AD-A108 526   INCO INC   MCLEAN VA   F/G 9/2

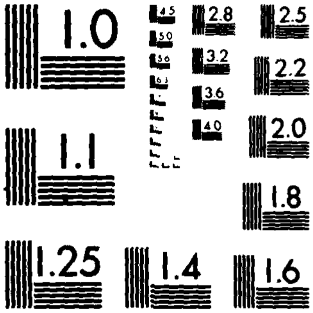
ORACLE VERSUS IDM COMPARATIVE ANALYSIS.(U)

NOV 81   N00014-81-C-0808

UNCLASSIFIED   NL

END

DATE
FILMED
[illegible]-82
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

INCO/UP-I-81-83155-U

ORACLE vs. IDM
Comparative Analysis

10 November 1981

Contract No. N00014-81-C-0808

Prepared for:

Office of Naval Research
Code 240
Attn: Joel Trimble
800 N. Quincy Street
Arlington, Virginia 22217

Prepared by:

INCO, INC.
8260 Greensboro Drive
McLean, Virginia 22102

TABLE OF CONTENTS

Page No.

SECTION 1. INTRODUCTION 1-1
1.1 Purpose of Comparative Analysis 1-1
1.2 Project References 1-2
1.3 Glossary 1-5

SECTION 2. ORACLE SQL DESCRIPTION 2-1
2.1 Query Statements 2-1
2.2 Data Definition Statements 2-6
2.3 Data Manipulation Statements 2-7
2.4 Security and Concurrency Control Statements 2-7

SECTION 3. UNIQUE SQL/DS FEATURES 3-1

SECTION 4. IDL DESCRIPTION 4-1
4.1 Query Statements 4-1
4.2 Data Definition Statements 4-3
4.3 Data Manipulation Statements 4-6
4.4 Security and Concurrency Control Statements 4-6

SECTION 5. IDM COMMAND RELATIONSHIP TO IDL 5-1
5.1 Query/Parse Tree 5-1
5.2 Query Statements 5-3
5.3 Data Definition Statements 5-7
5.4 Data Manipulation Statements 5-13
5.5 Security and Concurrency Control Statements 5-15

SECTION 6. CORRELATION BETWEEN SQL AND IDM FEATURES 6-1
6.1 Query Facilities 6-1
6.2 Data Manipulation Facilities 6-3
6.3 Data Definition Facilities 6-3
6.4 Security and Concurrency Control Facilities 6-4

SECTION 7. ADDITIONAL IDM FEATURES 7-1

SECTION 8. UNIMPLEMENTED FEATURES OF FULL SEQUEL 8-1

LIST OF FIGURES

Figure No.	Title	Page No.
5.1	IDM Parse Tree	5-2
5.2	IDL Parse Tree	5-2
5.3	IDM Target List	5-3
5.4	Query Tree With Qualification	5-4
5.5	IDM Retrieve Command Tree	5-6
5.6	IDM Retrieve Command Tree	5-6
5.7	Create View Query Tree	5-11

SECTION 1. INTRODUCTION

<u>1.1 Purpose of Comparative Analysis</u>. A comparison of the syntactic and semantic structures of the SEQUEL implemented in ORACLE (Version 2.3) and IDL (the Intelligent Database Language), the query language designed for the IDM 500, was conducted as the basis for building a Front-End to the IDM 500 Database Machine. Implementation of the transportable SEQUEL front-end with the IDM 500, which will support the ORACLE subset of SEQUEL, will provide the Navy IAIPS Program with a low-cost alternative for achieving a high performance relational database intelligence support environment.

The analysis focused on the features of each language, and on the features available with the IDM. This process was used to highlight the additional procedures necessary to create the language translation procedures that will be implemented by the SEQUEL/IDM Translation (SIT) component of the Front-End. Information on ORACLE SEQUEL 2.3 was derived from document review. Information about the IDM and Britton-Lee's query language, IDL, comes both from the literature and extensive hands-on use.

This document contains: a description of ORACLE SQL. Familiarity with the ORACLE manual is subsumed; SDL/DS features are described; a description of IDL and its relationship to the IDM commands; the correlation between SQL and the IDM features; additional IDM features not available within SQL; and unimplemented features of full SEQUEL.

