R Q L User Manual

Relational Query Language

January 15, 1982

McGill University

School of Computer Science
Burnside Hall (514) 392-8275

16 April 1982

TO WHOM IT MAY CONCERN

Prof. D.J. Burrage and Mr. Michael Gilman of the School of Computer Science at McGill have developed a relational algebra query language, RAQL, which I would like to endorse.

The language is based on the syntax used in the popular textbook by C.J. Date (Addison Wesley, 1977) and so is a valuable teaching tool in a database course. RAQL offers all the facilities described in Date's book, and more, and is implemented so as to run at a reasonable expense on any teaching database. It could be very useful in any course intended to give students direct experience of the variety of state-of-the-art database models and facilities.

RAQL interfaces with the Statistical Analysis System (SAS), which is a widely used tool for data manipulation and display, but lacks a systematic approach to databases. Anybody who wants to perform database operations on his SAS data would be well advised to have access to RAQL. This is a second area of great benefit from RAQL.

Too little university developed software is adequately disseminated, leading undoubtedly to frequent duplications of effort. I think it is admirable that the developers of this useful product are taking the trouble to polish and distribute it.

Yours truly,

T.H. Merrett
Associate Professor

pr

Postal address: 805 Sherbrooke Street West, Montreal, PQ, Canada H3A 2K6

PROVIDING FEEDBACK TO RQL

Any comments, criticisms and/or suggestions regarding RQL would be much appreciated. Users of MUSIC at McGill can interactively send such feedback to RQL via the RQLCOMMENTS facility.

First, create a regular MUSIC file containing your comments. Then enter; RQLCOMMENTS. This program will prompt you for the name of the MUSIC file containing your comments. Enter the file name.

TABLE OF CONTENTS

INTRODUCTION

RQL - Relational Query Language - is a relational data base management system based on the operations of relational algebra as first proposed by Codd. RQL's query language is an expanded version of the query language presented in the book by C. J. Date, Introduction to Database Systems, 2nd Edition, Chapter 6. Although familiarity with Date's query language would be advantageous, it is not a necessity for purposes of understanding RQL. The user is expected, however, to have an elementary knowledge of the theoretical constructs of the relational algebra. This knowledge should include an understanding of the traditional set operations; UNION, INTERSECTION and DIFFERENCE as well as the special set operations of the relational algebra; JOIN, DIVIDE, SELECT and PROJECT. The reader is referred to Date's book if this understanding is lacking.

RQL was written primarily to provide a relational data base system which would act as a powerful supplement to the SAS programming language. SAS is basically a statistical and report writing package that currently enjoys widespread acceptance in business and scientific environments.

The main data object in SAS is the file. As SAS is intended for essentially naive users, the manipulation of files is facilitated through the use of powerful and concise SAS statements. As flexible as SAS is for handling files, it lacks the necessary features to forthrightly implement a relational data base system. To make SAS more complete therefore, I decided to fill this perceived void.

RQL works by translating the user's query into a SAS source program which is then executed to ultimately answer the query. The advantages of this method are;

a) As previously mentioned, SAS has existing powerful features for manipulating files. These were used to simplify my task.

b) No changes to existing SAS data sets are required for their use as operands in a query. Although a user may wish to conform SAS data sets to the theoretical structure of relations, this is not necessary(though perhaps advisable).

c) As the result of any query is a relation, which is a SAS data set, all of SAS's editing, report writing and statistical features may be brought to bear on it.

This last point is worth reinforcing. Many existing data base packages lack sophisticated tools for analyzing and printing results of queries. As this further manipulation can be an important and common requirement, a query language imbedded in a host language having these features is desirable. SAS has the needed features(RQL is not actually technically imbedded in SAS. See REGULAR USERS OF SAS for details).

Although RQL is geared for those familiar with SAS, it can be used as a stand-alone system; i.e. no knowledge of SAS is necessary to run RQL. Users are urged however to acquaint themselves with the fundamentals of SAS for two reasons:

a) Although RQL generates bug-free code, the occasional logic and/or system error is bound to occur. The error messages will be mostly output by SAS, thus familiarity with its basic elements will facilitate message comprehension.

b) SAS is a very useful package in its own right.

The terms data base, relation, SAS data set and OS data set have related but different meanings. Here follows an attempt at clarification.

An OS data set is the data set given on the user's DD card.

```
//SUPPLIER   DD  DSN=H2A5.$MASTER,DISP=SHR
```

In the above JCL card, H2A5.$MASTER is the name of an OS data set. A SAS data set resides in a given OS data set. A data base, in the context of this manual, is the collection of all SAS data sets that reside in a given OS data set. A relation corresponds to a particular SAS data set.

The major part of this manual is devoted to presenting and explaining the query language that provides access to the data base. Each query category is presented in several steps; each step a variation of that query. The variations are intended to be in ascending order of complexity; the complexity resulting from the introduction of options. However, familiarization with the first and/or second step of each query category should be sufficient to use RQL.

The more adventurous and inquisitive reader is urged to discover what query language options are available. The initial effort should pay off in reduced programming time and/or greater task endeavors.

Although RQL is intended to be a complete relational data base management package, some operations found in other relational systems have been omitted. Specifically, an INPUT and UPDATE capability for relations has not been provided (though UPDATE can be implemented with a MINUS and UNION- also see SELECT). This is mainly because SAS itself handles these two functions quite adequately. I deemed it wasteful and unnecessary to duplicate functions already existing in a thorough form. Even though RQL is presented as a stand-alone system, I think it reasonable in the case of INPUT and UPDATE that the user refer to the SAS manual.

HOW RQL WORKS: AN OVERVIEW

RQL is divided into three job steps. These will be outlined in turn.

First Job Step

A SAS program is executed for the first job step. The directory of each data base given on the EXEC card is obtained and stored in a temporary OS data set. The contents of the directory includes;

1) The attribute names of the attributes in each data base.
2) The length(# of bytes) of each attribute.
3) The type(character or numeric) of each attribute.

This directory information is passed to the second job step where it is used to aid the translation of queries to a SAS source program.

There are two critical errors that can occur in the first job step. The first is when the data base given on the EXEC card has no corresponding JCL card defining the OS data set on which the data base resides. Note that the ddname of the corresponding JCL card must match the data base name on the EXEC card(see HOW TO RUN RQL).

The second critical error occurs when a data base does not contain at least one SAS data set. This error can be avoided by ensuring that all data bases included in an RQL program have at least one SAS data set.

