## Introduction

A considerable amount of work has been done on data bases for molecular structure data [1, 5, 6, 7, 8, 12, 15, 18, 20], although most of this work provided for only limited molecular structures. However, in an earlier paper [8], a comprehensive relational data base structure, based on a two path approach, was proposed for holding chemical structure data. The proposed data base structure allowed retrieval of chemical structures that contained correct IUPAC carbon atom occurrence numbers, and for retrieval of chemical substructures, recursively, down to the final atomic level.

Since new chemical structures result from chemical reactions and processes, a molecular structure data base is not complete without an associated data base for chemical reactions and processes. Although some constructive work has been done on specific chemical process data bases, particularly in chemical engineering [2, 4, 16], there appear to be no reports in the literature on how to structure a comprehensive chemical process data base.

In this paper the fundamental solution to the problem of a relational structure for data base for chemical reaction and chemical process data is presented. Practical data bases can be built using this solution or using many possible minor variations on the theme of it. It can also be used as the basis for extended versions using an object oriented or logic data base system.

For the purposes of the paper a reaction in a chemical process that cannot be broken down into subprocesses; and a process is

a chemical process, involving one or more reactions, and not a physical process, such as a diffusion process for separation of compounds. Thus some processes are reactions and do not break down, whereas other processes are not single reactions and do break down.

**Design problem and acceptable solution criteria**

The goals were,

(a) A data base structure that can hold every conceivable chemical process, together with the breakdown, via multiple levels of subprocesses, to the level of chemical reactions. The breakdown must be consistent with how constituent subprocesses feed molecules into, and are fed molecules from, other subprocesses of the breakdown, that is, with connecting upstream and downstream processes.

(b) A data base structure that can be built with a commercially available data base system and accessed by a non procedural language, where little or no programming is required, since it is envisaged that the primary use of the data base would be by scientists and engineers who needed to retrieve, sometimes in the field in remote locations, data about an individual target process.

In dealing with the second goal, the problem could be pursued in terms of CODASYL, hierarchical, or relational data base systems, or in terms of the two main extensions to the relation approach, namely the object oriented approach and the AI or logic data base approach.

Since neither the CODASYL nor hierarchical approaches [5, 8] permit the use of a flexible non procedural manipulation language, these can be eliminated. There are major advantages to using a relational data base manipulated by SQL [5, 3, 14]. Conceptual files are constrained to be relations so that predicate calculus can be applied to them. The outstanding advantage is that even with complex searches of the data base, it is usually unnecessary to write a program in a procedural language, as with CODASYL and hierachical approaches [5]. It is necessary only to specify what kind of data is to be retrieved using the non procedural SQL [5, 14] based on predicate calculus.

In the object oriented approach, objects (such as an individual process and its immediate (child) subprocess, or a process and its immediate (parent) superprocess) are manipulated by the user via a fixed set of routines. The objects are usually defined on top of an underlying relational data base. However, the object oriented approach [8] has more to do with the development of reusable manipulation modules, for a data base that will be manipulated via defined classes embedded in the routines of some general software system, and not via a non procedual manipulation language. Since such a development was not consistent with the goals of the project, the object oriented approach was rejected.

With the other extension, the artificial intelligence or logic database approach [8], there is a relational data base (the extensive data base) and a set of rules or predicates called the intensive data base. The logic programming approach requires a relational data base as its foundation, with the use of the in-

tensive data base containing relevant predicates (functions with values true or false) as an additional layer of knowledge. An example might be the predicates HALOGEN (FLUORINE), HALOGEN (CHLORINE), and so on, which embodies the knowledge that fluorine, chlorine, and so on, are halogens. The system could then use these predicates, when asked for information about halogens, to look up fluorine, chlorine, and so on, in the extensive relational data base. A very wide variety of intensive data bases could be devised, depending on the need, to supplement the extensive relational data base that contains the process breakdown data.

It follows that a fundamental chemical process relational data base structure that could be used with SQL, could also, with minor modification, serve as the underlying relational data base in the object oriented approach, and as the extensive relational data base in the AI or logic programming approach. Thus the first step must be the solution to the problem of a relational data base structure for a comprehensive chemical process data base.

SQL can be used with the data base proposed in this paper, and examples of its use are included. SQL is the standard data base manipulation and retrieval language with such common relational data bases systems as DATABASE2 [10], ORACLE [11], and INGRES [17].

Given that a relational data base design is needed, addressing the first goal forms the core of this paper. The major part of problem is the need for a universal data base structure to handle the subdivision of processes into subproceses, and so on, when any given level of the breakdown can have a substructure that can be a process sequence, process cycle (such as the Krebs cycle),

process fan, process funnel, for example, or an arbitrarily complex structure. The solution presented in this paper is relatively simple and fundamental, and is guaranteed to handle every conceivable process breakdown.

**Conceptual-level versus physical-level data base design**

In the design process, it is a conceptual data base (formerly called a logical data base) that is being designed, as distinct from a physical data base. By a conceptual data base is meant a data base containing essential data about the physical reality being modelled, and not containing physical implementation data such as pointers and indexes [5, 8]. However, readers need to distinguish these concepts clearly. In the data base field, the "physical" level refers to the implemention level, whereas the conceptual level refers to the real world, which, in the sciences would be called the physical world, and thus easily misunderstood as the physical level. The conceptual-level data base design parameters must be entered at a terminal or workstation and stored, when using a relational system. Physical-level data base design parameters are entered separately and merely tune the database system for efficiency of storage and speed of manipulation of the data base. The physical level is not considered in this paper.

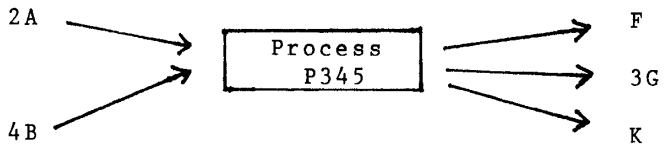



Figure 1

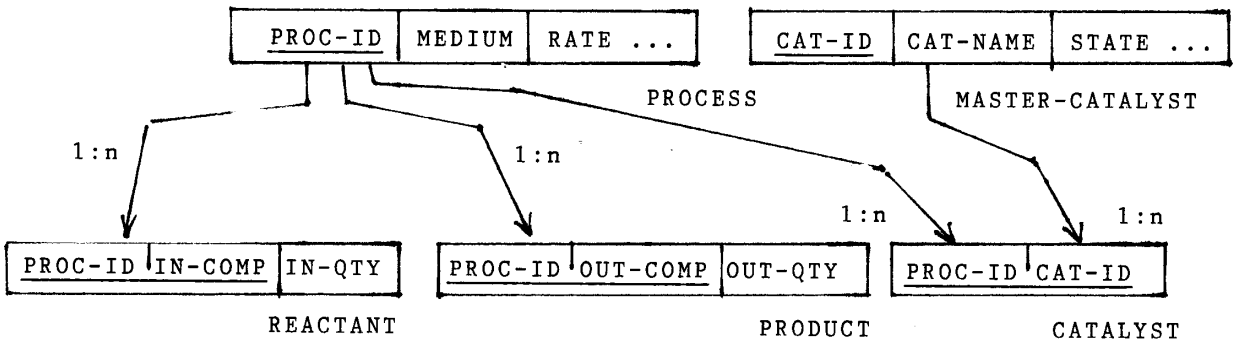



Figure 2

**Step-1 Design: Basic chemical reactions data base**

The proposed conceptual level data base structure is best understood by considering the data base in two steps. The first step simply involves a data base about reactions and processes, with no consideration of how a process can subdivide into sub-processes, and so on. Thus no distinction can be made initially between a reaction and a process, since the data base has no information on any further breakdown. Both have reactants and products, and physical properties, such a reaction rate data, and so on.

Because the number of reactants and processes varies from one reaction to another, a single relation for reaction/process data is not possible, since a relation, by definition, must have a fixed number of fields or attributes. In fact, using relational data base terminology, the dependency between a process (or reaction) and its reactants and products is multivalued [5, 8, 14], since for one process there is a set of reactants and a set of products (Figure 1). Because of this multivalued dependency there is only one reasonable way to place the reactant and product data in relations, as illustrated in Figure 2.

In Figure 2 there are five relations, where primary keys (unique record identifiers) are underscored. One of these, PROCESS(PROC-ID, MEDIUM, RATE, EQUILIB, ...) is straight forward. Each record or tuple describes a process (or reaction) in terms of an identifier (PROC-ID) and in terms of physical properties (such a gaseous or aqueous (MEDIUM), reaction rate rate (RATE), equilibrium constant (EQUILIB), and so on, depending on user needs). A process

identifier (PROC-ID) is necessary. This would have to be a numeric, alphabetic or alphanumeric code capable of serving the number of processes anticipated for the data base. (However, an international agreement, by which every known chemical process is assigned an agreed code, is preferable).