1.2 Project References.

1. Automated Data Systems Documentation Standards. Department of Defense Instruction 7935.1-S,Sept.1977.

2. Britton-Lee, Inc. "DBMS In a Box." Los Gatos, California.

3. Britton-Lee, Inc. "IDM. The Intelligent Database Machine."

4. Britton-Lee, Inc. IDM 500 Intelligent Database Machine Product Description. Los Gatos, California, 1981.

5. Britton-Lee, Inc. Preliminary Performance Report - IDM 500.

6. Chamberlin, P.D. et al "SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control." IBM Journal of Research and Development Vol. 20 (Nov, 1976), pp.134-149.

7. Dieckmann, E. Martin. "Three Relational DBMS." Datamation, (September, 1981), pp. 137-148.

8. "Eight Fortune 500 Companies Offer Test Sites for New Britton-Lee Intelligent Database Machine." 3 pgs. Britton-Lee News Release - September 1980.

9. "English Merchant Banks Provide One Million Dollar Second Phase Financing for Britton-Lee." 2 pgs. Britton-Lee News Release - September 1980.

10. Epstein, Robert and Hawthorn, Paula. "Aid in the 80s." Datamation Magazine, 1980, pp. 154-158.

11. Epstein, Robert and Hawthorn, Paula. "Design Decisions for the Intelligent Database Machine." AFIPS-Conference Proceedings, Volume 49, 1980, pp. 237-241.

12. IBM. SQL/Data System General Information - Program Product. Program Numbers 5748-XX5. G1+24-5012-0/File No. S370-50.

13. IBM. "SQL/Data System Concepts and Facilities - Program Product." Program Number 5748-XX5. G1+24-5013-0/File No. S370-50.

14. IDM 500 Software Reference Manual. Britton-Lee, Inc., September 1981.

15. Inco. "Data Base Technology - A Concept and Recommendation." June 1981.

16. Krass, Peter and Wiener, Hesh. "The DBMS Market is Booming." Datamation. (September, 1981), pp. 153-170.

17. ORACLE User`s Guide. Version 2.3. Relational Software Incorporated. April, 1981.

18. Outline for SQL Front-End Functional Description. INCO: October, 1981.

19. "Processor Does Data Base Tasks," Electronics. April 24, 1980.

20. Project Work Plan/Comparative Analysis Outine. INCO: September, 1981.

21. "Technical Proposal for Development of a SEQUEL Front-End (Language Feature Specification)." Prepared for: Office of Naval Research; INCO: July, 1981.

22. Technical Memos

Memo # : 1184/01
Date : 14 July 1981
Originator: M. Kerchner
Subject : SEQUEL/IDL Translation

Memo # : 1184/02
Date : 14 July 1981
Originator: M. Kerchner

Subject   : SQL-IDL Command Translation

Memo #   : 1184/03
Date   : 22 July 1981
Originator: M. Kerchner
Subject   : SQL/IDM Implementation Requirements Definition

Memo #   : 1184A/04
Date   : 22 July 1981
Originator: Fred Friedman
Subject   : BNF Syntax for CDS-1000 SQL Queries

Memo #   : 1184/09
Date   : 14 August 1981
Originator: M. Kerchner
Subject   : High Level Design Review of BNF Syntax for CDS-1000 SQL Queries

Memo #   : 1184A/10
Date   : 13 August 1981
Originator: Fred Friedman
Subject   : CDS-1000 SQL Architecture

1.3 Glossary

ALL BUT
indicates that all privileges except those listed in the GRANT command are to be granted, or that all privileges except those listed in the REVOKE command are to be withdrawn.

ALL RIGHTS
indicates that all privileges are to be granted, or that all privileges are to be revoked.

AND
indicates the boolean operator AND. Boolian operators are used to connect predicates to form compound logical expressions.

ASC
indicates ascending sort order. If no sort direction is specified for a field, ascending is assumed.

AVG
specifies the arithmetic average of the values contained in the set of qualifying fields.