Second Job Step

A SNOBOL program translates the user's query into a SAS source program. This SAS source program is executed in the third job step. All error detection under the control of RQL is done in this step. Using SNOBOL allowed the quick and easy definition of the grammar on which the RQL query language is based. Furthermore, modifications and supplements to the grammar can be implemented with ease with little or no side effects. Although there is some overhead incurred for the translation to SAS source code, the actual execution time and cost is minimal when measured against the cost of the SAS program which performs the data base operations(See HOW RQL WORKS for some details of execution).

A list of the more important errors that can occur follows.

1) A result relation has the same name as a relation in the user's data base and it is NOT prefixed by the data base name(see JOIN (9)).

2) A relation given in a query does not exist.

3) A query is syntactically incorrect.

4) Attributes given in a query do not exist or are used improperly. An example of improper usage would be if an attribute in a JOIN query is renamed to an existing attribute name(see JOIN (3)).

To confirm the successful translation of a query, a message to that effect is printed.

Third Job Step

The SAS program generated by the second job step is here executed. Although the program should be bug-free, some system and perhaps logic errors may occur. If so, the user is expected to analyze these errors at the SAS level. That is, RQL does not intercept errors in the third job step. Any errors occurring here forces the user to 'get into' SAS as much as is required to solve the problem.

The relations used in the query examples are those taken from Date's SUPPLIER data base supplemented with some relations of my own creation. The next page illustrates these relations. The reader is urged to have a quick look at these relations before proceeding to the presentation of the queries themselves.

RELATIONS USED IN THE QUERY EXAMPLES

S

SNUM	SNAME	STATUS	CITY
S1	SMITH	20	LONDON
S2	JONES	10	PARIS
S3	BLAKE	30	PARIS
S4	CLARK	20	LONDON
S5	ADAMS	30	ATHENS

P

PNUM	PNAME	COLOR	WEIGHT
P1	NUT	RED	12
P2	BOLT	GREEN	17
P3	SCREW	BLUE	17
P4	SCREW	RED	14
P5	CAM	BLUE	12
P6	COG	RED	19

J

JNUM	JNAME	CITY
J1	SORTER	PARIS
J2	PUNCH	ROME
J3	READER	ATHENS
J4	CONSOLE	ATHENS
J5	COLLATOR	LONDON
J6	TERMINAL	OSLO
J7	TAPE	LONDON

PQ

SNUM	JNUM	QTY
S1	J1	100
S2	J3	300
S2	J4	200
S3	J2	800
S3	J4	400

SPJ

SNUM	PNUM	JNUM	QTY
S1	P1	J1	200
S1	P1	J4	700
S2	P3	J1	400
S2	P3	J2	200
S2	P3	J3	200
S2	P3	J4	500
S2	P3	J5	600
S2	P3	J6	400
S2	P3	J7	800
S2	P5	J2	100
S3	P3	J1	200
S3	P4	J2	500
S4	P6	J3	300
S4	P6	J7	300
S5	P2	J2	200
S5	P2	J4	100
S5	P5	J5	500
S5	P5	J7	100
S5	P6	J2	200
S5	P1	J4	1000
S5	P3	J4	1200
S5	P4	J4	800
S5	P5	J4	400
S5	P6	J4	500

CON

QTY
200

PRESENTATION AND EXPLANATION OF QUERIES

SOME GENERAL OBSERVATIONS AND RULES

- Relation and attribute names cannot exceed 8 characters in length.
- Relation and attribute names must begin with a letter or an underscore. Subsequent characters can only be integers, letters or underscores.
- Except where commas and brackets are used, at least one blank must be inserted between keywords and identifiers. Extra blanks are ignored.
- Queries can be split across a line. Keywords, identifiers and constants cannot be split across a line.
- Each query must begin on a new line and be terminated by a period(.).
- Redundant brackets are not allowed.
- Every query, except for SELECT, involves the sorting of copies of the operand relations as a first step. It should be emphasized that only copies of the relations are sorted. The cperand relations remain untouched.

JOIN

The JOIN implemented by RQL is the natural join. The natural join is an equi-join with one of the two duplicate result attributes projected out.

JOIN (1)

```
JOIN SPJ AND PQ OVER SNUM,QTY GIVING TEMP.
```

The operand relations SPJ and PQ will be joined over SNUM and QTY to produce the result relation TEMP. SNUM and QTY must be common attributes of SPJ and PQ. The result relation, TEMP, will contain the attributes; SNUM,PNUM,JNUM, QTY.

Observations and Rules

- The list of attributes to be joined over is called the 'over-list'.
- Any number of attributes may be given in the over-list.
- The attributes in the over-list are those by which copies of the two operand relations are sorted.
- The order of the attributes in the over-list is the order in which the actual joining proceeds.
- All attributes in the over-list must reside in both operand relations. If not all attributes in the over-list are held in common, the job is terminated with an appropriate error message.
- The order of the attributes in the result relation is determined by the order of the attributes in the first operand relation followed by the order of the attributes in the second operand relation. The attributes in the over-list are placed as they occur in the first operand relation.

```
JOIN SPJ AND J OVER JNUM GIVING TEMP.
```

In the above example, the attributes in TEMP would have the following order;

SNUM, PNUM, JNUM, QTY, JNAME, CITY.

- It is possible for the two operand relations to have common attributes which are of different types. For example, QTY could be declared as a character attribute in SPJ and a numeric attribute in PQ. This means that, technically, QTY is not a common attribute of SPJ and PQ. RQL does <u>not</u> check for the occurence of this unlikely situation and so assumes same-named attributes are, indeed, common attributes. Note, however, that SAS will automatically convert character values to numeric values, where possible. Although this may lead to some problems, it does provide some potentially useful flexibility.
- The result relation is stored in a work area and is not written to the user's OS data set. However, the result relation is available for use in later queries. To permanently keep result relations, see JOIN (8).

JOIN (2)

```
JOIN J AND SPJ GIVING TEMP.
```

Notice that there is no explicit over-list in the above query. The over-list is obtained by determining all the attributes J and SPJ have

in common. In this example, there is only one common attribute; JNUM. J and SPJ are therefore joined over JNUM. If J and SPJ had two attributes in common, the join would be performed over those two.

Note that if the two operand relations have exactly all their attributes in common, the above join, with no declared over-list, implements the standard set operation; INTERSECT.

Remember that if the two relations have no attributes in common, the job is terminated with an appropriate error message.

JOIN (3)

```
JOIN SPJ(RENAME SNUM TO SPJ_SNUM,QTY TO SPJ_QTY) AND PQ OVER JNUM GIVING
TEMP.
```

When joining two relations, often there exist attributes in common over which the relations are not to be joined. In the above example, SPJ and PQ are to be joined over JNUM. The result relation, TEMP, will therefore have JNUM as one of its attributes. As SPJ and PQ also have SNUM and QTY in ccmmon, which SNUM and QTY will be included in TEMP? TEMP cannot have two SNUMs and two QTYs as this will give rise to ambiguous attribute selection at a later time. The solution to the problem is to rename all common attributes exclusive of those in the over-list.