In the relation MASTER-CATALYST (CAT-ID, CAT-NAME, STATE, ...), each record holds data about a substance, in a particular form or state, that serves as a catalyst in one or more reactions. CAT-ID gives the identifier for the catalyst; for example C237 might identify a hot platinum wire, whereas C238 might identify particulate platinum. The other fields give descriptive data about the catalyst, such as CAT-NAME (its name) and STATE (solid, liquid, powder, hot wire, ...), and so on.

The relation CATALYST (PROC-ID, CAT-ID) has a record for every process that uses a catalyst. This relation is needed because some reactions require more than one catalyst, and becauseexactly the same catalyst may be used with different reactions. This means that the relationship between PROCESS and MASTER-CATALYST is many-to-many (n:m), that is, a process can use many catalysts and a catalyst can be used in many processes.

The other two relations in Figure 2 are REACTANT (PROC-ID, IN-COMP, IN-QTY) and PRODUCT (PROC-ID, OUT-COMP, OUT-QTY). A record in REACTANT identifies a process with PROC-ID, gives an input compound (IN-COMP) or reactant molecule of the process, and gives the quantity of the reactant moleculue (IN-QTY) required to balance the reaction equation for the process. Similarly, a record in PRODUCT identifies a process (PROC-ID), and gives an

output compound (OUT-COMP) or product molecule, and gives the quantity of the product molecule (OUT-QTY) required to balance the reaction equation. Thus, with the data base in Figure 2, the data for the reactions:

P001: 2 + 4B = F + 3G + K

P002: 4P + A = 3S + 4F

would be in the relations REACTANT and PRODUCT as follows:

| PROC-ID | IN-COMP | IN-QTY |
|---|---|---|
| P001 | A | 2 |
| P001 | B | 4 |
| P002 | P | 4 |
| P002 | A | 1 |

REACTANT

| PROC-ID | OUT-COMP | OUT-QTY |
|---|---|---|
| P001 | F | 1 |
| P001 | G | 3 |
| P001 | K | 1 |
| P002 | S | 3 |
| P002 | F | 4 |

PRODUCT

For the purposes of the paper we use upper case letters to identify chemical compounds. In practice, the identifiers used in a molecular structure data base, typically IUPAC names, would be used.

It should be understood that REACTANT essentially lists the input compounds to processes, with a single record for each input compound to a specific process. The primary key is a therefore a

composite of PROC-ID and IN-COMP. Similarly, PRODUCT lists the output compounds from a process, with a single record for each output compound from a specific process, with PROC-ID OUT-COMP as a composite primary key. This may seem counter-intuitive. A structure where each record listed all the input compounds, or all the output compounds, to a process is more appealing. But because such records would have varying numbers of fields, and are therefore not relations, they cannot be used.

There is a one-to-many (1:n) relationship [5] between PROCESS and REACTANT, faciliated by the field PROC-ID, since for one process there can be many reactants. There is also a 1:n relationship between PROCESS and PRODUCT, also based on PROC-ID, since for on process there can be many products.

Unskilled users who have learned elementary SQL will have no trouble using this data base, as the following examples show :

**Example 1. Find the products of each process that involves reactants A and B in aqueous solution, using the data base in Figure 2.**

```
SELECT PROC-ID, OUT-COMP FROM PRODUCT
WHERE PROC-ID IN
(SELECT PROC-ID FROM PROCESS
WHERE MEDIUM = 'AQUEOUS'
AND PROC-ID IN (SELECT PROC-ID FROM REACTANT
                WHERE IN-COMP = 'A')
AND PROC-ID IN (SELECT PROC-ID FROM REACTANT
```

```
WHERE IN-COMP = 'B'));
```

**Example 2 Find the reactants of reactions in which the enzyme hexokinase participates.**

```
SELECT PROC-ID, IN-COMP FROM REACTANT
WHERE PROC-ID IN
(SELECT PROC-ID FROM CATALYST
 WHERE CAT-ID IN (SELECT CAT-ID FROM MASTER-CATALYST
                  WHERE CAT-NAME = 'HEXOKINASE'));
```

It should therefore be clear that the data base structure in Figures 2 is quite practical, and could store the essential reactant/product/environment data for every conceivable chemical process. However, it will not store how a process breaks down, nor how the breakdown constituents connect in a flow network of processes. That is accomplished by the extension to the design in step-2.

**Step-2 Design: Data base structures for process composition**

The way in which a process can break down into subprocesses, and each subprocess into further subprocesses, can be infinitely varied, as readers familiar with biochemical processes can testify [2,19]. For a given process, some common breakdowns are:

1. Straight sequence. A processes breaks down into a straight sequence of subprocesses, in which at least some of the output molecules from one process serves as input molecules to the next process in sequence.

2. Cycle. The simplest cycle will have one process feeding molecules into the cycle and one being fed molecules by the cycle. Within the processes in the cycle, at least some of the output molecules from one process will be fed as input molecules to the next process in the cycle. With more complex cycles there can be more than one process feeding into the cycle and more than one being fed by the cycle.

3. Fan-out. A single process feeds output molecules into more than one sequence of processes.

4. Funnel. More than one process feeds output molecules into a single process.

The above breakdowns listed are not exhaustive. Many other breakdown structures, probably uncommon, are possible. Furthermore, there can be many levels of breakdown. Any of the processes in a breakdown can be a process that breaks down further, in any of the ways listed, and other ways besides.

The problem is how to devise a data base structure that will handle any of the wide variety of possible process breakdowns

that can occur. There appears to be two general approaches to the problem. One approach is to devise individual data base structures (and thus relations) that will match the individual process substructures, such as those listed above. This approach was extensively reseached and proved futile, mainly because no matter how many additional structures were used, it was apparently always possible to show that there was a possible process breakdown that would not fit any of the proposed structures; thus it was not possible to prove that any of the structures were universally applicable.

The alternative is to ignore the structure of the breakdowns and devise a data base structure that directly reflects the structure of the data, which is the path taken in this paper. The structure of chemical process data can be seen to involve two basic recursive relationships:

(a) A many-to-many (n:m) composition recursive relationship

This involves the breakdown or explosion of a process, no matter which, into its subprocesses, and the implosion of a given process out of its parent processes. The recursive relationship occurs because for a given type process, no matter which, the following must hold:

(1) There can be zero or more parent (or super-) processes, and for each parent process in turn zero or more parent processes, and so on. It is not the case that a given type of process or

reaction can have only one parent process; it can have many. For example, a common oxidation reaction could occur within many quite different processes.

(2) There can be zero or more subprocesses or child process, and for each child process, in turn, zero or more child processes, and so on.

Since a process is related to zero or more parent processes, and so on, and to zero or more child processes, and so on, it follows that process entities are many-to-many related to themselves, and are thus recursively n:m related. Call this (process) composition recursivity.

(b) A many-to-many (n:m) stream recursive relationship

This involves an implosion of a given process, no matter which, out of its upstream processes, and the explosion of a given process into its downstream processes. This recursive relationship occurs because for any process the following must hold:

(1) Its reactants may be generated as the products of processes that feed it (immediate upstream processes), with the reactants of the immediate upstream processes being generated as the products of further upstream processes, and so on.

(2) The products it generates may be the reactants of processes that it feeds (immediate downstream processes), with the pro-

ducts of the immediate downstream processes serving as the reactants of further downstream processes, and so on.

Since a process is related to zero or more upstream processes, and so on, and to zero or more downstream processes, and so on, it follows that process entities are many-to-many related to themselves, and are thus recursively n:m related. Call this the stream recursivity.

The existence of two distinct (composition and stream) recursive many-to-many relationships in chemical process data is a basic phemomenon, and its recognition is the key to structuring a chemical process data base, no matter what data base technology is employed. (As it happens, current relational data base technology with SQL is probably the best there is for handling n:m recursive relationships, although there is room for improvement.) It is also worth pointing out that the existence of two fundamental recursive relationships in chemical process data appears never to have been mentioned in the chemical and chemical engineering literature.

Note that the two distinct recursive relationships, while independent, involve the same entities, namely processes, so that, with general chemical process data, when considering the implications of one relationship, the other must not be forgotten. For example, suppose we are considering the upstream and downstream processes for process P. Now P might be a process within process X, that is a child of X, and we can give this particular instance of P the path name X.P. Furthermore, P might occur as Y.P within process Y, and as Z.P within process Z. It follows that each of X.P, Y.P

and Z.P could have a different set of upstream processes feeding into it, and a different set of downstream processes being fed by it, even though, in all three cases we are dealing with the same type of process P. The upstream/downstream implosions/explosions (that is, the implications of the stream recursivity) for a process depend on its parent process (that is, the implications of the composition recursivity).