BEGIN TRANSACTION
identifies the start of a logical transaction consisting of one or more SQL statements. The BEGIN TRANSACTION must specify those tables (if any) being locked for UPDATE purposes, and those tables (if any) being locked for READ purposes.

BETWEEN
indicates the range comparison operator. The range is specified as a pair of constants, expressions, or columns connected by an AND.

CHAR
indicates the column is to contain alpha-numeric character string values.

column
specifies the name of a column defined in a table or view.

(column,...)
specifies the names of the columns of the table in the order the values will appear.

<column,...>
specifies a set of numeric or character-string literal values. The set is enclosed in angle-brackets < > and items within the set are separated by commas.

CONNECT BY
specifies the selection of rows according to their tree-structure relationship. The clause requires specification of the major and minor columns.

constant
specifies any numeric or character-string constant literal value that is to be inserted into the database in the

specified column. Single quotation marks are required around all character-string constants to distinguish them from column names.

COUNT — specifies the count of the set of all fields or rows qualified by the WHERE clause. COUNT indicates null fields in its total.

CREATE TABLE — defines a new table that is to be physically stored in the database. A table may contain from 1 to 255 columns. The CREATE TABLE specifies the name of the table, the names of the columns, and the column data types.

DEFINE USER — adds an authorized user to a secure ORACLE database. Only defined users are permitted to log on to a secure database.

DEFINE VIEW — creates an alternative view of data stored in tables in the database. The DEFINE VIEW statement names the view and optionally names its columns. A view may be defined in terms of other views. Views may be queried in the same way as stored tables; however, DELETE, UPDATE, and INSERT clauses may "not" reference views.

DELETE

DELETE clause — specifies the name of the table containing a row or set of rows that are to be removed from the database. The specific rows that are to be deleted are qualified by a WHERE clause.

DESC — indicates descending sort order.

DROP statement — removes tables or views from the system. Once a system entity has been dropped, its name may be reused. A table cannot be dropped if the table contains data. A table or view cannot be dropped if another view is defined upon it.

END TRANSACTION — is used to terminate a transaction that was started with a BEGIN TRANSACTION statement.

EXPAND TABLE — adds a new column to an existing table stored in the database. The new column is added to the right side of the table. Existing rows are considered to have null values in the new column until they are updated.

expression — specifies an arithmetic expression made up of columns and constants that are connected by the operators +, -, *, /. Parenthesis ( ) are used to establish precedence. Note that expressions involving a column value of NULL, will result in a null value unless the NULL Function parameter is used.

<expression,...> specifies a set of constant values or expressions.

FROM clause lists the tables and views that are referred to by the other clauses in the query block. A query block must contain a SELECT and FROM clause, and may optionally contain a WHERE, GROUP BY, or HAVING clause.

function indicates any of the SQL built-in group functions COUNT, SUM, AVG, MAX, MIN. The presence of a built-in function within a SELECT clause implies a GROUP BY. If the GROUP BY is not explicitly stated, the entire query result is treated as one group and each field in the SELECT clause must be a unique property of the group.

generic-constant specifies the leading character-string of a literal value. The leading string must be followed by the ellipsis notation "..." and the result must be enclosed in single quotation marks. Specification of a generic constant allows for a search on a leading character-string of a value.

GRANT allows the user who creates a table or view to give access privileges to other users. Those privileges are: READ, INSERT, DELETE, UPDATE, and EXPAND.

HAVING clause qualifies groups that are to be returned as the result of a query. Each field listed in the HAVING clause must be a unique property of the group.

GROUP BY is used to partition tables or views into groups according to the values in a column or a list of columns. A built-in set function is then applied to each group. A GROUP BY clause is always used with a built-in function.

IMAGE indicates that an index is to be maintained for the values in the column. Join operations can be performed only if both columns referenced in the joining predicate are defined as IMAGE.

IN indicates the set inclusion operator. IN tests a field for inclusion in a set of values. The comparison operator = may be used in place of IN.

INCLUDING is used with queries on tree-structured tables in conjunction with the WHERE clause to determine which rows are to be returned as a result of the query.