In the above example, the result relation will have the attributes SPJ_SNUM, PNUM, JNUM, SPJ_QTY, SNUM, QTY. SPJ_SNUM, PNUM and SPJ_QTY are taken from relation SPJ while SNUM and QTY are taken from relation PQ. JNUM is the attribute over which SPJ and PQ are joined.

Observations and Rules

- If a join is performed on two relations having common attributes other than those in the over-list(either explicit or implicit), the attributes in the first relation must be renamed. If not, the result relation will contain only one of the non-renamed attributes. The values of this attribute will be those of the attributes in the second operand relation. If this occurs, RQL outputs a warning message.
- There is no limit to the number of attributes that can be renamed.
- Attributes in the over-list cannot be renamed.
- Only attributes in the first relation can be renamed.
- Renaming of attributes does not affect the attribute names of the first operand relation. The new names are only found in the result relation.

JOIN (4)

```
JOIN SPJ AND ('P1',200) OVER PNUM,QTY GIVING TEMP.
```

In the above example, ('P1',200) is known as a constant relation. It has the effect of creating a temporary relation of one tuple and two attributes, PNUM and QTY.

It is possible to have a constant relation which results in a temporary relation of more than one tuple:

```
JOIN SPJ AND ('P1','P4','P7') OVER PNUM GIVING TEMP.
```

The constant relation above will produce the temporary relation:

PNUM
P1
P4
P7

It is also possible to have a constant relation which produces a temporary relation of arbitrary degree(# of attributes) and cardinality(# of tuples).

```
JOIN SPJ AND ('P1','J1',200,'P3','J6',100,'P9','J3',600) OVER
PNUM,JNUM,QTY GIVING TEMP.
```

The above constant relation will produce the temporary relation:

PNUM	JNUM	QTY
P1	J1	200
P3	J6	100
P9	J3	600

Once the temporary relation is formed, it is used as the second operand relation in the JOIN query.

Observations and Rules

- Literals must be enclosed in single quotes.
- Constant relations may be continued on the next line:
  E.G. ('P1','J2',
  200)
- Redundant brackets are NOT allowed.
- The order of the single values in the constant relation must exactly correspond to the order of the attributes in the over-list to which the values are wished to be associated. In the example above, if the over-list was given as; JNUM,PNUM,QTY instead of PNUM,JNUM,QTY, the temporary relation produced would be:

JNUM	PNUM	QTY
P1	J1	200
P3	J6	100
P9	J3	600

  Note that JNUM would contain the 'P' values and PNUM would contain the 'J' values.
- The values of the constant relation must correspond in TYPE with their associated attribute in the over-list. In the query above, 'P1' is associated with PNUM. Assuming PNUM to have TYPE 'character', using 'P1' is correct. If instead of 'P1' we had for example, 200, a TYPE error would occur.
- Literals may have a length LESS THAN the length of its associated attribute. A literal is truncated if it has a length greater than

the declared length of its associated attribute.
- Literals may contain any character. If a single quote is part of the literal string, write two quotes. E.G. ('DATE''S').

JOIN (5)

```
JOIN SPJ AND ('S1','P1','J1',200) GIVING TEMP.
```

The above is a combination of JOIN (2) and JOIN (4). There is no explicit over-list and there is a constant relation. In this case, the over-list is determined by the first relation; SPJ. All attributes of SPJ are declared to be in the implicit over-list. Furthermore, the order in which the attributes appear in SPJ is preserved in the over-list. This is important as the values in the constant relation are ultimately associated with the attributes in the over-list (see JOIN (4) above). The query, therefore, is internally generated to have the form:

```
JOIN SPJ AND ('S1','P1','J1',200) OVER SNUM,PNUM,JNUM,QTY GIVING TEMP.
```

JOIN (6)

```
JOIN(JOIN S AND J OVER CITY) AND SPJ OVER JNUM GIVING TEMP.
```

Nested queries are allowed. In the above query, (JOIN S AND J OVER CITY) will be evaluated first and the result stored in a temporary relation, say T00. The query then will have the form:

```
   JOIN T00 AND SPJ OVER JNUM GIVING TEMP.
```

The query can now proceed in the usual fashion.

Observations and Rules
- Structures of the form; (JOIN S AND J OVER CITY), are called 'nested relations'.
- Nested relations can appear anywhere an operand relation can (some exceptions noted later).
- Relations can be nested to any depth.
- Nested relations are enclosed in single brackets. Redundant brackets are NOT allowed.
- Nested relations NEVER have an explicit result relation declared.
- Nested relations can split over lines.
- See EXECUTION OF NESTED QUERIES for a detailed explanation on how a nested query is evaluated.

JOIN (7)

```
JOIN SUPLIER1.SPJ AND SUPLIER2.J OVER JNUM GIVING TEMP.
```

In a given run of RQL, a user may wish to access relations drawn from different data bases. RQL allows the inclusion of multiple data bases (See HOW TO RUN RQL for JCL details).

To distinguish between relations of the same name, prefix the relation name by the data base name in which the relation resides. Follow this by a dot (.). For example, if a relation 'J' existed in both data base SUPLIER1 and SUPLIER2, entering SUPLIER1.J would access the 'J' relation in the SUPLIER1 data base.

Observations and Rules
- There is no limit to the number of data bases that can be included.
- If more than one data base is included and a relation name is not prefixed by a data base name, the data bases are scanned in order for the FIRST occurence of the given relation name. This is the relation used in the query(see HOW TO RUN RQL for clarification of the ordering of data bases).

JOIN (8)

```
JOIN SPJ AND S OVER SNUM GIVING SUPPLIER.SPJ.
```

Usually, a result relation is only needed as an interim result. This is the reason result relations are stored in a temporary work area. Sometimes, however, the user may wish to permanently add the result relation to the data base. This is simply done by prefixing the result relation with the DDNAME of the JCL card which defines the user's data base.

Observations and Rules
- When permanently storing relations on a data base, the DISP parameter on the JCL card defining the data base must be; DISP=OLD.

JOIN (9)

```
TEMP=SPJ * J | JNUM.
```

The above query is equivalent to :

```
JOIN SPJ AND J OVER JNUM GIVING TEMP.
```

For the regular user of RQL, shorthand operators are recommended.

