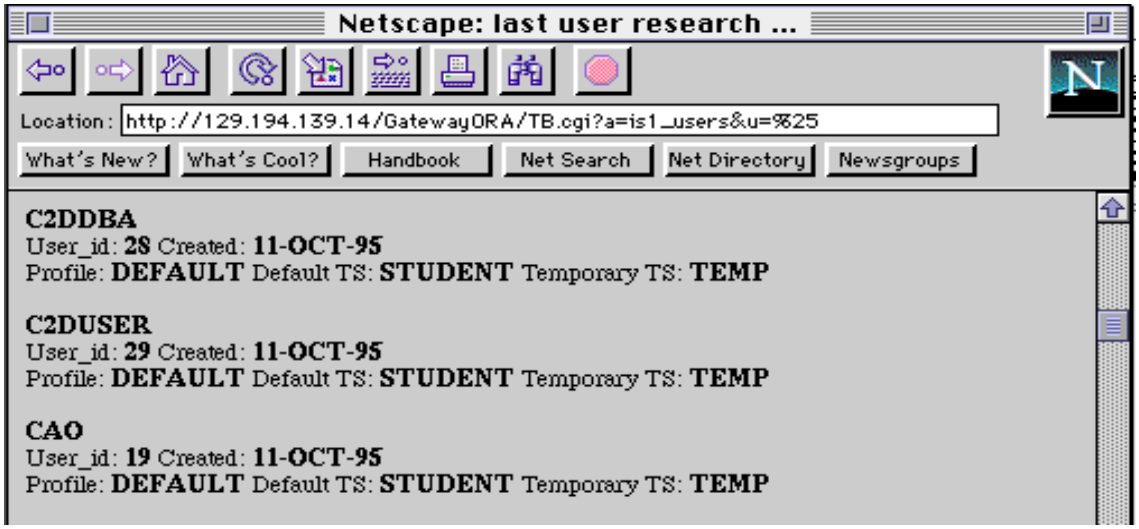


Figure 2. A node on relation **USERS**

4.1.2 Selection Criteria and Nodes Identities

The **selected by** clause specifies that each node of this type is composed of all the relation's tuple which have the same value for the given list of attributes. Thus the identity of a particular node of type T on relation R with selection attributes A_1, …, A_k is an expression of the form T[v_1, …, v_k] and its content is (a representation of) all the tuples t of R such that t.A_1 = v_1 and … and t.A_k = v_k.

For example, if node type **tables_of_user** is defined as

```
node  tables_of_user
…
selected  by  owner
from  "sys.dba_tables"
```

the node **tables_of_user["CAO"]** contains a representation of all the tuples of **sys.dba_tables** which have **owner** = **"CAO"**, i.e. a list of all the tables of user CAO.

It is clear that if A_1, …, A_k is a key of the relation R, each node T[v_1, …, v_k] contains a single item corresponding to the unique tuple of R with values v_1, …, v_k for the attributes A_1, …, A_k. This corresponds to the direct tuple level mapping of section 3.1.

4.2. Links

Links to other nodes are specified through the **`href`** statement. This statement must appear within the content (display) definition of an item. It comprises a node reference and an anchor content. The anchor content will be displayed as an active text. For instance, consider the node type **`users_2`** defined as

```
node  users_2
        display
                bold ( username ) break ,
                " User_id: " bold ( User_id ) ,
                " Created: " bold ( created ) break,
                " Profile: "   href profiles [profile]
                                  ( bold ( profile ) ) ,
                ...
                href  tables_of_user  [username]
                                  ( " [ Tables ]" )  ,
                ...
        selected by ...
        order by ...
        from "sys.dba_users"
```

An item of a **`users_2`** node will appear as follows (anchors are underscored)

C2DUSER
User_id: **29** Created: **11-OCT-95**
Profile: **DEFAULT** Default TS: **STUDENT** Temporary TS: **TEMP**
[Used Space] [Segments] [Tables] [Views] [Triggers] [Sources] [Roles]

Activating the link [tables] will display the node tables_of_user[“CAO”] which contains a description of all the tables owned by user CAO. Figure 3 shows a part of the link structure between **`user_2`** and **`tables_of_user`** nodes.