INCLUDING clause specifically contains any predicates or logical expressions that may be contained within the WHERE clause. INCLUDING is an optional clause used in conjunction with the CONNECT BY clause.

INSERT specifies the adding of a new row or set of rows into a table. Fields that are not present in the insertion statement are given null values.

len specifies the maximum length of a value to be stored in a character string field. The length must be a number from 1 to 255.

major-column specifies the name of the assembly column.

MAX specifies the maximum numeric value contained in the set of qualifying fields.

MIN specifies the minimum numeric value contained in the set of qualifying fields.

minor-column specifies the name of the component column.

NONULL indicates that null values are not permitted in the column.

NOT specifies that the following predicate or boolean expression is to be negated.

NULL indicates the absence of a value in the database. Null values are ignored in the evaluation of all arithmetic expressions, and the computation of all built-in functions except COUNT. NULL values are treated as unknowns in the evaluation of logical expressions (see Three Values Logic).

Null Value Function assigns a temporary value to null value encountered within an expression. The Null Value Function may be used in a SELECT, SET, or WHERE clause anywhere a column name may be used including within arithmetic expressions and built-in-functions.

NUMBER indicates the column is to contain numeric values. Numeric values are stored internally in base 256 format to maintain maximum precision.

OR indicates the boolean operator OR.

ORDER BY indicates the sequence that the query result is to be returned. The ORDER BY clause may contain a major and up to 254 minor sorting fields, with a maximum concatenated sort field of 255 characters.

password specifies the name of the password for the user being defined. The user must enter this password when logging on to an ORACLE database. The password can have a maximum length of 20 characters.

PRIOR

specifies the direction in which rows are to be selected. If the PRIOR keyword is placed before the minor (component) column, the query proceeds down the tree (explosion). If the PRIOR is placed before the major (assembly) column, the query proceeds up the tree (implosion).

privilege

specifies the type of operations that are to be authorized for the table. Privileges that have been granted by means of the GRANT command may be withdrawn through the use of the REVOKE command.

PUBLIC

all users of the database

READ

specifies that the table should be locked to update transactions. Read transactions may concurrently access the table.

SELECT...

specifies the use of the result of a query as a view on the database. Any vaid query bock can be used as a database view. The query blocks may be nested to any number of levels.

SET

specifies a column or list of columns to be modified within the table referenced by the UPDATE clause. A SET clause is always used in conjunction with an UPDATE clause. New values for fields that are to be updated may be state as constants or expressions.

SQL

a relational data language that provides a unified set of facilities for query, data manipulation, data definition, and data control. SQL is both a terminal interface for nonspecialists in data processing, and a data sublanguage embedded in host programming language for use by application programmers.

START WITH

specifies the rows that are to be used as starting points in queries on tree-structured tables. The START with clause may contain any predicate or logical expressions that may be contained within a WHERE clause. The START WITH clause is always used in conjunction with the CONNECT BY clause.

SUM

specifies the arithmetic sum of the values of qualifying fields.

table

specifies the name of a table or view that contains columns referenced by SELECT, WHERE, GROUP BY, HAVING, or ORDER

BY clauses.

table.* returns all the columns in the table or view specified. The * can be qualified with a table name when there are multiple tables and/or views listed in the FROM clause.

value specifies a temporary numeric value to be assigned to null values encountered during processing.

VAR indicates that the value stored in a character string field is to be stored in variable length format. Currently, ORACLE stores all character string values in variable length format whether or not VAR is specified.

view specifies the name of the view that is being defined. Table and view names must be unique within the database. The maximum length of the view name is 30 characters. The first character must be alphabetic.

WHERE space qualifies the rows that are to be returned as the result of a query. The WHERE clause may contain any combination of predicates that compare fields of rows to constant values, compare two fields of a row with each other, compare fields to expressions, etc.

WITH GRANT OPTION specifies that the grantee may grant the privileges listed to other users.

* returns all columns from all of the table(s) and view(s) specified in the FROM clause of the query block, and can also specify the count of all rows that satisfy the WHERE clause. The * may only be used with the COUNT function in the form: COUNT(*).