1) The join operation is presented as a multiplication of two relations:
```
SPJ * J
```
2) The keyword OVER can be replaced by '|' .
```
SPJ * J | JNUM
```
3) The construct 'GIVING result relation' can be replaced by 'result relation=' :
```
TEMP=SPJ * J | JNUM.
```
4) Shorthand operators may be used in any of the forms of JOIN:
```
E.G.    TEMP=SPJ(RENAME QTY TO SPJ_QTY)*J.
```

JOIN (10)

```
JOIN SPJ& AND S& OVER SNUM GIVING TEMP.
```

In response to the above query, RQL would normally first sort SPJ by SNUM then sort S by SNUM (the sorting is done on copies of the relations). As SPJ and S are already sorted on SNUM, this is an unnecessary action.

Appending an ampersand(&) to a relation informs RQL that this is the case and thus prevents the sorting of that relation. As sorting is expensive in terms of time and cost, judicious use of the ampersand can result in sigificant savings(See SORTED ON for declaration of sort orders outside queries).

Observations and Rules

- Only use the ampersand(&) when you are ABSOLUTELY certain that the relation is already sorted by the relevant attribute(s). If an '&' is used with a relation not appropriately sorted, erroneous results will probably occur for the immediate query AND for all later queries using that result relation.
- If the over-list has more than one attribute, the relation(s) must be already sorted on those attributes IN THE SAME ORDER as they appear in the over-list. That is, if an over-list contained the attributes; PNUM,SNUM, a relation T& involved in query in which that over-list is used, must be already sorted on PNUM,SNUM. If T& was previously sorted on say, SNUM,PNUM, the query would produce erroneous results.
- The ampersand(&) can be put to particular good use when employing constant relations. As the constant relation is being created 'on the spot', the user can easily ensure that the values therein are sorted. An example of this is;

```
JOIN SPJ AND ('P1','J3')& over PNUM,JNUM GIVING TEMP.
```

DIVIDE

The DIVIDE implemented by RQL is as given in Date(see pages 116-117).

DIVIDE (1)

```
DIVIDE SPJ BY CON OVER QTY DETERMINING SNUM GIVING TEMP.
```

This form of the DIVIDE is as given in Date's book except for one refinement, the 'DETERMINING' keyword. In Date, the division is assumed to be done on a dividend which is a binary relation(only two attributes). However, in practice many dividends are not binary, they have more than two attributes(conceptually, of course, any relation may be thought of as binary). The DETERMINING keyword allows the user to specify which attribute(s) from the first operand relation will appear in the result relation.

Observations and Rules

- The list of attributes appearing as the object of the DETERMINING keyword is called the determine-list. In the query above, SNUM is the only attribute in the determine-list.
- Any number of attributes may appear in the determine-list.
  E.G. DIVIDE SPJ BY PQ OVER QTY DETERMINING JNUM,SNUM GIVING TEMP.
- The ordering of attributes in the determine-list IS important as the dividend will be first sorted on these attributes.
- All attributes in the determine-list MUST exist in the dividend.
- All attributes appearing in the over-list MUST exist in BOTH the dividend and divisor.
- Any attribute appearing in the determine-list MUST NOT appear in the over-list and vice versa.

DIVIDE (2)

```
DIVIDE SPJ BY CON OVER QTY GIVING TEMP.
```

This form of DIVIDE is exactly that as presented in Date. As can be seen, the determine-list is omitted. In this case, a determine-list is internally generated by RQL. The attributes of this determine-list are constructed by taking all the attribute names that exist in the dividend and then subtracting those attributes appearing in the over-list. For example, SPJ has the attributes; SNUM,PNUM,JNUM and QTY. The over-list has the attribute QTY. Therefore, the determine-list will be; SNUM,PNUM,JNUM. In other words the above query is treated by RQL as;

```
DIVIDE SPJ BY CON OVER QTY DETERMINING SNUM,PNUM,JNUM GIVING TEMP.
```

Note that the result of the subtraction of the over-list from the attributes of the dividend must NOT be null. This would occur if the dividend had only those attributes appearing in the over-list.

DIVIDE (3)

```
DIVIDE SPJ BY CON GIVING TEMP.
```

The over-list as well as the determine-list may be omitted. In this case, RQL first internally generates an over-list and from it a

determine-list(as in DIVIDE (2)). The over-list is obtained by taking the common attributes of the DIVIDEND and DIVISOR. In the query above, as CON has only one attribute, QTY, and SPJ also has QTY, the over-list becomes QTY. The resulting determine-list is thus SNUM,PNUM,JNUM.

DIVIDE (4)

```
DIVIDE SPJ BY CON DETERMINING JNUM GIVING TEMP.
```

The over-list may be omitted and the determine-list declared. This case is just a simple variation of DIVIDE (3). The over-list is determined as in DIVIDE (3) and the query is complete. In other words, the query is internally generated as;

```
DIVIDE SPJ BY CON OVER QTY DETERMINING JNUM GIVING TEMP.
```

DIVIDE (5)

```
DIVIDE SPJ BY ('P1',200) OVER PNUM,QTY DETERMINING SNUM GIVING TEMP.
```

Constant relations are allowed as the divisor in DIVIDE queries. See JOIN (4) for the rules for using constant relations.

N.B. When using a constant relation as the divisor, an over-list MUST be explicitly declared.

DIVIDE (6)

```
DIVIDE SPJ BY ('P1',200) OVER PNUM,QTY GIVING TEMP.
```

This is essentially the same as DIVIDE (2) except that a constant relation is used as the divisor.

DIVIDE (7) (8) (9)

See JOIN (6) for rules for nesting queries.
See JOIN (7) for rules for using multiple data bases.
See JOIN (8) for rules for permanently storing result relations.

DIVIDE (10)

```
TEMP=SPJ / CON | QTY @ SNUM.
```

The above query is equivalent to:

```
DIVIDE SPJ BY CON OVER QTY DETERMINING SNUM GIVING TEMP.
```

For the regular user of RQL, shorthand operators are recommended.

1) The divide operation is presented as a division of the first operand relation, the dividend, by the second operand relation, the divisor:
   ```
   SPJ / CON
   ```
2) The keyword OVER can be replaced by '|' :
   ```
   SPJ / CON | QTY
   ```
3) The keyword DETERMINING can be replaced by '@' :
   ```
   SPJ / CON | QTY @ SNUM
   ```
4) The construct 'GIVING result relation' can be replaced by

```
'result relation=' :
      TEMP=SPJ / CON | QTY @ SNUM.
```

5) Shorthand operators may be used in any of the forms of DIVIDE:
E.G. TEMP=SPJ/(200)| QTY.

DIVIDE (11)

```
DIVIDE SPJ& BY ('S2')& OVER SNUM DETERMINING PNUM GIVING TEMP.
```

To use the ampersand with the dividend, the dividend must be already sorted on the attributes in the determine-list AND the attributes in the over-list IN THAT ORDER. The divisor must be sorted on the attributes in the over-list only.

See JOIN (10) for general rules and observations on the use of the ampersand to prevent sorting.