**Recursive many-to-many relationships**

Because recursive n:m relationships are basic to a chemical process data base, a brief review is relevant. Consider two relations RA and RB, each related in a 1:n relationship with a third relation RR, as shown in Figure 3a. Each RA tuple or record describes a unique A-type entity identified by the primary key A (underscored) value, and each RB tuple describes a B-type entity identified by the primary key B. RA and RB have attributes that describe the A and B entities, but these are not relevant and are not shown. A tuple or record in RA shows an A entity paired or related to a B entity (Figure 3b). The other attributes in the record, such as Q, describe the relationship. For example A could be an engineer and B a project, so that Q could denote the time spent by an engineer on a project.

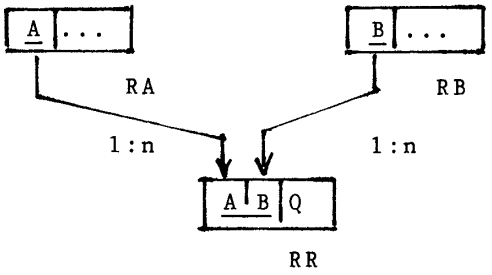



Figure 3a

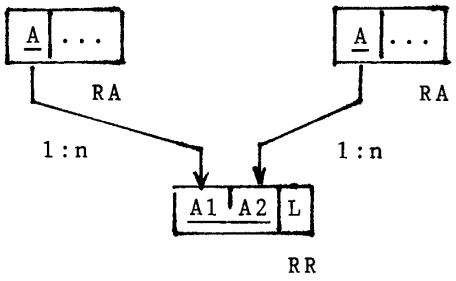



Figure 4a

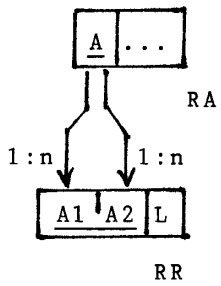



Figure 4b

| A ... |
|---|
| a1 ... |
| a3 ... |
| a4 ... |
| a7 ... |

RA

| B ... |
|---|
| b1 ... |
| b3 ... |
| b4 ... |
| b6 ... |

RB

| A | B | Q |
|---|---|---|
| a1 | b1 | 6 |
| a4 | b1 | 8 |
| a3 | b3 | 9 |
| a7 | b3 | 3 |
| a1 | b4 | 2 |
| a1 | b6 | 6 |
| a3 | b6 | 8 |
| a4 | b6 | 5 |

RR

Figure 3b

| | b1 | b3 | b4 | b6 |
|---|---|---|---|---|
| a1 | 6 | | 2 | 6 |
| a3 | | 9 | | 8 |
| a4 | 8 | | | 5 |
| a7 | | 3 | | |

Figure 3c

The attributes A and B from RA and RB do not occur in RR as primary keys, and thus do not have to have unique values, which is the case in RA and RB. Accordingly, RA is 1:n related to RR and RB is also 1:n related to RR, since there must necessarily be a 1:n relationship between two relations with a common field, where the

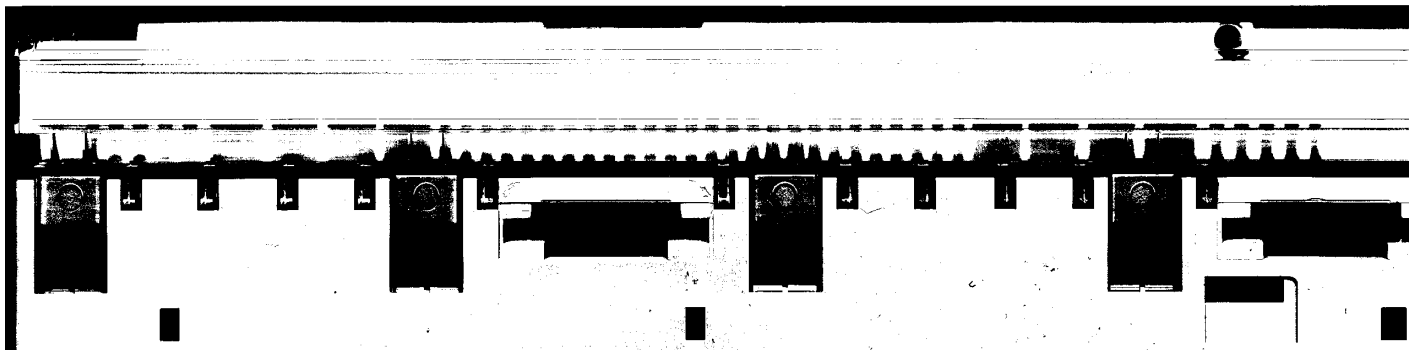
| A1 | A2 | L |
|---|---|---|
| a4 | a1 | 6 |
| a4 | a3 | 9 |
| a1 | a10 | 7 |
| a3 | a10 | 15 |
| a3 | a8 | 42 |
| a7 | a1 | 18 |
| a6 | a8 | 14 |
| a10 | a8 | 7 |
| a10 | a9 | 9 |

RR

Figure 4c

| | a1 | a3 | a4 | a5 | a6 | a7 | a8 | a10 |
|---|---|---|---|---|---|---|---|---|
| a1 | | | 4 | | | | | 7 |
| a3 | | | | | | | 42 | 15 |
| a4 | 6 | 9 | | | | | | |
| a5 | | | | | | | | |
| a6 | | | | | | | 14 | |
| a7 | 18 | | | | | | | |
| a8 | | | | | | | | |
| a10 | | | | 9 | | | 7 | |

Figure 4d



common field is a primary key in one relation and not in the other. However, since for an RA tuple there are many related RR tuples, but for one of these RR tuples only one related RB tuple, it follows that for one RA tuple there are many related RB tuples. Similarly, for one RB tuple there are many related RA tuples. Hence there is a many-to-many (n:m) relationship between RA and RB, which can be displayed as a matrix (Figure 3c). For example, from the matrix, and from RR, we can see that b3 is related to a3 and a7, and that a3 is related to b3 and b6.

The n:m relationship between RA and RB becomes recursive when RA and RB are the same, that is, RA = RB, as shown in Figure 4a. The relation RR shows how an entity from RA (identified by an A1 value) is related to an entity from RA (identifed by an A2 value). The A1 and A2 columns in in RR thus contain A values from RA (Figure 4c). If A entities are parts used in manufacturing assembly process, then, going in the direction from A1 values to A2 values, an RR tuple <a1 a10 7> could indicate that part type a1 contained part type a10 at location 7, which, from the relationship matrix in Figure 4d, also means that in turn part type a10 contained part types a5 and a8, and so on recursively, giving us a parts explosion; going in the direction from A2 to A1 the RR tuple <a1 a10 7> means that part type a10 is contained within part type a1, and that in turn (see relationship matrix) part type a1 is contained with part type a4, and so on, recursively, giving an implosion. For the sake of brevity an n:m recursive relationship is often depicted as in Figure 4b.

**Separate relations for composition and stream recursivity**

If we are concerned only with the breakdown composition of processes, and not with how one process feeds another, then the data base in Figure 5 containing the two relations PROCESS and COMPOSITION is necessary and sufficient for a comprehensive chemical process data base, since PROCESS and COMPOSITION embody the necessary composition recursivity. [The relations CATALYST AND MASTER-CATALYST have to be added, exactly as in Figure 2, however.]

The relation COMPOSITION requires a minimum a three fields. To see why, consider a data base for processes P3 and P8, which breakdown as in Figure 6a. A tuple of COMPOSITION gives a process SUPER-PROC-ID along with one of its subprocesses SUB-PROC-ID. Thus, since P3 breaks down into P4, P6 and P14, there must be tuples <P3, P4, 1>, <P3, P6, 1>, <P3, P4, 2> and <P3, P4, 1>. The final field OCCUR gives an occurrence number, to inform when a subprocess is occurring more than once at the same level in a breakdown, as in the case of P3 having two distinct occurrences of P4 (Figure 6a). The contents of COMPOSITION for the breakdowns is shown in Figure 6b.

Note that COMPOSITION will generate a number of distinct breakdown (or explosion) trees or hierarchies equal to the number of processes that are not subprocesses of any higher level process. It will also generate a number of implosions equal to the number of lowest level processes (reactions), showing what processes a given reaction occurs in. The implosion for process P1 is shown in Figure 6c.