= indicates the equal comparison operator.

^= indicates the not equal comparison operator.

> indicates the greater than comparison operator.

>= indicates the greater than or equal comparison operator.

< indicates the lsss than comparison operator

<= indicates the less than or equal comparison operator.

table* specifies that the rows of the table listed in the from clause are to participate in the join if the join-column contains a null value. This is referred to as a "Outer-Join". An outer join table cannot be the first table listed in the FROM clause.

table.column specifies the name of a column qualified by the name of the table that contains the column. Qualified column names are used to eliminate ambiguity when the FROM clause lists multiple tables or views that contain duplicate column names.

table label specifies that the table or iew is to be renamed within the context of a query block. The renaming of a table with a label is necessary when the same table or view is listed more than once in the same FROM clause. This mechanism is used to join a table to itself. The temporary label is used in place of the table name to qualify columns referenced by the other clauses within the query block.

tran-id specifies an integer value. Tran-id must be specified when transactions are numbered in the BEGIN TRANSACTION statement.

UC indicates that the index to be maintained on this column is to have forward compression only. If UC is not specified, the index will have both forward and backward compression.

UNIQUE indicates that duplicate rows are to be eliminated from the query result in a WHERE clause or that no two fields within a column can have the same value if IMAGE has been specified.

UPDATE specifies that the table should be locked for all other update and read transactions.

UPDATE space specifies the name of the table containing a row or set of rows that are to be modified. A SET clause is used to specify the updates which are to be performed on the one or more columns within a row.

USER returns the name of the user (as specified in the DEFINE USER command) who is executing this SQL statement.

user-name specifies the name or identifier of the user being defined. The user must enter this name when logging on to an ORACLE database. The user-name can have a maximum length of 20 characters.

SECTION 2. ORACLE SQL DESCRIPTION

ORACLE SQL is a relational data language with facilities for query statements, data manipulation, data definition, and data control. SQL is based on SEQUEL which was originally developed by IBM as the main external interface for System R. Relational Software Incorporated (RSI) developed ORACLE incorporating SQL with a relational model of data. ORACLE SQL (hereafter referred to as SQL) was designed to increase productivity by producing a highly sympathetic user language, data independence and flexibility.

The format notation that follows conforms to the ORACLE SQL manual notation, as referenced on page 2-2 of the <u>Oracle User's Guide</u> - Version 2.3:

CAPITALIZED WORDS — identify words that have specific meanings in SQL.

lower case words — identify words that are names or labels to be specified by the user.

[ ] Square Brackets — are used to indicate that the enclosed word is optional and may be omitted.

| | Vertical Bars — enclosing vertically stacked items indicate that one of the enclosed items may be chosen.

... Ellipsis — indicates that the immediately preceding unit may occur once or any number of times in succession.

<u>2.1 Query Statements</u>. The basic SQL retrieval or query statement consists of one or more Query Blocks, and is of the form:

SELECT a_i (specifies what is to be returned as a result of the query block)

FROM relation r (specifies what tables and/or views are involved in the query)

The following optional clauses (detailed in the BNF) may be contained in the query: WHERE, GROUP BY, HAVING, CONNECT BY, START WITH, and INCLUDING. Values resulting from processing one Query Block can be referred to in the WHERE clause of another Query Block. This is accomplished by nesting Query Blocks withi a Query statement.

```
SELECT    $a_i$
FROM      $r_1$
WHERE     aj IN
 SELECT   $a_j$
 FROM     $r_2$
 WHERE    $a_k$ satisfy {set of boolean conditions}
```

Query Blocks can be nested at any level, and may be combined with other SQL predicates using boolean AND, OR and NOT. "SELECT ..." always denotes a nested Query Block.