SELECT

The SELECT implemented by RQL is as described in Date.

SELECT (1)

```
SELECT J WHERE CITY='ATHENS' GIVING TEMP.
```

Any logical expression may appear in the where-list. The list of comparison operators includes:

=	or	EQ	¬=	or	NE	<	or	LT	>	or	GT
<=	or	LE	>=	or	GE	¬<	or	NL	¬>	or	NG

The 'where-list' may have several conditions:

```
SELECT SPJ WHERE  SNUM='S5' AND (PNUM>'P3' OR QTY NE  200) AND SNUM<'S2'
GIVING TEMP.
```

Observations and Rules

- Literals are enclosed in quotes, numeric constants are not.
- There is no limit to the complexity of the where-list.
- Brackets should be used for clarification and/or to enforce a desired sequence of evaluation.
- Redundant brackets ARE allowed.
- When using alphabetic comparison operators(e.g. NE), it must be surrounded by blanks. E.G. CITY NE 'PARIS'.
- The relation on which the SELECT is performed is NOT sorted. Therefore the order of the tuples in the result relation will reflect the tuple ordering of the operand relation.
- The presence of where-list attributes in the operand relation is not verified(see SELECT (3) for details).

SELECT (2)

```
SELECT J WHERE CITY =: 'A' GIVING TEMP.
```

Notice the colon(:) after the equal(=) sign. Normally when two strings are of unequal length in a comparison, the shorter string is internally temporarily lengthened to the length of the longer string. However, using a colon after the comparison operator does the reverse; the longer string is temporarily truncated to the length of the shorter string. The above query, for example, will find all tuples with CITY values beginning with 'A'.

SELECT (3)

The where-list can be considerably more complex than that presented in SELECT (1) and (2).

RQL translates the WHERE keyword to the SAS keyword IF and then outputs the where-list as the object of the IF keyword. No translation processing of the where-list occurs. This means that the where-list can be a SAS IF statement. For example, the following is valid;

```
SELECT SPJ WHERE SNUM='S2' THEN QTY=10 ELSE DO IF SNUM='S3' THEN
QTY=20 ELSE QTY=QTY * .25 END; GIVING TEMP.
```

Notice that using this flexibility of the where-list allows an ad hoc UPDATE capability.

An unfortunate offshoot of permitting the inclusion of a SAS IF statement, is that no error checking on the where-list is done. Most notably, the presence of where-list attributes in the operand relation is not verified and unmatched quotes for literals are not detected. Errors of this kind, however, will cause the SAS program, which ultimately answers the user's query, to produce a compilation error. The user is urged to be aware of this nuance for the SELECT query.

SELECT (4) (5) (6) (7)

See JOIN (6) for rules for nesting queries.
See JOIN (7) for rules for using multiple data bases.
See JOIN (8) for rules for permanently storing result relations.

Note that use of the ampersand to prevent sorting the operand relation is NOT valid with SELECT.

SELECT (8)

```
TEMP=SPJ : (PNUM LT 'P3' OR QTY >=500).
```

The above is equivalent to:

```
SELECT SPJ WHERE (PNUM LT 'P3' OR QTY >=500) GIVING TEMP.
```

1) The keyword WHERE can be replaced by ':' :
```
    SELECT SPJ : (PNUM LT 'P3')
```
2) The keyword SELECT can be omitted.
```
    SPJ : (PNUM LT 'P3')
```
3) The construct 'GIVING result relation' can be replaced by 'result relation=' :
```
    TEMP=SPJ : (PNUM LT 'P3').
```

Other rules for shorthand notation are as described in JOIN (9).

PROJECT

PROJECT is implemented in RQL just as described in Date. The result relation contains the attributes projected over the operand relation. All tuple duplicates are deleted in the process.

PROJECT (1)

```
PROJECT SPJ OVER QTY,JNUM GIVING TEMP.
```

TEMP will contain QTY and JNUM, in that order, with all duplicate tuples removed.

Observations and Rules
- All the attributes in the over-list must be found in the operand relation.
- The order of the attributes in the result relation will be the same order as the attributes in the over-list.

PROJECT (2)

```
PROJECT SPJ OVER NOT QTY GIVING TEMP.
```

Prefixing the over-list with the keyword 'NOT' has the effect of creating an over-list which contains all SPJ's attributes except QTY. That is, the above query is internally translated to the equivalent form;

```
PROJECT SPJ OVER SNUM,PNUM,JNUM GIVING TEMP.
```

This feature is useful when one is dropping fewer attributes than one is keeping. Note that the '¬' symbol may be used instead of the keyword 'NOT'.

E.G. `PROJECT SPJ OVER ¬ QTY GIVING TEMP.`

PROJECT (3)

```
PROJECT SPJ OVER QTY,JNUM WHERE SNUM='S3' AND QTY>100 GIVING TEMP.
```

Notice the inclusion of the where-list. It has the effect of performing a SELECT operation on SPJ. The PROJECT is then executed on the result. The inclusion of a where-list in a PROJECT query simplifies and clarifies overall query presentation. Nothing is saved in actual execution however, as it is implemented by a SELECT followed by a PROJECT (later versions of RQL may optimize this).

PROJECT (4) (5) (6) (7)

See JOIN (6) for rules for nesting queries.
See JOIN (7) for rules using multiple data bases.
See JOIN (8) for rules for permanently storing result relations.
See JOIN (10) for rules for preventing the sorting of relations.

PROJECT (8)

```
TEMP=SPJ | PNUM,SNUM.
```

The above query is equivalent to:

```
PROJECT SPJ OVER PNUM,SNUM GIVING TEMP.
```

1) The keyword OVER can be replaced by '|' :
   ```
   PROJECT SPJ | PNUM,SNUM
   ```
2) The keyword PROJECT can be omitted.
   ```
   SPJ | PNUM,SNUM
   ```

   Other rules for shorthand notation are as described in JOIN (9).

MINUS

This is the traditional set operation except that the two operand relations do not have to be union compatible. Not only can same-named attributes be of different type (see JOIN (1) for explanation), but also the order and the number of attributes in the the relations may differ.

MINUS (1)

```
SPJ MINUS PQ OVER JNUM,QTY GIVING TEMP.
```

TEMP will contain all tuples of SPJ except those tuples that have JNUM and QTY values found in PQ.

Say PQ had a tuple where JNUM = 'J4' and QTY = 200. Then any tuple in SPJ that had JNUM = 'J4' and QTY = 200, would not appear in TEMP. Both values of JNUM and QTY in PQ must be found in SPJ for that tuple NOT to be included in TEMP.