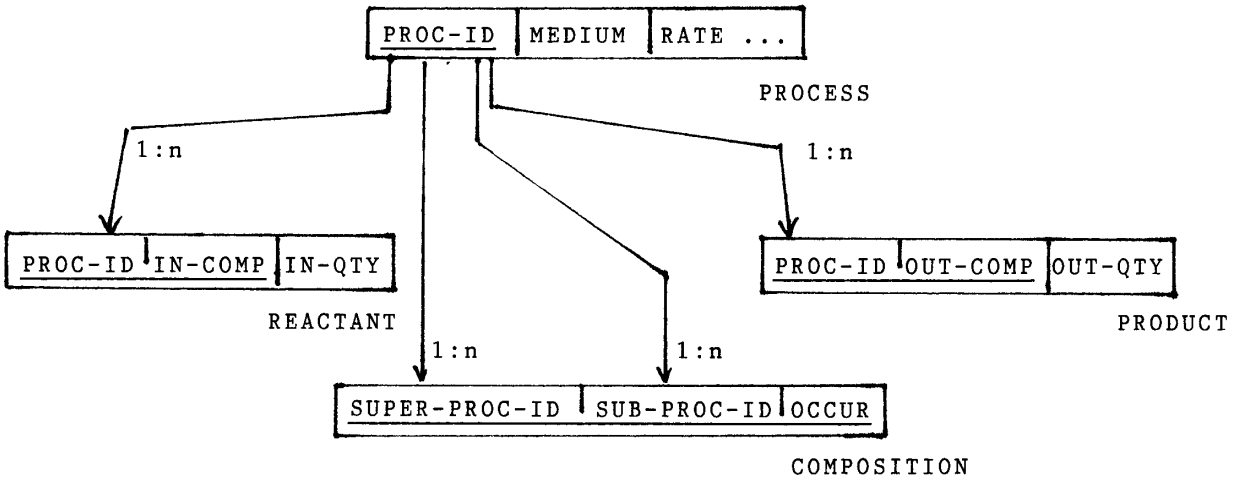



Figure 5

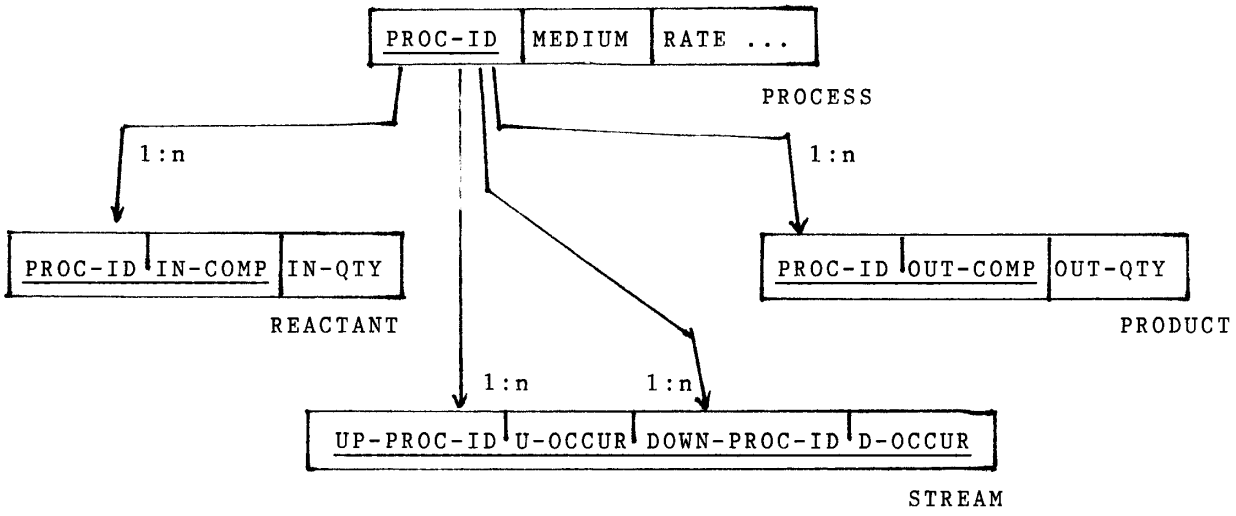



Figure 7

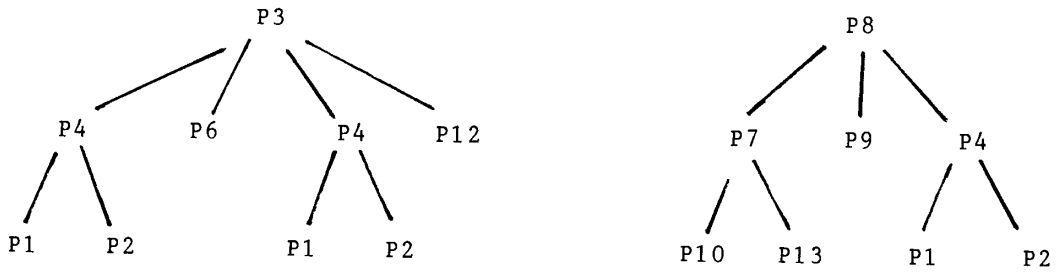



Figure 6a

| SUPER-PROC-ID | SUB-PROC-ID | $OC_{CUR}$ |
|---|---|---|
| P3 | P4 | 1 |
| P3 | P6 | 1 |
| P3 | P4 | 2 |
| P3 | P12 | 1 |
| P4 | P1 | 1 |
| P4 | P2 | 1 |
| P8 | P7 | 1 |
| P8 | P9 | 1 |
| P8 | P4 | 1 |
| P7 | P10 | 1 |
| P7 | P13 | 1 |

COMPOSITION

Figure 6b

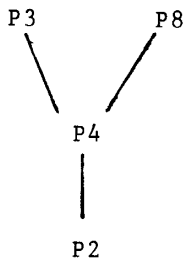



Figure 6c

If, however, we are merely concerned with how one process feeds another, that is, with the network of processes, and are not concerned with how a process might break down further, or with how a group of processes might be taken as a superprocess, then the data base in Figure 7 with relations PROCESS and STREAM is necessary and sufficient. [CATALYST and MASTER-CATALYST, not shown, must be added, as in Figure 2, however.]

A tuple of STREAM requires a minimum of four fields. To see why, consider the network of proceses shown in Figure 8a, and corresponding STREAM relation in Figure 8b. Process P4 feeds process P12, and there is a corresponding tuple <P4, 1, P12, 1>. The first field UP-PROC-ID gives the upstream process of the pair, followed by its occurrence field U-OCCUR, and the third field DOWN-PROC-ID gives the downstream process of the pair, followed by its occurrence number D-OCCUR. Occurrence number fields are needed since the same type of process can occur more than once in the network, as for example, P4, which occurs three times. If there is no matching upstream or downstream process there is a null field value. From