The SELECT instruction specifies the return of columns from the table(s) and/or view(s) specified in the WHERE clause of the Query Block. The SELECT instruction may be modified by any of the following commands:

```
SELECT [UNIQUE] | *           | , |column       | , . . .
                |column       |   |table.column|
                |table.column|   |table.*      |
                |table.*      |   |expression   |
                |expression   |   |function     |
                |function     |   |user         |
```

Explanation of the SELECT modifiers appears in the BNF at the end of this section and in the User's Manual. Essentially the SELECT clause is used to request: "all columns; specific columns; results of arithmetic expressions or build-in functions; or a combination of columns, expressions and functions." Note that duplications are not eliminated unless SELECT UNIQUE is specified. UNIQUE is an option not a default, because extra processing is required to eliminate duplicate expressions.

The FROM clause is used to list the tables and views referred to in the other clauses of the query block. The query block will always contain a SELECT and FROM clause, and may contain a WHERE, GROUP BY, and HAVING clause. Table, table label, and table * are modifiers of the FROM clause that specify location or names of participating elements.

The WHERE clause is used to qualify the rows that are to be returned as the result of a query. Any grouping of predicates that compares fields of rows to constant values, or two fields of a row with each other, or compares fields to expressions may be contained in the WHERE clause. In SQL multiple predicates in the WHERE clause can be connected by AND or OR with square brackets [ ] to form logical expressions and establish procedure. Specifying NOT before a predicate negates the predicate on boolean expressions. Exclusion of a WHERE clause causes all rows in the specified table or view in the FROM clause to be returned.

In ORACLE five functions were built-in, as standard to the system.

```
|count|  |*           |
|sum  |  |column      |
|avg  |  |table.column|
|max  |
|min  |
```

These functions may be used in both the SELECT and HAVING clauses. If these functions are used within the SELECT clause, there need be no GROUP BY clause in the query block. The entire table is treated as one group. Here, only unique attributes of the group may be selected. The function, COUNT, may be applied to columns defined as CHAR in the CREATE TABLE. With the exception of the COUNT function, null values will not be included in a built-in function unless the NULL function parameter is used.

The Null Value Function assigns a temporary value to null values found within an expression. It can be used in a SELECT, SET or WHERE clause anywhere a column name may be used including within arithmetic expressions and built-in functions.

The GROUP BY clause partitions tables or views into groups according to the values in a column or a list of columns. Then, a built-in function is always applied to each group. "When a GROUP BY clause is used, or implied by the presence of a built-in function in the SELECT clause, each field in the SELECT clause must be a unique property of the group."

The HAVING clause delineates groups that are to be returned as the result of a query. Each field listed in this clause must be a unique property of the group. The HAVING clause will accept any combination of predicates in order to specify the appropriate groups. When there are both WHERE and HAVING clauses, the WHERE clause is to be applied first to qualify rows. The groups are then formed, and then the HAVING clause is applied, to qualify the groups. The following expressions, detailed in the User`s Guide, modify the HAVING clause.

```
|column       |  |=     |  |column          |  |AND|
|table.column|  |>=    |  |table.column    |  |OR |
|constant     |  |>     |  |constant        |
|NULL         |  |>=    |  |generic-constant|
|expression   |  |<     |  |NULL            |
|<column,...>|  |<=    |  |expression      |
|USER          |  |between|  |<column,...>     |
                  |IN     |  |<expression,...>|
                             |SELECT           |
                             |USER             |
```

CONNECT BY, in ORACLE is used to specify the selection of rows according to their tree-structure relationship. This clause requires specification of major and minor columns. The PRIOR keyword is positioned before the column to indicate the direction the rows are going on the tree.

The START WITH clause is used to specify the rows designated as starting points in queries on tree-structured tables. This clause may contain any predicate or logical expression that can be contained in a WHERE clause. START WITH is always used in conjunction with the CONNECT BY clause.

The INCLUDING clause is also used with queries on tree-structured tables in conjunction with the WHERE clause to determine the rows to be returned as the result of the query. Rows excluded because they fail to satisfy the WHERE clause cause exclusion of an entire branch of a tree structure. Rows excluded because they fail to satisfy an INCLUDING clause result only in that row being excluded. INCLUDING is an optional clause. It may combine any predicates or logical expressions that can be used in a WHERE clause. INCLUDING is used in conjunction with the CONNECT BY clause.

ORDER BY is an instruction that indicates the sequence in which the query result is to be returned. The ORDER BY clause is not a part of the query block, and may only be used next to the first query block of a SQL query statement.