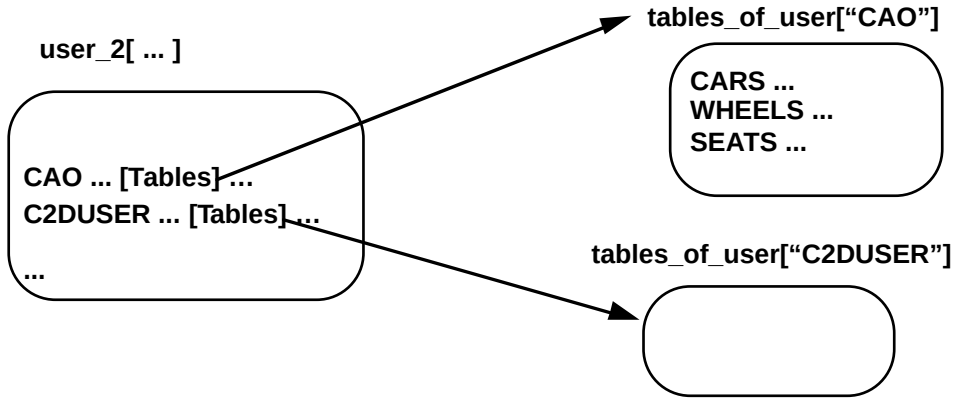


Figure 3. A node of type **`user_2`** linked to **`tables_of_users`** nodes

4.3. Nodes and links design

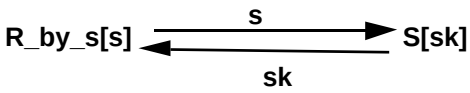
As mentioned earlier, the semantic relationships represented in the database must also be somehow represented in the hypertext structure.

4.3.1. Two-way foreign key references

Consider two relations R(rk, s, …) and S(sk, …) where rk and sk are primary keys and s is a foreign key of S in R. The following node definitions define a hypertext structure that enables to navigate from R to S (the functional direction) and from S to R (the one-to-many direction):

```
node  S
         display
                 sk, …
         selected  by  sk  from  S
node  R_by_s
         display
                 rk,
                 href  S  [s],  …
selected  by  s  from  R
```

This hypertext structure can be graphically represented as:



4.3.2. Binary relationships

We consider now the case of a binary many-to-many relationship between relations R(rk, …) and S(sk, …) implemented by a relation A(r, s, …). If we apply the above defined method for representing foreign key based links we end up with four nodes:

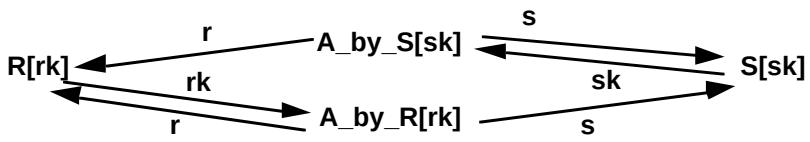
```
node  R  display  ...,  href  A_by_R  [rk],  ...  selected  by  rk  from  R

node  A_by_R
         display  ...,  href  S  [s],  href  R  [r]
         selected  by  rk

node  S  display  ...,  href  A_by_S  [sk],  ...  selected  by  sk  from  S

node  A_by_S
         display  ...,  href  R  [r],  href  S  [s]
         selected  by  sk
```

The graphical representation is:



This structure can be further improved in two ways

1. By adding a link href A_by_S[s] in A_by_R and href A_by_R[r] in A_by_S one creates a “short-cut” in the circuit R → A_by_R → S → A_by_S → R. Figure 4 shows such a hypertext structure generated for the many-to-many relationship between **USERS** and **ROLES**, Figure 5 shows two navigation steps in this hypertext.