Observations and Rules
- All attributes in the over-list must be found in the second relation
- The over-list need not contain attributes found in the first relation. In other words, this query is valid;

```
SPJ MINUS S OVER SNUM,SNAME,STATUS GIVING TEMP.
```

Although SNUM is found in both operand relations, SNAME and STATUS are only found in S. The reason this is a valid query is that the over-list is used as only a guide for the MINUS operation. What actually happens is that all the attributes of SPJ are compared with the attributes in the over-list. Only those attributes that are in common are used in the MINUS operation. These common attributes form the 'minus-list'. In the above example, therefore, the minus-list would contain only SNUM. Ultimately this means that any tuple in SPJ having a value of SNUM equal to an SNUM value in S, will not be included in TEMP.

MINUS (2)

```
SPJ MINUS S GIVING TEMP.
```

The over-list may be omitted. If so, it is internally defined to be all the attributes found in the second relation. The above query is therefore equivalent to;

```
SPJ MINUS S OVER SNUM,SNAME,STATUS,CITY GIVING TEMP.
```

MINUS (3)

```
SPJ MINUS ('S3',100,'S4',100) OVER SNUM,QTY GIVING TEMP.
```

The second operand relation may be a constant relation. See JOIN (4) for the general rules that apply to the use of constant relations.

Observations and Rules
- As constant relations can be used in conjunction with an over-list, considerable flexibilty is available to update relations. For example, should the user wish to remove all tuples from SPJ having

an SNUM value of 'S3', the following could be given as the query;

SPJ MINUS ('S3') OVER SNUM GIVING SUPPLIER.SPJ.

(Note that when updating a result relation stored on the data base, it must be prefixed by the data base name).

- When using a constant relation, it is often preferable to rework the query to a SELECT operation. The same results can be achieved and SELECT does not involve sorting the operand relation. As sorting is expensive in time and money, this alternative is preferable. The immediately above query should therefore be given as;

SELECT SPJ WHERE SNUM NE 'S3' GIVING SUPPLIER.SPJ.

The only disadvantage of using SELECT is that the result relation will not be sorted on the attributes contained in the over-list. If you don't care about that, use the SELECT instead of the MINUS.

MINUS (4)

SPJ MINUS ('S1','P3','J2',200) GIVING TEMP.

The over-list can be omitted when using a constant relation. The over-list is internally generated using all the attributes found in the first operand relation. The above query, therefore, is equivalent to;

SPJ MINUS ('S1','P3','J2',200) OVER SNUM,PNUM,JNUM,QTY GIVING TEMP.

Care should be used when using this form of the MINUS query. The values of the constant relation must exactly match the name and ordering of the attributes in the first operand relation.

MINUS (5) (6) (7) (8)

See JOIN (6) for rules for nesting queries.
See JOIN (7) for rules for using multiple data bases.
See JOIN (8) for rules for permanently storing result relations.
See JOIN (10) for rules for preventing sorting of relations.

MINUS (9)

TEMP=SPJ - ('S2') | SNUM.

As with the other queries, shorthand operators are available.

1) The minus operation is presented as a subtraction of the two operand relations:
   SPJ - ('S2')

See JOIN (9) for other general rules regarding the use of shorthand operators.

UNION

This is the traditional set operation between two union compatible relations except that same-named attributes may be of different type and attributes may be ordered differently in the two operand relations(see JOIN (1) for further explanation).

UNION (1)

```
SPJ UNION TEMP GIVING TEMP2.
```

Observations and Rules

- Note that there is no over-list. All attributes of the first and second relation take part in the union. All attributes of the second relation must be found in the first relation and vice versa.
- The order of the attributes in the result relation is that of the attribute order in the first operand relation.

UNION (2)

```
SPJ UNION ('S1','P1','J4',200) GIVING TEMP.
```

Constant relations can be used as the second operand relation. Care should be used when using this form of the UNION operation. See JOIN (4) for observations and rules.

UNION (3) (4) (5) (6)

See JOIN (6) for rules for nesting queries.
See JOIN (7) for rules for using multiple data bases.
See JOIN (8) for rules for creating multiple relations.
See JOIN (10) for rules for preventing the sorting of relations.

UNION (7)

```
SUPPLIER.SPJ=SPJ + ('S3','P4','J9',100).
```

As with other queries, shorthand operators are allowed.

1) The union operation is presented as the addition of two relations:
   SPJ + ('S3','P4','J9',100)

See JOIN (9) for general rules for the use of shorthand operators.

TIMES

The TIMES implemented in RQL is as presented in Date(page 114). TIMES is used to obtain the extended cartesian product of two relations.

TIMES should be used only when absolutely necessary as it involves execution cost of the order M * N; where M and N are the number of tuples in the first and second operand relation respectively.

TIMES (1)

S TIMES P GIVING TEMP.

The result relation, TEMP, will have the attributes;

SNUM,SNAME,STATUS,CITY,PNUM,PNAME,COLOR,WEIGHT

The sort order of the attribute values in S will be inherited by TEMP.

TIMES (2)

SPJ(RENAME SNUM TO SPJ_SNUM) TIMES S GIVING TEMP.

To preserve the identity of those attributes that the operand relations have in common, the RENAME feature is available. Rules for its use can be found in JOIN (3) (statements regarding the over-list should be ignored).

TIMES (3) (4)

See JOIN (6) for rules for nesting queries.
See JOIN (8) for rules for permanently storing result relations.

TIMES (5)

TEMP=S TIMES P.

The only shorthand notation available is replacing the 'GIVING result relation' construct with 'result relation='.

PRINT

SAS has sophisticated output facilities. I have included a PRINT query for the sake of completeness but be aware that it does not include all the options of the SAS PRINT.

IMPORTANT: A relation having no tuples will NOT be printed. In fact, absolutely nothing is printed if a relation has no tuples. Even if a title is given, still there is no output. As the result of a query can easily be an empty result relation, don't think there is necessarily a bug in your program if there is no printed output.

PRINT (1)

```
PRINT SPJ.
```

SPJ will be printed.

PRINT (2)

```
PRINT SPJ 'THIS IS THE RELATION SPJ'.
```

Optionally, a title may be given. The title will appear at the top of every output page.

Observations and Rules
- The title must be the last component of the PRINT query.
- A title may NOT be broken across a line.

PRINT (3)

```
PRINT SPJ OVER QTY,PNUM 'THIS IS THE RELATION SPJ WITH NO SNUM OR JNUM.'
```

An over-list may be included in the PRINT query. The output will then contain only those attributes of the operand relation named in the over-list. Furthermore, the printed order of the attributes will reflect the order of the attributes in the over-list.