2.2 Data Definition Statements. Data definition statements allow modification of data definitions in the ORACLE Data Dictionary. Use of these statements does not require reorganization activity. There are four basic statements in this category. They are: CREATE TABLE; EXPAND TABLE; DEFINE VIEW; and DROP.

The statement CREATE TABLE is used to define a new table to be inserted in the database. This statement specifies the name of the table, the names of the columns, and the column data types. In the CREATE TABLE statement null or duplicate values may be restricted and high performance access paths may be specified. A table can contain up to 255 columns. An index (IMAGE) is automatically maintained in the first column defined in the table. Sequential processing of rows in the table is aided by storage in physical sequence based on the index.

EXPAND TABLE adds a new column to a table that already exists in the database. New columns are added on the right side of the table. A query or a view written in terms of the base table (without addition) is not affected by the expansion. Existing rows are treated as null values in the new column until they are updated.

Alternative views of data stored in tables in the database can be created by use of the DEFINE VIEW statement. Any query formation can be used to define the view, or the view may be defined in terms of other views. The DEFINE VIEW statement will name the view and may (optionally) name its columns.

The DROP statement is used to eliminate tables or views from the system. A table cannot be dropped if it contains data. Neither a table nor a view can be dropped if another view is defined upon it. All applicable rows must be deleted before the DROP statement is used.

2.3 Data Manipulation Statements. Data manipulation statements provide for addition, deletion, or modification of column values or rows of a table. There are four SQL clauses designed for these functions. They are: INSERT INTO; DELETE; UPDATE; and SET.

INSERT INTO is used to add a new row or set of rows into a table. Fields without values are defined as null values. If all the fields are present in the correct order for the row, the list of column names may be omitted.

The DELETE instruction specifies the table name containing the row(s) to be removed from the database. The specific rows to be deleted are qualified by a WHERE clause. The WHERE clause in a DELETE instruction is identical to the WHERE clause of a query statement and may be contained in nested query blocks.

An UPDATE clause is used to name the table that contains the row(s) to be modified. The SET clause specifies the updates to be performed on the column(s) within a row. Here, too, the WHERE clause identifies the specific row(s) to be modified. An UPDATE statement may not be used to modify primary keys. The new values being updated can be stated as constants or expressions.

2.4 Security and Concurrency Control Statements . This section includes discussion of SQL Data Control Statements for Security and Concurrency Control. The following statements provide the framework for Security Control: DEFINE USER; GRANT; REVOKE; and PASSWORD. The concurrency control statements are: BEGIN TRANSACTION and END TRANSACTION.

DEFINE USER is the statement that allows an authorized user access to a secure database. Only authorized users can log on to a secure database. The user who builds the database is, until otherwise specified, the only authorized user. The DEFINE USER instruction allows new users to log on to the database, to add tables and to allow new users access to the database. It does not, however, allow access to stored data in the database without data access privileges which are given via the GRANT command.

The user who builds the table or view, controls access to it. The user may allow others to access the table or view through the GRANT command. Within this command the following privileges may be allowed: READ; INSERT; DELETE; UPDATE (by column); and EXPAND. Only the READ privilege may be granted for a view. Use of the WITH GRANT OPTION will allow additional users to grant privileges to other users.

The REVOKE statement withdraws privileges that have been allowed through the GRANT command. The privileges named are removed from "the grantee and from all users to whom he has granted them." All of the privileges that may be granted may be revoked.

A user`s password is redefined through use of the PASSWORD statement. The user can redefine only his own password.

Logical transactions consisting of one or more SQL statements use the BEGIN TRANSACTION statement to identify the start point. This statement specifies the tables (if any) being locked up for UPDATE purposes or for READ purposes.

The END TRANSACTION statement terminates the transaction that started with a BEGIN TRANSACTION statement.