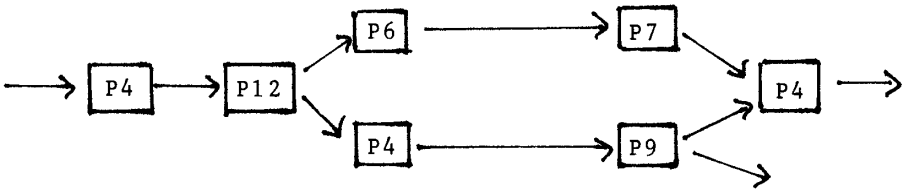



Figure 8a

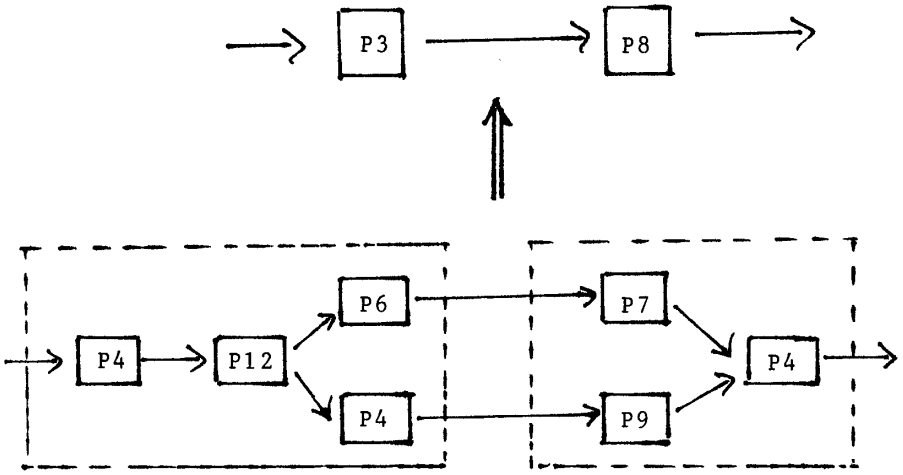



Figure 9a

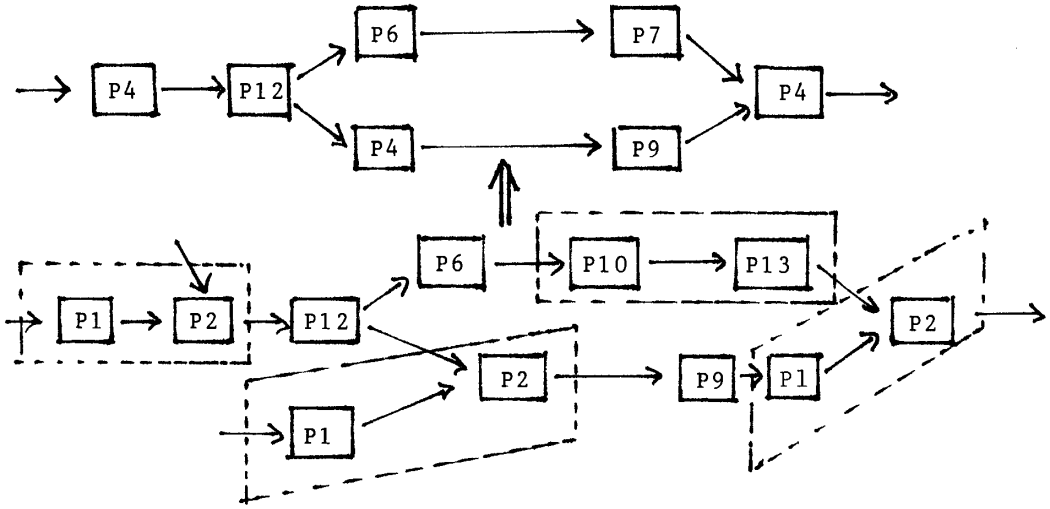



Figure 9b

COMPOSITION, explosions and implosions can be generated. For example, the downstream explosion for P12 and the upstream implosion for P4 occurrence 3 are shown in Figures 8c and 8d respectively.

| UP-PROC-ID | U-OCCUR | DOWN-PROC-ID | D-OCCUR |
|---|---|---|---|
| - | - | P4 | 1 |
| P4 | 1 | P12 | 1 |
| P12 | 1 | P6 | 1 |
| P12 | 1 | P4 | 2 |
| P6 | 1 | P7 | 1 |
| P7 | 1 | P4 | 3 |
| P4 | 2 | P9 | 1 |
| P9 | 1 | P4 | 3 |
| P4 | 3 | - | - |

STREAM

Figure 8b

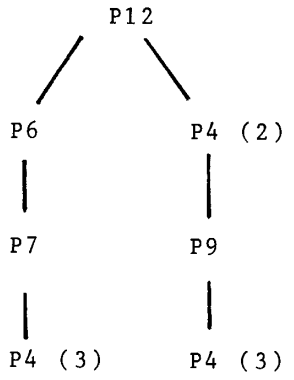



Figure 8c

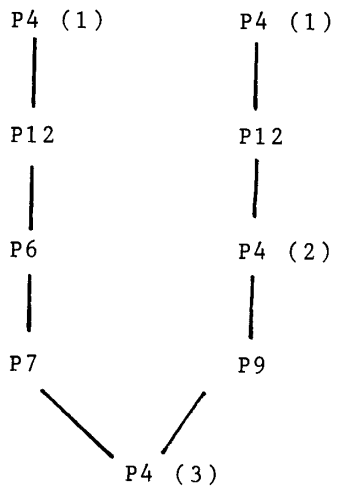



Figure 8d

(A graphics system would be required to display the processes as the boxed network as in Figure 8a; the data retrieved by the data base system would simply feed the graphics system.)

But suppose now that we are not just interested in the process breakdowns and integration for the network in Figure 8a. Suppose that the breakdown hierarchies in Figure 6a apply to the network in Figure 8a. Thus at a higher level (of abstraction), Figure 8a is integrated to involve just two processes (Figure 9a), and can be broken down further to give the detailed network in Figure 9b (and, in consequence, adding more tuples to STREAM in Figure 8b).

The problem is how to combine the data base structures in Figures 5 and 7, that is, combine the relations COMPOSITION and STREAM, so that composition explosions and implosions, upstream implosions and downstream explosions, and the shape of the network of

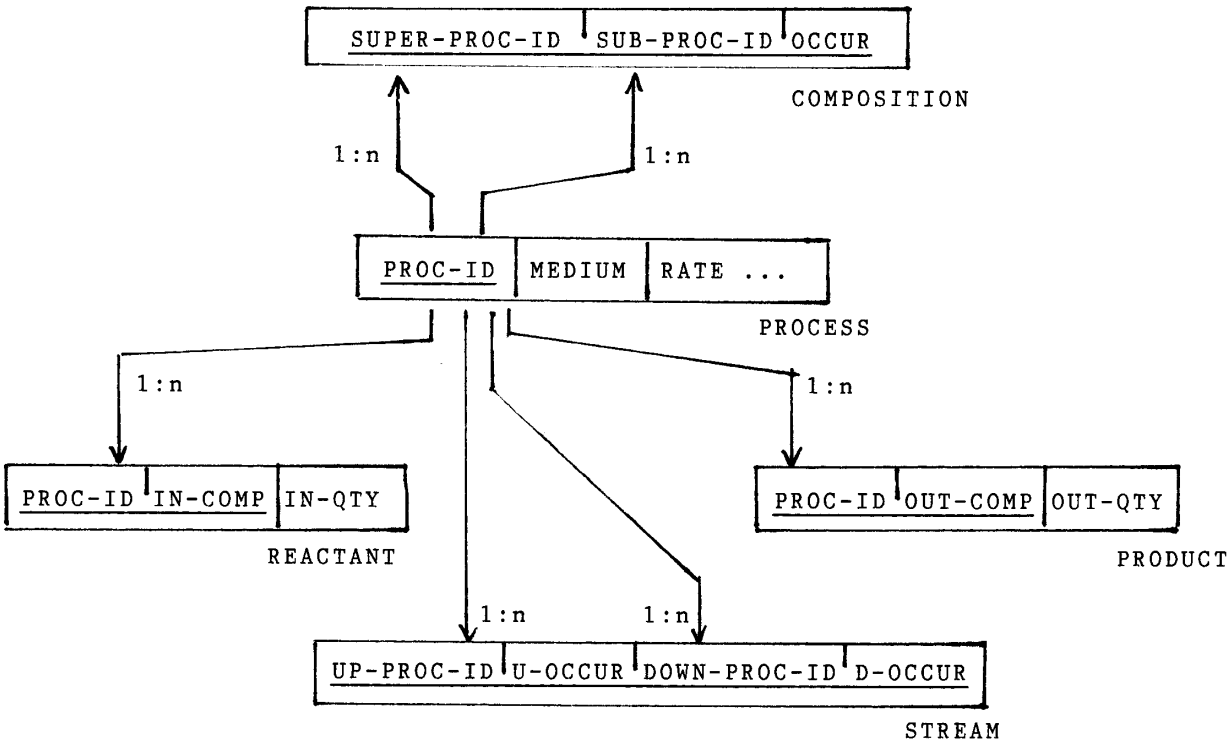

SUPER-PROC-ID
SUB-PROC-ID
OCCUR
COMPOSITION
1:n
1:n
PROC-ID
MEDIUM
RATE ...
PROCESS
1:n
1:n
PROC-ID
IN-COMP
IN-QTY
REACTANT
PROC-ID
OUT-COMP
OUT-QTY
PRODUCT
1:n
1:n
UP-PROC-ID
U-OCCUR
DOWN-PROC-ID
D-OCCUR
STREAM

Figure 10

processes at any level of composition breakdown, may be extracted correctly. As things stand with the separate COMPOSITION and STREAM relations, the occurrence numbers do not (and can not) match. In COMPOSITION (Figure 6b), P4 has two occurrences, but in STREAM (Figure 8b); at a lower level (Figure 9b), P2 has three distinct occurrences in STREAM, but only 1 in COMPOSITION.