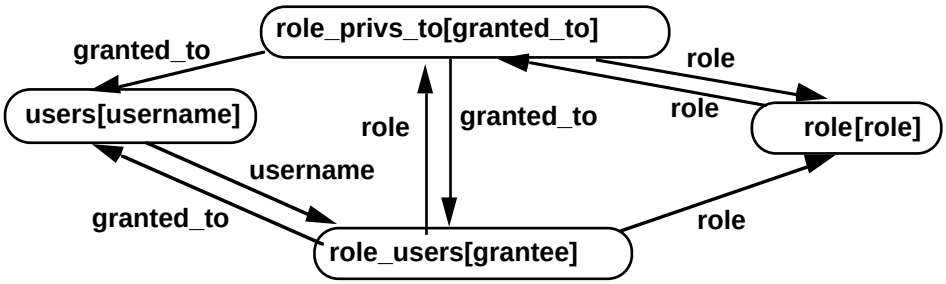


Figure 4. Hypertext structure generated from a binary many-to-many relatiohship

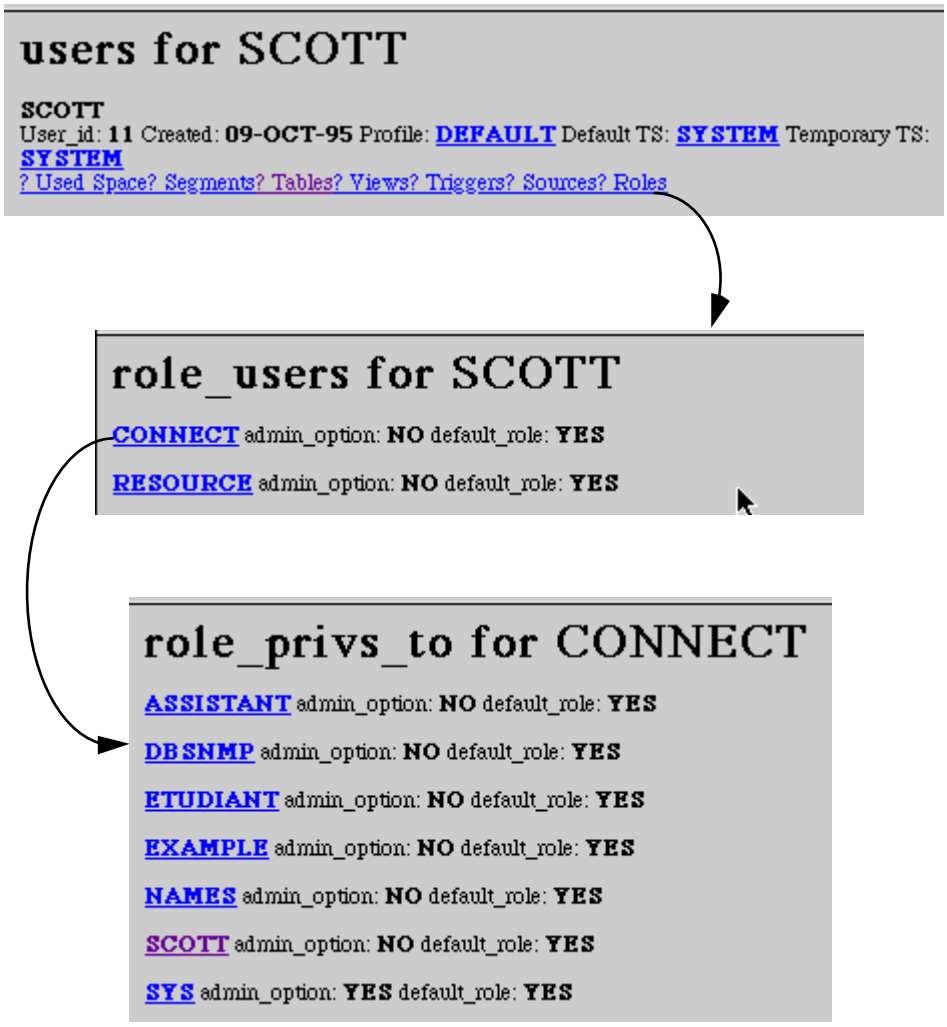


Figure 5. Navigation through a many-to-many relationship

2. By defining A_by_S (resp. A_by_R) on a join view A_R = A *r=rk R to include some or all of R's attributes in A_by_S. This suppresses one navigation step to get information about R's entities when coming from S.