PRINT (4)

```
PRINT.
```

The name of the relation to print may be omitted. In this case, the relation printed will be the LAST relation that was the result of a query. For example, in the following sequence:

```
JOIN SPJ AND S OVER SNUM GIVING TEMP1.

DIVIDE TEMP1 BY (200) OVER QTY DETERMINING SNUM GIVING TEMP2.

PRINT.
```

The relation printed would be TEMP2.

PRINT (5)

```
PRINT 'THE ANSWER TO EXERCISE 6.18'.
```

This is just like PRINT (4) except that a title is given.

PRINT (6)

```
PRINT(JOIN SPJ AND S OVER SNUM) 'USING PRINT WITH A NESTED RELATION.'
```

PRINT can have a nested relation(s) as its operand relation. This is useful when the result of a query is to be printed but not used in successive queries. The PRINT query itself, however, cannot be nested.

COPY

The user is able to make a copy of a relation. This is most useful when copying a relation from one data base to another.

COPY (1)

```
COPY SPJ GIVING NEW_SPJ.
```

As can be seen, the syntax is straightforward. The COPY can be expressed more simply as;

```
NEW_SPJ=SPJ.
```

When copying a relation which is to be permanently stored on a data base, the new relation must be prefixed with the data base name. An example of this could be;

```
HOUSES.NEW_SPJ=SPJ.
```

The COPY operation cannot be nested.

PURGE

PURGE (1)

A relation can be deleted by the following operation.

```
PURGE TEMP.
```

If the relation resides on the data base, it must be prefixed with the data base name.

```
PURGE SUPPLIER.SPJ.
```

SORTED ON

All RQL queries implementing the relational algebra involve the prior sorting of copies of the operand relations by the indicated attributes(SELECT query excepted). For example, in the following query;

```
JOIN SPJ AND S OVER SNUM GIVING TEMP.
```

copies of SPJ and S are sorted by SNUM as the first step of the join. As sorting is expensive in execution time, reducing the number of sorts required is a desirable goal. To this end, RQL maintains a sort-list for every relation. This sort-list contains a list of attributes by which a given relation is sorted. Before an operand relation is sorted in response to a query, its sort-list is checked against the list of attributes by which the relation is about to be sorted. If there is a match, the relation is not sorted. The result relation then 'inherits' a sort-list comprised of those attributes relevant to the particular query(e.g. the over-list in a PROJECT query). If not all attributes match, sorting of the operand relation proceeds in the usual fashion. The result relation then acquires a sort-list comprised of the attributes by which the operand relation was sorted.

There is a way to initialize the sort-list of a relation.

```
SPJ SORTED ON SNUM.
```

The above operation informs RQL that SPJ is sorted on SNUM. Thus in the query;

```
JOIN SPJ AND S OVER SNUM GIVING TEMP.
```

the sorting of the copy of SPJ would be prevented.

To clear the sort-list of a relation, declare no attributes. That is;

```
SPJ SORTED ON.
```

Normally the SORTED ON declarations would be placed at the beginning of an RQL program, although they could be inserted anywhere.

Don't confuse the SORTED ON declarations with the notion of keys. Although very often a relation is sorted on its key, it need not be. The declaration of the SORTED ON attributes is useful whatever the situation.

Note that the SORTED ON keywords can be replaced by an ampersand(&). For example;

```
SPJ & SNUM.
```

NEW_PREFIX and NEW_SUFFIX

RQL works by translating the user's query into a SAS source program which is then executed to ultimately answer the query. The SAS program itself contains variables that it uses to contain temporary values. Unfortunately, these variable names may conflict with attribute names in the user's relations. To lessen the chance of this happening, all variable names and temporary relations generated by RQL start and end with an underscore. If this still does not prevent a conflict with the user's names, there is a way to change the prefix and/or suffix of RQL names.

NEW PREFIX=B.

The above will change the first character of all variable and temporary relation names to 'B'.

NEW SUFFIX=C.

The above will change the last character of all variable and temporary relation names to 'C'.

The following is a list of the variables and temporary relation names used by RQL.

All temporary relation names of the form _T00nnn_ where nnn is a sequence number.

The variables are;

MCHFND _FSTNGP_ _PRVMCH_ _NA_ _NB_ _NATOTL_ _NBTOTL_
FSTPSN_ _LSTPSN_ _ALLEQL_ _INCRD1_ _FSTOBS_ _COUNT_
NEXT _ENDA_ _ENDB_

and all variables of the form;
S1Jnnn where nnn is a sequence number.

Note that NEW PREFIX and NEW SUFFIX can be used repeatedly at the user's discretion.

COMMENTS

To include single line comments in an BQL program, enter an asterisk(*) as the first character on a new line. For example,

```
*  THIS IS A CCMMENT
```

To include multi-line comments, 'bracket' them with the character pair, '(*' and '*)'. For example;

```
(*   THIS IS AN EXAMPLE OF A COMMENT THAT OCCUPIES MORE THAN
     THAN ONE LINE.     *)
```

IMPORTANT: Ccmments can not be imbedded within a query. For example, the following is illegal;

```
PROJECT(JOIN SPJ AND S OVER SNUM)
* IMBEDDING A COMMENT WITHIN A NESTED QUERY IS ILLEGAL
      OVER SNUM,SNAME,QTY.
```

AUTHORIZED USERS OF RQL

To provide some measure of security for users' data bases, RQL maintains a directory of authorized users. This directory is comprised of a list of users' OS codes currently permitted to use RQL. An OS code, not in the directory, attempting to use RQL, will result in an RQL abort.

Owners of OS codes wishing to be placed in the RQL directory, should contact the data base administrator at the relevant location.

Note that for present developmental purposes, all OS codes can use RQL.

HOW TO RUN RQL

The JCL cards required to run RQL are the following:

```
(1)  //   JOBCARD
(2)  //   EXEC RQL,DBASE=SUPPLIER
(3)  //STEP1.SUPPLIER  DD DSN=ucod.$.SUPPLIER,DISP=OLD
(4)  //QUERY.SYSIN   DD  *

(5)       Your queries.

(6)  /*
(7)  //STEP3.SUPPLIER  DD  DSN=ucod.$.SUPPLIER,DISP=OLD
```

Card (1) is your jobcard. The only thing here of interest is the JOB CLASS. It could be declared as low as 2, but a JOB CLASS of 3 or 4 results in greater program execution efficiency.

Card (2) invokes the RQL catalogued procedure. The DBASE=SUPPLIER option tells RQL that the name you are giving to your data base is SUPPLIER. If you had more than one data base, you would enter them here. For example, say you wanted to include three data bases in a given session; SUPPLIER,CARS, and HOUSES. You would then enter; DBASE='SUPPLIER,CARS,HOUSES'. Notice that multiple data bases must be enclosed in quotes.

The order that the data bases appear on the EXEC card is important. When relations in a query are not prefixed with the name of a data base, the data bases are searched for that relation name in the order in which they appear on the EXEC card.