One solution is to combine the relations COMPOSITION and STREAM in a single data base, as in Figure 10. Since removal of STREAM from this data base gives us the original data base in Figure 5, and the removal of COMPOSITION gives us the original data abse in Figure 7, it is clear that everything that can be retrieved from the original data bases in Figures 5 and 7 can be retrieved from this composite data base. Unfortunately, that is all it can do, and it cannot tell us where a particular process in the network flow of processes occurs in the composition breakdowns, for example, referring to process P2 in Figure 9b, we could not tell from the data base if it was descended from process P3 or P8 (Figure 6a). A more powerful solution is thus needed.

**A single relation for composition and stream recursivity**

The solution is the relation COMPOSTREAM in the data base in Figure 11a, as a replacement for both COMPOSITION and STREAM from Figure 10. In the general case of a composition breakdown, with multiple composition levels, COMPOSITION consists of tuples, each of which has two full process path names, the first for an upstream process and the second for the downstream process that is fed by the first.

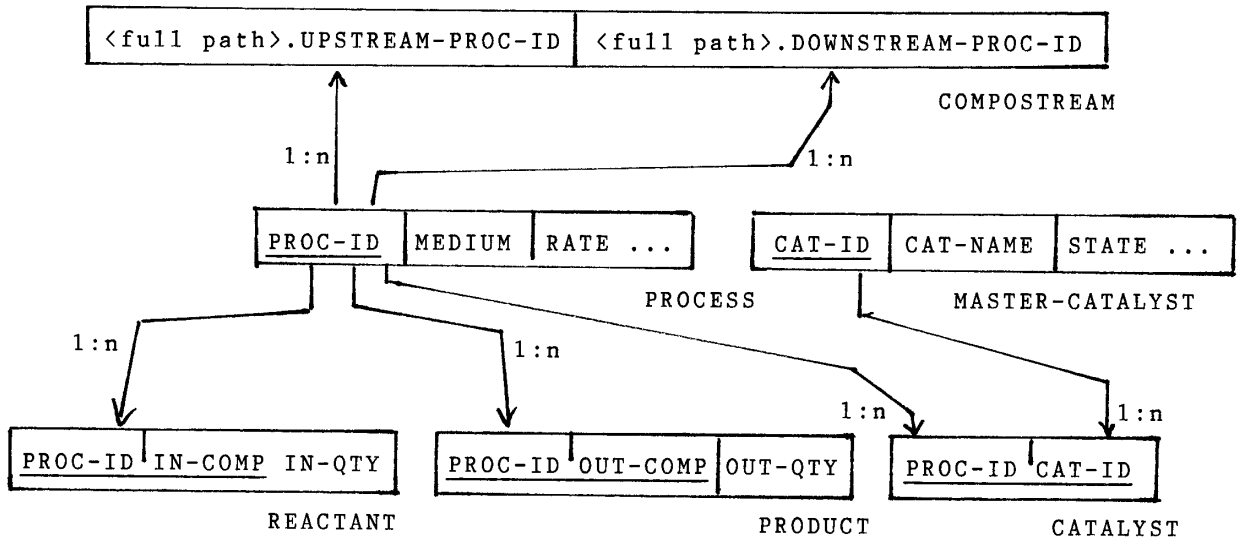

<full path>.UPSTREAM-PROC-ID
<full path>.DOWNSTREAM-PROC-ID
COMPOSTREAM
1:n
1:n
PROC-ID
MEDIUM
RATE ...
CAT-ID
CAT-NAME
STATE ...
PROCESS
MASTER-CATALYST
1:n
1:n
1:n
1:n
PROC-ID
IN-COMP
IN-QTY
PROC-ID
OUT-COMP
OUT-QTY
PROC-ID
CAT-ID
REACTANT
PRODUCT
CATALYST

Figure 11a

An explanation of full process path names is called for before proceeding further. Process identifiers, as in the PROCESS relation are necessary in COMPOSTREAM to identify the upstream and downstream processes, but are **insufficient.** For example, suppose an upstream process P100. There could three occurrences of P100 in the super or parent process P1000 and 4 in another parent process P2000. [Remember composition is based on a recursive many-to-many relationship.] We need to identify (say) occurrence 2 of P100 within P1000, written as P1000.P100.2, and not occurence 3 of P100 within P2000, written as P200.P100.3. But, in turn, there could be three occurrences of P1000 within grandparent process P10,000, and in addition, there could be more than one P10,000 grandparent process. If follows that to fully identify which P100 we are referring to we need to include the occurrence number of the parent within the grandparent and the occurence number of the grandparent within the greatgrandparent, and so on. Thus, if there are four levels within the composition breakdown, four level process path names are needed to identify both the upstream and downstream processes in COMPOSTREAM, for example P100000.3.P10000.2.P1000.3.P100.2.

As a further example, referring to Figure 6a, there are three distinct occurrences of process P2, each of which is distinct and plays a different part in the network flow of processes in Figure 9b. To uniquely identify these three distinct P2 processes for use in COMPOSTREAM, their full path names would be used, as follows:

P3.1.P4.1.P2.1

P3.1.P4.2.P2.1

P8.1.P4.1.P2.1

The contents of COMPOSTREAM are shown in Figure 11b, assuming a composition breakdown as shown in Figure 6a and process networks as in Figure 8a, 9a and 9b.

| UGP | UGP-0 | UP | UP-0 | UPRO | UPRO-0 | DGP | DGP-0 | DP | DP-0 | DPRO | DPRO-0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | P3 | 1 | P4 | 1 | P1 | 1 |
| P3 | 1 | P4 | 1 | P1 | 1 | P3 | 1 | P4 | 1 | P2 | 1 |
| P3 | 1 | P4 | 1 | P2 | 1 | P3 | 1 | P12 | 1 | - | - |
| P3 | 1 | P12 | 1 | - | - | P3 | 1 | P6 | 1 | - | - |
| P3 | 1 | P12 | 1 | - | - | P3 | 1 | P4 | 2 | P2 | 1 |
| - | - | - | - | - | - | P3 | 1 | P4 | 2 | P1 | 1 |
| P3 | 1 | P4 | 2 | P1 | 1 | P3 | 1 | P4 | 2 | P2 | 1 |
| P3 | 1 | P6 | 1 | - | - | P8 | 1 | P7 | 1 | P10 | 1 |
| P3 | 1 | P4 | 2 | P2 | 1 | P8 | 1 | P9 | 1 | - | - |
| P8 | 1 | P7 | 1 | P10 | 1 | P8 | 1 | P7 | 1 | P13 | 1 |
| P8 | 1 | P7 | 1 | P13 | 1 | P8 | 1 | P4 | 1 | P2 | 1 |
| P8 | 1 | P9 | 1 | - | - | P8 | 1 | P4 | 1 | P1 | 1 |
| P8 | 1 | P4 | 1 | P1 | 1 | P8 | 1 | P4 | 1 | P2 | 1 |
| P8 | 1 | P4 | 1 | P2 | 1 | - | - | - | - | - | - |

COMPOSTREAM

Figure 11b

The field names making up the full path process names are in this instance:

| | |
|---|---|
| UGP | upstream grandparent process |
| UGP-O | upstream grandparent process occurrence number |
| UP | upstream parent process |
| UP-O | upstream parent process occurrence number |
| UPRO | upstream process |
| UPRO-O | upstream process occurrence number |
| | |
| DGP | downstream grandparent process |
| DGP-O | downstream grandparent process occurrence number |
| DP | downstream parent process |
| DP-O | downstream parent process occurrence number |
| DPRO | downstream process |
| DPRO-O | downstream process occurrence number |

Where there is no upstream or downstream process, null values are used for the full path name. Where there is no lower level composition breakdown, for example, with P6, null values are used for the lowest level fields.