5. Immediate Links

Immediate links plays two roles: a) they enable to construct hierarchical nodes to represent complex entities; b) they also provide an outlining mechanism that gives information about the content of a linked node without having to navigate to it.

5.1. Representing Complex Entities

While a normal link specification creates an active text that refers to another node, an immediate link specification adds the content of the referred node (or a part of it) to the content of the node being specified. In other word, each item of the referring node imports the content of the node it refers to. In fact, an immediate link creates sub-items within each item of the referencing node. The content of these sub-items is determined by the "display" part of the referenced node.

Example.

Let node type users_2 be redefined as

```
node  users_2
    display
         bold ( username ) break ,
         " User_id: " bold ( user_id ) ,
         " Created: " bold ( created ) ,
         " Profile: " immediate href profile_of_user [username]
                      ( profile) break ,
         ...
         immediate href tables_of_user [username] ( "? Tables ")
                  ...
    from "sys.dba_users"
```

The node **users_2["FALQUET"]**will display as shown on Figure 6

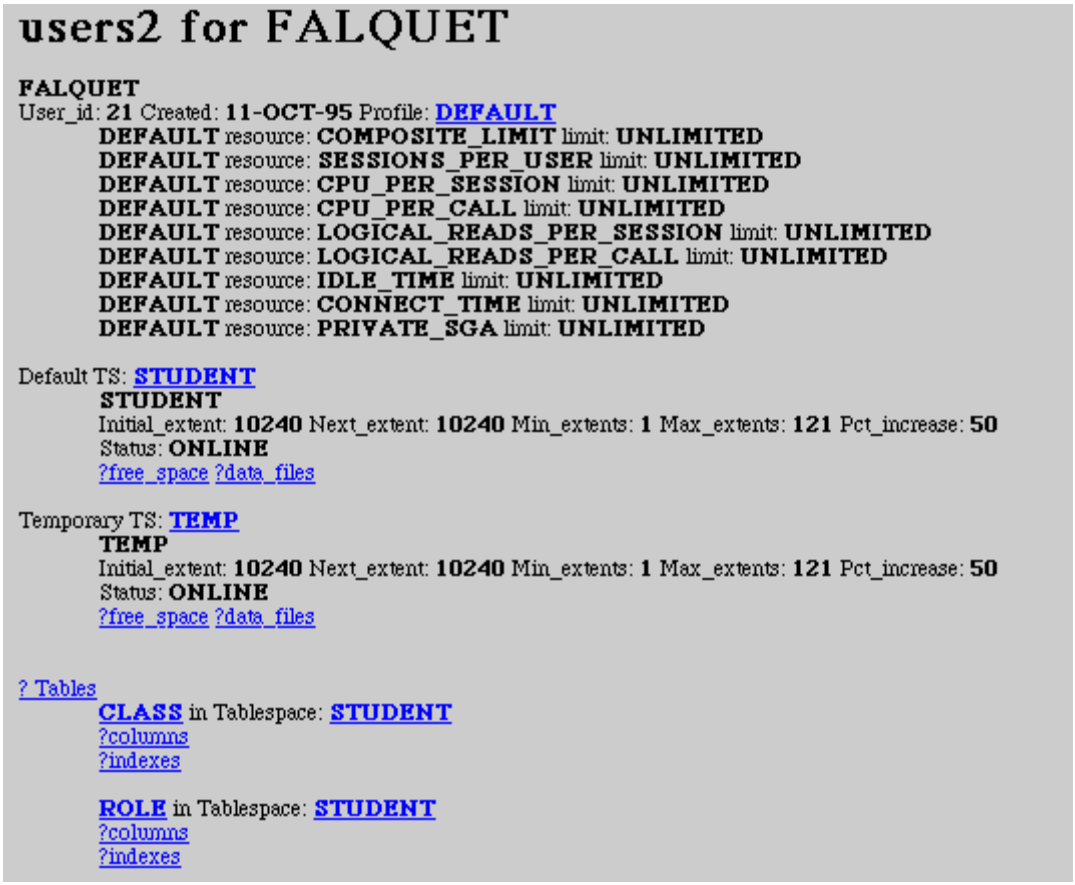


Figure 6. A node representing a complex entity

Note that the normal reference is still included to allow navigation to the referenced node.

If an immediately referenced node contains immediate references, these one will in turn generate sub-sub-items of the initial referring node, and so on recursively. This enables the creation of multi-level hierarchical nodes to represent complex entities stored in several database relations.

5.2. Representing Specialized Entities

In the relational model specialization hierarchies (or IS-A hierarchies) are often implemented by a set of relations having the same primary key [Smith Smith 77]. For instance, suppose that the specialization hierarchy Car IS-A Vehicle and Bicycle IS-A Vehicle is implemented by the three relations: **`VEHICLE(veh_no, type, … )CAR(veh_no, motor, …)`** and **`BICYCLE(veh_no, frame_size, …)`** The following node definition is sufficient to represent any vehicle whatever its type.

```
node  Vehicle
        display  veh_no,  type,
                 immediate  href  car  [veh_no]  (“  car  “),
                 immediate  href  bicycle  [veh_no]  (“  bicycle  “)
        ...
        from  VEHICLE
```