The data bases given on the EXEC card must each contain at least one SAS data set. If not, the job is terminated.

Several options may be given on the EXEC card.

A listing of the compiled SAS program source which ultimately answers the query can be obtained by declaring 'SOURCE=SOURCE' on the EXEC card;

```
//   EXEC RQL,DBASE=SUPPLIER,SOURCE=SOURCE
```

When the source is desired, the NOTES that SAS outputs as part of the LOG can also be obtained;

```
//   EXEC RQL,DBASE=SUPPLIER,SOURCE=SOURCE,NOTES=NOTES
```

The answers to the querys are output with a default line length of 132 characters. This output length can be changed by declaring a line length ranging from 64 to 255 using the construct LS=. An output line length of 80 characters, for example, can be obtained as follows;

```
//   EXEC RQL,DBASE=SUPPLIER,LS=80
```

SAS provides for the selection of a variety of other system options. The reader is referred to the SAS manual for their details. To declare any of these other options, enter;

```
OPTIONS='list of options'
```

on the EXEC RQL card. Note that the list of options must be enclosed in quotes.

Those options declared with the OPTIONS= feature take precedence over any other options declared. If the user enters the following, for example;

```
//   EXEC RQL,DBASE=SUPPLIER,LS=80,OPTIONS='LS=132'
```

the output line size will be 132 characters.

Card (3) defines the OS data set on which the data base resides. If either new relations are to be added to the data base or old relations modified, the DISP parameter must equal OLD; DISP=OLD. If only retrieval queries are to be executed, the DISP parameter should be SHR; DISP=SHR.

The ddname of card (3) must be of the form; STEP1.databasename. In this example, the data base name is SUPPLIER, so the ddnname reads; STEP1.SUPPLIER. There must be one such card for every data base on the EXEC card. If there were three data bases as above, you would have the following:

```
//STEP1.SUPPLIER  DD  DSN=----------
//STEP1.CARS      DD  DSN=----------
//STEP1.HOUSES    DD  DSN=----------
```

The ordering of these cards is unimportant.

Card (4) precedes the actual queries.

Card (6) follows the actual queries.

Card (7) is an exact copy of card (3). If there are multiple data bases, all cards of type (3) must be copied here. Note that if a data base will be accessed often enough and by enough people, cards of type (3) and (7) can be included in the RQL catalogued procedure. This would simplify the JCL setup cards considerably.

EXECUTION OF NESTED QUERIES

The following example will attempt to illustrate the sequence of nested query execution.

```
JOIN(PROJECT(SELECT S WHERE CITY='LONDON') OVER SNUM,CITY) AND
(SELECT J WHERE CITY='LONDON') OVER CITY GIVING TEMP.
```

The first nested relation encountered is;

```
(SELECT S WHERE CITY='LONDON')
```

The result of evaluating this nested relation is stored in a interim relation of the form; _T00nnn_; where nnn is a sequence number. Therefore, the original query now looks like;

```
JOIN(PROJECT _T001_ OVER SNUM,CITY) AND (SELECT J WHERE CITY=
'LONDON') OVER CITY GIVING TEMP.
```

The next nested relation encountered is;

```
(PROJECT _T001_ OVER SNUM, CITY)
```

The result is stored in _T002_ and so the original query now looks like;

```
JOIN _T002_ AND (SELECT J WHERE CITY='LONDON') OVER CITY GIVING
TEMP.
```

The next(and last) nested relation encountered is;

```
(SELECT J WHERE CITY='LONDON')
```

The result is stored in _T003_ and so the original query is;

```
JOIN _T002_ AND _T003_ OVER CITY GIVING TEMP.
```

The query execution now proceeds in the regular fashion.

There are two reasons why this explanation has been presented. First, to illustrate the mechanism of evaluation and second, to warn the user of potential error messages.

When a nested query is in error, RQL writes relevant error messages. For example;

```
JOIN(_T002_ AND _T003_ OVER CITY GIVING TEMP.

***ERROR**** THE ABOVE IS SYNTACTICALLY INCORRECT.
IT HAS NOT BEEN RECOGNIZED AS A VALID STATEMENT.
```

(Note that the above query has a superfluous left bracket immediately following the keyword JOIN).

The above query looks quite different to the user's original query as the interim result relations have replaced the nested relations. The user who is unaware of how nested queries are executed would be quite

perplexed as to how the query was transformed. Hopefully, as a result of reading this section, the user will be able to proceed on an informed basis.

REGULAR USERS OF SAS

Users familiar with SAS have a distinct advantage in RQL. Potential error messages will be better understood and result relations manipulated with greater power and ease.

USING SAS STATEMENTS IN AN RQL PROGRAM

Regular SAS statements can be imbedded in an RQL program. To include regular SAS statements, enter a '$' as the first character on a new line. For example;

```
$   OPTIONS NOSOURCE;
```

To include a block of SAS statements, 'bracket' them with the character pairs, '($' and '$)'. For example;

```
($  DATA NEWSET;
     SET NEW_SPJ;
         IF FIRST.SNUM;     $)
```

Note that when using the SAS comments constructs; '/*' and '*/', leave at least one blank between '($' and '/*'. For example;

```
($ /* THIS IS A TEST OF THE SAS COMMENTS CONSTRUCTS.
      THIS IS A TEST OF THE SAS COMMENTS CONSTRUCTS.  */ $)
```

RQL knows nothing of the actions in the SAS statements given by the user. For example, if in a given RQL program a user deletes a relation using SAS, RQL will still assume it exists. Or if, say, an attribute is dropped or renamed using SAS, RQL will still look for the old attribute. Therefore SAS statements which alter the essential structure of a relation should be placed at the very end of an RQL program. Statistical and report writing procedures are normally 'safe' and can be inserted anywhere.

ROUTING THE SAS SOURCE PROGRAM

The user's queries are translated to a SAS source program which is then executed to ultimately answer the queries.

It is possible to defer the execution of the SAS program by having it routed to an output data set of the user's choosing. This feature thus enables the user to save the SAS source for later modification and/or execution. To accomplish this, two alterations to the JCL cards of RQL are required. First, on the EXEC card, the following must be added; COND.STEP3=(0,LE). To illustrate;

```
//    EXEC    RQL,DBASE=SUPPLIER,COND.STEP3=(0,LE)
```

Second, immediately preceding the JCL card;

```
//QUERY.SYSIN    DD  *
```

the following card must be inserted;

```
//QUERY.FT14F001    DD  output data set
```

The above card defines the output data set to which the SAS source will be written(most typically it will be an OS data set).

It should be emphasized that routing the SAS program to a user defined data set precludes the execution of the SAS program.