COMPOSTREAM, as is, gives directly the lowest level breakdown of the flow network in the data base, that is, using the example, the level-3 flow network in Figure 9b. The flow network at the next level up (level 2, as in Figure 8a) can be easily retrieved with a simple SQL expression, and, if desired, made available to users as a view:

```
CREATE VIEW VIEW-LEVEL-2
      AS SELECT (UGP, UGP-0, UP, UP-0, DGP, DGP-0, DP, DP-0)
         FROM COMPOSTREAM;
```

which is equivalent to a relational algebra projection on the grandparent and parent columns of COMPOSTREAM. Readers are invited to check that the result, in Figure 11c, does indeed correspond exactly to the level-2 network in Figure 8a.

| UGP | UGP-0 | UP | UP-0 | DGP | DGP-0 | DP | DP-0 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | P3 | 1 | P4 | 1 |
| P3 | 1 | P4 | 1 | P3 | 1 | P4 | 1 |
| P3 | 1 | P4 | 1 | P3 | 1 | P12 | 1 |
| P3 | 1 | P12 | 1 | P3 | 1 | P6 | 1 |
| P3 | 1 | P12 | 1 | P3 | 1 | P4 | 2 |
| - | - | - | - | P3 | 1 | P4 | 2 |
| P3 | 1 | P4 | 2 | P3 | 1 | P4 | 2 |
| P3 | 1 | P6 | 1 | P8 | 1 | P7 | 1 |
| P8 | 1 | P7 | 1 | P8 | 1 | P7 | 1 |
| P8 | 1 | P7 | 1 | P8 | 1 | P4 | 1 |
| P3 | 1 | P4 | 2 | P8 | 1 | P9 | 1 |
| P8 | 1 | P9 | 1 | P8 | 1 | P4 | 1 |
| P8 | 1 | P4 | 1 | P8 | 1 | P4 | 1 |
| P8 | 1 | P4 | 1 | - | - | - | - |

VIEW-LEVEL-2

Figure 11c

Similarly, the relation that gives the level 1 view of the network can be retrieved and used as a view with:

```
CREATE VIEW VIEW-LEVEL-1
      AS SELECT (UGP, UGP-O, DGP, DGP-O)
         FROM COMPOSTREAM;
```

with the obviously correct result in Figure 11d.

| UGP | UGP-O | DGP | DGP-O |
|---|---|---|---|
| - | - | P3 | 1 |
| P3 | 1 | P8 | 1 |
| P8 | 1 | - | - |

VIEW-LEVEL-1

Figure 11d

In an object oriented approach, these basic views could be useful objects.

**Redundancy elimination in reactant and product relations versus ease of use**

The REACTANT and PRODUCT relations in the final solution in Figure 11a could each contain a set of input reactant tuple or output product tuples for each process listed in the relation PROCESS.

Thus, for example, if P3 has reactants A, B, C and products W Z, then REACTANT and PRODUCT could be expected to contain the tuples:

| REACTANT | PRODUCT |
|---|---|
| P3 A ... | P3 W ... |
| P3 B ... | P3 Z ... |
| P3 C ... | ... |
| ... | ... |

However, this data would be redundant, and so can, at least in theory , be dispensed with, although this may not be wise in practice. To see this, consider that P3 is breaks down ultimately (Figure 6a) into processes P1, P2, P6, P1, P2, P12 which are arranged in a part of the network shown in Figure 9b. This information is obtainable directly from COMPOSTREAM (Figure 11b). Accordingly, the inputs and outputs of this group of processes must be the inputs and outputs of P3. The inputs and outputs of the group can be deduced from the inputs and outputs of each of the individual processes of the group. Thus it is strictly necessary to store only the reactants and products for the lowest level processes in each breakdown hierarchy, from which data the reactant and products of higher level processes can be deduced. In the case of P3, its upstream and down stream interactions are described by the tuples from COMPOSTREAM retrieved by:

```
SELECT * FROM COMPUSTREAM
WHERE UGP = 'P3' OR DGP = 'P3';
```

giving us the tuples shown in Figure 12.

| UGP | UGP-O | UP | UP-O | UPRO | UPRO-O | DGP | DGP-O | DP | DP-O | DPRO | DPRO-O |
|---|---|---|---|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | P3 | 1 | P4 | 1 | P1 | 1 |
| P3 | 1 | P4 | 1 | P1 | 1 | P3 | 1 | P4 | 1 | P2 | 1 |
| P3 | 1 | P4 | 1 | P2 | 1 | P3 | 1 | P12 | 1 | - | - |
| P3 | 1 | P12 | 1 | - | - | P3 | 1 | P6 | 1 | - | - |
| P3 | 1 | P12 | 1 | - | - | P3 | 1 | P4 | 2 | P2 | 1 |
| - | - | - | - | - | - | P3 | 1 | P4 | 2 | P1 | 1 |
| P3 | 1 | P4 | 2 | P1 | 1 | P3 | 1 | P4 | 2 | P2 | 1 |
| P3 | 1 | P6 | 1 | - | - | P8 | 1 | P7 | 1 | P10 | 1 |
| P3 | 1 | P4 | 2 | P2 | 1 | P8 | 1 | P9 | 1 | - | - |

Figure 12

The lowest level processes, within this set of tuples, that do interact with non P3 processes or with no process at all, either upstream or downstream, are the processes at the interface of the P3 group of lowest level processes; the reactants of this subset of processes must be the reactants of P3 and the products of this set of processes must be products of P3. These tuples could be retrieved by:

```
SELECT * FROM COMPUSTREAM
WHERE UGP = 'P3' OR DGP = 'P3'
      AND ((UGP IS NULL OR DGP IS NULL)
           OR (UGP NOT = 'P3' OR DGP NOT = 'P3'));
```

giving the tuples in Figure 13.

| UGP | UGP-O | UP | UP-O | UPRO | UPRO-O | DGP | DGP-O | DP | DP-O | DPRO | DPRO-O |
|---|---|---|---|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | P3 | 1 | P4 | 1 | P1 | 1 |
| - | - | - | - | - | - | P3 | 1 | P4 | 2 | P1 | 1 |
| P3 | 1 | P6 | 1 | - | - | P8 | 1 | P7 | 1 | P10 | 1 |
| P3 | 1 | P4 | 2 | P2 | 1 | P8 | 1 | P9 | 1 | - | - |

Figure 13

From these tuples it can be seen that that reactants of P3 are those of P1, P1, whereas the products are those of P6, P2. The reactants and products for these processes can then be easily obtained from REACTANT and PRODUCT. The complete SQL expression for the products of P3 would be:

```
SELECT OUT-COMP FROM PRODUCT
WHERE PROC-ID IN
        ( SELECT UPRO FROM COMPOSTREAM
          WHERE DGP NOT = 'P3'
          AND  ((UGP = 'P3' OR DGP = 'P3'
          AND ((UGP IS NULL OR DGP IS NULL)
```

```
        OR (UGP NOT = 'P3' OR DGP NOT = 'P3'))));
OR PROC-ID IN
      ( SELECT UP FROM COMPOSTREAM
        WHERE UPRO IS NUL AND DGP NOT = 'P3'
        AND  ((UGP = 'P3' OR DGP = 'P3'
        AND ((UGP IS NULL OR DGP IS NULL)
        OR (UGP NOT = 'P3' OR DGP NOT = 'P3')));
```

The conditions could be written more simply, but for ease of understanding have been kept consistent with the step-by-step buildup in the earlier SQL expressions for retrieving the subsets of COMPOSTREAM in Figures 12 and 13. A similar expression would be needed to retrieve the reactants of P3.

This must appear somewhat complex, but the complexity appears only because we are squeezing every last bit of redundancy out of the data base, by listing only lowest level processes in REACTANT and PRODUCT. Complexity of retrieval expressions is not the only disadvantage however. Notice that the molecular quantities to balance the process were not retrieved by the SQL expression above. The information is in the data base, but cannot be retrieved by SQL alone. To compute the overall products from a group of processes requires balancing the reactants and products of all the process in the group, and would have to be carried out by a procedural routine, written in a higher level language, containing the necessary SQL expressions, or by equivalent routines attached to a class in an object oriented approach.

A far simpler alternative is to live with the redundancy caused by having all processes, regardless of level, listed in

REACTANT and PRODUCT. If we do that, the SQL expression to retrieve the products of process P1 is the almost trivial:

```
SELECT OUT_COMP, OUT_QTY FROM PRODUCT
WHERE PROC_ID = 'P1';
```

and this time the quantities required to balance input and output are in OUT_QTY.

The author is in favor of this simpler approach, even at the expense of some redundancy. Since in practice it is unlikely that more that 4 levels of composition would be needed, the extra tuples would probably not increase the size of REACTANT and PRODUCT by more than 50%. As a percentage of the total data base this redundancy would likely be less than 10%, which, in these days of inexpensive storage hardware, is a small price to pay for a significant saving in the complexity of retrieval expressions, and elimination of the need for additional non SQL routines.

From these considerations it follows that the optimum design is that in Figure 11a, where the relations COMPOSITION and STREAM have been combined to form the relation COMPOSITREAM, and where REACTANT and PRODUCT have all processes listed, regardless of level in the breakdown hierarchy.

Similar redundancy versus ease-of use considerations should be applied to CATALYST. Thus we can either have records in CATALYST for every process, regardless of level, or we can have just CATALYST records for the lowest level processes, which will be reactions. It is simpler, from a retrival viewpoint, to list all

processes in CATALYST. However, in this case, the level of redundancy is quite high, since a single high level process that breaks down ultimately in perhaps 20 reactions, each of which needs one or two catalysts, will generate as many as 40 redundant tuples in CATALYST. On the other hand, it can be argued that it is unlikely that anyone will ever want just the catalysts needed with a high level process; only the catalysts used with the ultimate breakdown reactions will be needed, so that listing only lowest level processes, or reactions, in CATALYST will be acceptable. Which avenue is chosen probably will reflect the detail design criteria for the data base. All we can say here is that storing only catalysts for lowest level processes will always be sufficient - catalysts for higher level processes can always be deduced.