If the vehicle is a car the immediate link to **`car`** will include the car specific attributes of the vehicle found in the car node linked to this one. The link to **`bicycle`** will add nothing to the node since no bicycle node is linked to this vehicle. The opposite will happen for a vehicle which is a bicycle.

5.3. Outlining Linked Nodes

The other usage of immediate referencing is to give information about the content of linked nodes. This feature is of particular interest for very large and distributed hypertexts, like W3, where navigation operations may become time consuming due to network and servers load.

To avoid navigating to nodes which are not relevant to the user, immediate links add an outline of the referred nodes to the content of the referring node [Ichimura Matsushita 93]. This mechanism is based on the distance between the referring node and the referred one. By definition, a node is at distance 0 from itself and a node immediately referenced by a node at distance *i* is at distance *i* + 1. A visibility distance can be associated with each display field of a node type, indicating the maximum distance from which this field is visible through an immediate reference.

This mechanism also ensures that the content of a node cannot become infinite since for each type of node there is a distance from which no fields are visible.

Example

```
node  tables_of_user
    display
        (2) bold(table_name) ,
        (0) “in table space “ table_space ,
         (1) immediate href columns_of_table[owner, table_name]
            “ Columns”
…
selected by owner
from  "sys.dba_tables"

node  columns_of_table
    display
        (2) column_name ,
        (1) data_type ,
        (0) data_length
selected by owner, table_name
```

In a node of type **`users_2`** (defined in 5.1) the immediate reference to **`tables_of_user`** will cause the name of each table to be displayed but not its table space which is invisible at distance 1. The immediate reference to **`columns_of_table`** will also be visible but only the name of each column will be displayed since this node is at distance 2 from **`users_2`**.

5.4. Logical independence

An interesting property of immediate links is that the definition of a node depends only on the schema of its base relation, even if it has immediate references to other nodes. Thus, when the schema of a relation si modified (i.e. some attributes are added or deleted), only the nodes based on this relation must be adapted. Other nodes linked to theses nodes through immediate links need not be redefined. They will simply have a different content and appearance the next time they are accessed. Thus the consequences of a database schema update are automatically propagated in the hypertext without any global update of the hypertext structure.

6. Implementation Techniques

The implementation of this database viewing mechanism relies on two components: a node definition translator and a node server connected to a W3 HTTP server.

6.1. Nodes Translation to Views

The node definition translator takes as input a set of node types definitions and produces for each node type T a database view of the form

```
create  view  html_view_T  as
select  <display>  n_display  ,
          <selected  by> n_select ,
                  <ordered  by>  n_ordre
from  <table>
```

The translation of the “display” clause of the node specification produces the <display> part of the view which is a concatenation of attribute names and HTML tags,

The <selected by> and <ordered by> parts are the concatenation of the selection and ordering attributes respectively.

For instance the specification

```
node  users
     display  bold(username)  break,  “  user_id:  “  user_id
     selected  by  user_group
     ordered  by  username
from  sys.dba_tables
```

is translated into

```
create  view  http_users  as
select  '<b>'  ||  username  ||  '</b>'  ||  '  user_id:  '  ||  user_id
                  n_display  ,
         user_group  n_select ,
         username    n_order
from  sys.dba_tables
```

(Note: “||” is the concatenation operator)

Hyperlinks specified in the form “href node_type [expression] (anchor_text)” are translated into the text of an HTML anchor:

<A HREF=”http://server/hyperview.cgi? n=node_type&i=expression” > anchor_text </A>. This is a reference to a server script with node_type and expression as parameters.

When display fields have different viewing distances it is necessary to generate a view for each distance from 0 to the largest specified distance. Since only the definition of a view, not its content, is stored this does not consume storage space.

6.2. The Node Server

Our design strategy was to let the database server support all the retrieval and formatting work and to have the node server collect the formatted query results and transmit them to the client. The node server is a CGI (Common Gateway Interface) program connected to a standard W3 HTTP server on one side and to an Oracle database server on the other side.

The server script takes requests of the form n=node&i=value, transforms them to the SQL query “select n_display from html_view_node where n_select = value”, sends this query to the database server, collects the results, and sends the results back to the client. The script does no HTML formatting since everything is done by the SQL processor when executing the query corresponding to the view.

In the current version of the system the server is also responsible for the expansion of immediate references. If it detects immediate references within the query results it sends the corresponding queries to the database server and includes the result into the main query result before sending it to the client. This process is similar to macro expansion in programming languages. Depending on the depth of the expansion, which corresponds to the distance from the initial node, it chooses which view to query, when the maximal depth has been reached it stops the expansion process.

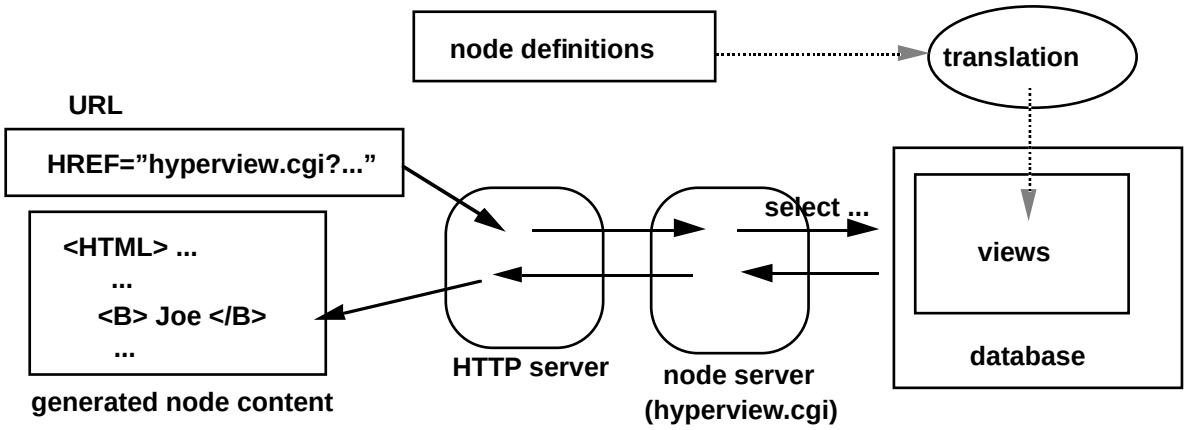


Figure 7. The node generation and retrieval process

7. Comparison with other Database to Hypertext Translators

The Decoux [Decoux] and WOW [WOW] gateways illustrate two possibilities to interface a database management system, namely, Oracle and W3.