**Manipulation of the comprehensive chemical process data base**

The final data base in Figure 11a contains a single recursive many-to-many relationship - for upstream and downstream explosions. Instead of being handled by a recursive relationship, as in the original relation COMPOSITION, the composition breakdown of a process is fully spelled out by the full path names in COMPOSTREAM, although essentially, it is still present.

The design clearly complies with the original design aims. It covers the breakdown of processes to the reaction level, and covers the arrangement of processes as a flow network at any level of composition breakdown, and it will do this faithfully for any network and any composition breakdown, no matter how complex.

Some SQL retrieval expression illustrate the use of the data base. Three levels of breakdown, and COMPOSTREAM fields names as in Figure 11b, are assumed.

**Example 1. Retrieve a full description of each level-1 process that has at least two levels of breakdown?**

```
SELECT * FROM PROCESS
WHERE PROC-ID IN (SELECT UGP
       FROM COMPOSTREAM WHERE UP IS NOT NULL)
```

**Example 2. For each subprocess of level-1 process P30, determine the subprocess identifier and the catalysts needed with that subprocess.**

```
SELECT PROC-ID, CAT-ID,
FROM CATALYST
WHERE PROC-ID IN (SELECT UP FROM COMPOSTREAM
        WHERE UGP = 'P30');
```

**Example 3. How many processes feed into level 3-process P25, occurence 2.**

```
SELECT COUNT (*) FROM COMPOSTREAM
WHERE DPRO = 'P25'
AND UGP IS NOT NULL.
```

**Example 4 If unique level-2 process P20 has a cyclic level-3 substructure, get this substructure, complete with reactants and products for each subprocess of P20.**

```
SELECT IN-COMP, UPRO, UPRO-O, OUT-COMP
FROM REACTANT, COMPOSTREAM, PRODUCT
WHERE REACTANT.PROC-ID = UPRO AND PRODUCT.PROC-ID = UPRO
      AND UP = 'P20'
      AND UP IN (SELECT INPROC-ID FROM PROCESS
                 WHERE TYPE = 'CYCLE')
```

An additional optional field TYPE is assumed in the relation PROCESS here. It could have values 'CYCLE', 'FUNNEL', 'FAN', 'OTHER", and so on, as an aid to retrievals

**Lack of a recursive version of SQL**

A disadvantage of current releases of SQL is that the language does not have a facility for handling recursion to an unknown number of levels. For example, referring to the data base in Figure 11a again, suppose one wants the furtherest downstream subprocesses of a given process, where it is not known in advance how many recursion levels down the furtherest downstream process lies. There is no SQL expression that can express the retrieval request. This is not a drawback of the data base design, but of SQL. Eventually,

a recursive version of SQL will likely become commercially available, enabling retrievals involving unknown numbers of recursion levels to be expressed. Currently there are two possible ways of dealing with this SQL defect in the case of the chemical process data base in Figure 11a.

One approach is to use a distinct SQL expression for each recursion level, and work upwards in to the implosion, or downwards into explosions, using an SQL expression, usually the same one, at each level.

The alternative is to guess how many levels there are, and then construct an SQL expression to retrieve the subprocesses at the guessed level. If no processes are retrieved, then the number of levels guessed was too large. The number of levels in the expression can then be reduced by one and the expression tried again. If this fails reduce by one and try again, repeating until some processes are retrieved. These are the lowest level processes. If the first expression did not fail, then increase the number of levels in the expression by one, and repeat until a failure occurs.

Using techniques of this nature the lack of a recursive feature in SQL is no great handicap, and the user can quickly determine the nature of the breakdown of any process.

**Graphics display of process and molecular structures**

A data base with the structure in Figure 11a could be used to generate data on process substructure that could be displayed by a graphics display system. Displays could be similar to those shown

in Figure 9b. At the same time, if a comprehensive molecular structure data base [6] were available, the chemical structure of the reactants and products for each process could also be displayed. This would be especially valuable when the process (and breakdown subprocesses) involved changes to a giant molecule, such as an enzyme, protein or DNA molecule [7, 12]. If the giant molecule were involved in a complex series of changes, the processes for which were in the process data base, then the sequence of molecular structure changes could easily be displayed.

**Summary**

A fundamental relational data base structure for comprehensive data about chemical processes is proposed. The data base can hold data about the physical attributes of chemical processes and about their reactants, products and catalysts. It includes a relation about subprocesses, embodying a recursive many-to-many relationship. This recursive relationship holds data about the breakdown of any process into its subprocesses, and the breakdown of these subprocesses into their subprocesses and so on, to any desired breakdown level. In addition, it holds the data about how the processes and subprocesses feed molecules to other processes, thus giving the process flow networks at each level of breakdown, regardless of the complexity of the flow network structure (sequence, cycle, funnel, fan or other arbitrary structure).

The relational data base language SQL, commonly available commercially, can be used with the data base. Data about processes,

in terms of physical properties, reactants and products, and breakdown into subprocesses, can be retrieved easily, normally by means of one or two SQL expressions.

The chemical process data base structure presented in this contains no facilities for data about the structure (and substructure) of the molecules involved in the processes. However, in an earlier paper a relational data base structure was presented for structure and substructure, down to the atomic level, of chemical compounds [6]. Where molecular substructure data is needed in addition to chemical process data, then both the chemical process data base and the molecular structure data base could be used. These data bases together would be especially useful with a graphics system, for displaying the effect of a sequence of chemical processes on molecular structure, particularly when giant molecules are involved.

**References**

1. Allen F.H., Kennard, O., 1983. The Cambridge Data Base of molecular structures, Perspect. Computing, 3(3), 28-43.

2. Attias, R., Dubois, J. E., 1990. Substructure systems, concepts and classification, J. Chem. Inf. Comput. Sci., 30(1), 2-7.

3. Bernstein, F.C., et al, 1977. The protein databank: A computer-based archival file for macromolecular structures, Journal of Molecular Biology, Vol. 112, 535-542.

4. Berzins, V., Jones, K., 1989. Maximizing the potential of process engineering databases, Comput. Integr. Process Eng., 114, 225-41.

5. Bradley, 1982. File & Database Techniques, Holt, Rinehart & Winston, New York.

6. Bradley, J., 1990. A two-path recursive relational database structure for molecular information systems, J. Mol. Graphics, Vol. 8, 2-10.

7. Bryant, S.H., Sternberg, M.J.E.,1987. Comparison of protein structural profiles by interactive computer graphics, Journal of Molecular Graphics, 5(1), 4-7.

8. Date, C.J. Introduction to Database Systems, Vol. 1, 5th ed., Addison-Wesley, 1990.

9. Fletterick, R.J., Schroer, T., and Matela, R.J., 1985. Molecular structure: Macromolecules in Three-Dimensions, Blackwell Scientific Publications.

10. IBM Corp., 1984. IBM DATABASE2 General Information, Form GC26-4073, IBM Corp., Endicott, New York.

11. ORACLE Corp., 1984. ORACLE SQL/UFI Reference Guide, Menlo Park, California.

12. Jakes, S.E., Willet, P., 1986. Pharmacophoric pattern matching in files of 3-dimensional chemical structures: Selection of inter-atomic distance screens, Journal of Molecular Graphics, 4(1), 12-20.

13. Kim, Won, 1979. Relational database systems, ACM Computing Surveys, 11, pages 185-211.

14. Maier, D. 1983. Theory of Relational Databases, Computer Science Press, Potomac, Maryland.

15. Morfew, A.T., Todd, S.J.P, Snelgrove, M.J., 1983. The use of a relational data base for holding molecule data in a molecular graphics system, Computers & Chemistry, 7(1), 9-16.

16. Motard, R. L., 1989. Integrated computer-aided process engineering, Comput. Chem. Eng., 13(11-12), 1199-206.

17. Relational Technology Inc., 1984. INGRES Reference Manual, Berkeley, Calif.

18. Ricketts, D., 1987. Perq: interactive molecular modelling systems, Journal of Molecular Graphics, 5(2), 63-70.

19. Stryer, L., 1988, Biochemistry, W.H. Freeman & Co., New York.

20. Taylor, R, 1986. The Cambridge Structural Database in molecular graphics: Techniques for the rapid identification of conformal minima, Journal of Molecular Graphics, 4(2), 123-131.