With the Decoux CGI gateway a SQL query is directly embedded in a "page". The HTTP server redirects the "page" to the Decoux CGI which then sends the necessary query to Oracle. Oracle returns the query result to Decoux, which then inserts the result into the original "page" which is returned to the client as an HTML page via the HTTP server. In this case all the formatting is executed by the CGI thereby being independent of any particular DBMS. The Zelig system [Varela Hayes 94] takes a similar approach, it defines special tags to include queries in HTML documents.

Web-Oracle-Web (WOW) is a utility intended to develop CGI gateways for Web-servers with PL/SQL in an Oracle7 database. Wow procedures are executed within an Oracle database. Wow contains: a CGI program called wowstub; a PL/SQL package called wow; PL/SQL optional packages called HTP and HTF, these encapsulate HTML formatting; a standalone PL/SQL compiler with CGI enhancements.

Wowstub extracts the name of the procedure to execute, additional parameters are extracted and decoded before passing them to the PL/SQL procedure. HTF and HTP use the standard Oracle package DBMS_OUTPUT to write back to wowstub, which finally dumps the output to the HTTP server.

Most of the 'work' is executed in Oracle, thereby being largely independent of any particular HTTP server. On the other hand, each query requires a PL/SQL procedure.

O2Web [O2Web 95] is an HTTP server that takes as input an OSQL query, submits the query to an O2 database server [O2 91], adds HTML tags to the query's result and sends it to the client. O2Web creates a hypertext node (HTML document) for each database object; the content of the HTML document is a hierarchical representation of the object's value which is itself a hierarchical complex value. References to other objects are translated to HTML anchors which contain OSQL queries to get the referred object(s). The O2Web approach correspond to the direct mapping we have presented in section 3.1: one object is mapped to one node and one object reference to one link.

8. Conclusions

We have presented a hypertext specification language to build a hypertext from the content of a database. The approach we took was to build hypertext nodes from sets of database tuples and to

reconstruct complex entities. Unlike other approaches, ours is entirely declarative and based on the database's semantic links, thus it does not require any database or server programming.

Another important point is that the hypertext generation technique can remedy some shortcomings of the W3 hypertext model.

- W3 links are one-way, but for each semantic link of the database it is possible to create two W3 links, one in each direction, thus providing two-way navigation in the generated hypertext.
- W3 links are single-headed, i.e. they point to only one node. Since each generated node represents a set of database tuples, a single link to a node correspond to several database links, thus the link plays the role of a multi-headed link pointing to several nodes.
- Following a link may be slow in W3. Immediate links provide an outlining mechanism to minimize useless navigation operations.

The first prototype we have developed is based on virtual nodes that are constructed on demand by a node server. It has already been used to publish a hypertext view of different production databaes:

- the Oracle Dictionary (with links to a textual document describing each table)
- a database of Swiss federal polls
- a university database (curriculum, courses, schedules, enrolments)

Another generator creates all the nodes at once and store them as HTML documents. This provide a standalone hypertext representation of a database which is completely independent of the database management system.

There are still many directions to which this research can be extended. For instance, in the next version we plan to include global string search and hierarchic indices generation, as well as navigation maps generation. We are also working on a very efficient implementation of immediate links based on SQL stored functions. One can also consider adding interactive manipulation functions on the generated hypertext, such as creating or removing links. From a more theoretical point of view we plan to formalize database to hypertext mappings and study the properties of the generated hypetexts, for instance the equivalence between navigation paths and database queries.

References

[Barsalou et al. 91] T. Barsalou, N. Simabela, A. Keller, G. Wiederhold. "Updating Relational Databases through Object-Based Views". In Proc. ACM SIGMOD, Denver, 248-257, 1991.

[Berners-Lee 94] T. Berners-Lee, R. Cailliau, A. Luotonen, H. Frystyk Nielsen, A. Secret. "The World-Wide Web". Comm. of the ACM, Vol. 37, No. 8, 76-82, 1994..

[Bobrowski 92] S. Bobrowski. Oracle7 Server Concepts Manual, Oracle Corp., Redwood City, CA, 1992.

[Codd 70] E. F. Codd. "A Relational Model for Large Shared Data Banks". Comm. of the ACM, Vol. 13, 377-387, 1970.

[Conklin 87] J. Conklin. "Hypertext: An Introduction and Survey". IEEE Computer, Vol. 20, No. 9, 17-42, 1987.

[Decoux] G. Decoux. "The DECOUX Oracle WWW Gateway". http://dozer.us.oracle.com:8080/sdk10/decoux/ .

[Ichimura Matsushita 93] S. Ichimura, Y. Matsushita "Another Dimension to Hypermedia Access". In Proc. of the Hypertext'93 Conf., Seattle, 63-72, 1993.

[Isakowitz & al] T. Isakowitz, E. Stohr, P. Balasubramanian "RMM: A methodology for Structured Hypermedia Design". In CACM., 1995 number 8, 34-43

[Nanard Nanard 93] J. Nanard, M. Nanard. "Shoud Anchors Be Typed Too? An Experiment with MacWeb". In Proc. of the Hypertext'93 Conf., Seattle, 51-62, 1993.

[Teuhola 93] J. Teuhola. “Tabular Views on Object Databases”. Tech. Rep. R-93-11, University of Turku, Finland, 1993

[Varela Hayes 94] C. A. Varela, C. C. Hayes. “Zelig: Schema–Based Generation of Soft WWW Database”. In Proc. W3 Conf., 1994.

[WOW] “The WOW Gateway”. http://dozer.us.oracle.com:8080/sdk10/wow/

[SIGMOD 92] Special Issue: Advanced User Interfaces for Database Systems. SIGMOD Record, Vol. 21, No. 1, 1992.

[Smith Smith 77] J. M. Smith, D. C. P. Smith. “Database Abstractions: Generalization and Specialization”. ACM Trans. on Database Syst., Vol. 2, No. 2, 1977.

[O2 91] “The O2 System”. Comm. of the ACM, Vol. 34, No. 10, 1991

[O2Web 95] “O2 Web Presentation”, O2Technology, Versailles, France, 1995.

Appendix

BNF Syntax of the Node Definition Language

```
startrule  =  "define" {node_def} "end"  .
node_def  =  "node" nodeident display_part select_part order_part from_part  .
display_part  =  "display" field {"," field}.
field  =  [ level  ]  [ header  ] modified_sqlpart [ footer  ]  .
level  =  "level"  "0 .. 9"  .
header  = string  .
modified_sqlpart  =
              (  "bold" | "italic" | "underline" | …) "("  ( modified_sqlpart | string )  ")"
      |        [  "immediate"  ]  "href" nodeident "[" sqlpart_list "]"
                                        "("  ( modified_sqlpart | string )  ")"  )
      |        sqlpart  .
sqlpart  =            columnident
              |       (  "sqlexpr"  "(" string  ")"  )  .
sqlpart_list  = sqlpart {"," sqlpart}.
string  =  """ {"character"} """  .
footer  =  "break" | "newpara"  .
from_part  =  "from" fromlistsqlpart  .
fromlistsqlpart  = string {"," string}.
select_part  =  "selected"  "by" sqlpart_list  .
order_part  =  "order"  "by" sqlpart_list  .
columnident  = identifier  .
nodeident  = identifier  .
identifier  =  "a-z" {"a-z" | "0-9" | "." | "_" }.
```