SQL

BNF Syntax

```
sql-statement :: = query
                 | dml-statement
                 | ddl-statement
                 | control-statement
dml-statement :: = insertion
                 | deletion
                 | update
query :: = query-block [ ORDER BY ord-spec-list ]
insertion :: = INSERT INTO receiver : insert-spec
receiver :: = table-name [ ( field-name-list ) ]
field-name-list :: = field-name
                   | field-name-list , field name
insert-spec :: = query-block
               | lit-tuple
deletion :: = DELETE table-name [ where-clause ]
update :: = UPDATE table-name [ where-clause ]
            SET set-clause-list [ where-clause ]
where-clause :: = WHERE boolean
set-clause-list :: = set-clause
                   | set-clause-list , set-clause
set-clause :: = field-name = expr
query-block :: = select-clause
                   FROM from-list
                   [ WHERE boolean ]
                   [ GROUP BY field-spec-list ]
                   [ HAVING boolean ]
                   [ CONNECT BY [PRIOR] field-spec = field-spec]
                   [ START WITH boolean ]
                   [ INCLUDING boolean ]
select-clause :: = SELECT [ UNIQUE ] set-expr-list
                 | SELECT [ UNIQUE ] *
sel-expr-list :: = sel-expr
                 | sel-expr-list , sel-expr
sel-expr :: = expr
            | var-name . *
            | table-name . *
from-list :: = table-name [ var-name ]
             | from-list , table-name [ var-name ]
field-spec-list :: = field-spec
                   | field-spec-list , field-spec
ord-spec-list :: = field-spec [ direction ]
                 | expr
                 | ord-spec-list , field-spec [ direction ]
direction :: = ASC
             | DESC
```

```
boolean :: = boolean-term
           | boolean OR boolean-term
boolean-term :: = boolean-factor
                | boolean-term AND boolean factor
boolean-factor :: = [ NOT ] boolean primary
boolean-primary :: = predicate
                   | [ boolean ]
predicate :: = expr comparison expr
             | expr BETWEEN expr AND expr
             | expr comparison table-spec
             | < field-spec-list>  = table spec
             | < field-spec-list > [ IS ]  IN table-spec
table-spec :: = query-block
              | literal
expr :: = arith-term
        | expr add-op arith-term
arith-term :: = arith-factor
              | arith-term mult-op arith-factor
arith-factor :: = [ add-op ] primary
primary :: = field-spec
           | set-fn ( expr )
           | COUNT ( * )
           | NVL ( field-spec , constant )
           | constant
           | ( expr )
field-spec :: = field-name
              | table-name . field-name
              | var-name . field-name
comparison :: = comp-op
              | [ IS ] IN
comp-op :: = =
           | ^=
           | >
           | >=
           | <
           | <=
add-op :: = +
          | -
mult-op :: = *
           | /
set-fn :: = AVG
          | MAX
          | MIN
          | SUM
          | COUNT
```

```
literal :: = < lit-tuple-list >
           | lit-tuple
           | constant
lit-tuple-list :: = lit-tuple
                  | lit-tuple-list , lit-tuple
lit-tuple :: = < entry-list >
entry-list :: = entry
              | entry-list , entry
entry :: = [ constant ]
constant :: = quoted-string
            | number
            | NULL
table-name :: = name
image-name :: = name
name :: = identifier
field-name :: = identifier
var-name :: = identifier
integer :: = number
ddl-statement :: = create-table
                 | expand-table
                 | define-view
                 | drop
create-table :: = CREATE TABLE table-name ( field-defn-list )
field-defn-list :: = field-defn
                   | field-defn-list , field-defn
field-defn :: = field-name ( type [ , type-mod ] )
type :: = CHAR ( integer ) [ VAR ]
        | NUMBER
type-mod :: = NONULL
            | IMAGE [ image-mod ]
image-mod :: = UNIQUE
             | UC
expand-table :: = EXPAND TABLE table-name ADD COLUMN field-defn
define-view :: = DEFINE VIEW table-name
                 [ ( field-name-list ) ] AS query
drop :: = system-entity name
system-entity :: = TABLE
                 | VIEW
control-statement :: = define-user
                     | password-spec
                     | revoke
                     | begin-trans
                     | end-trans
define-user :: = DEFINE USER user-defn
user-defn :: = user-name/password
password-spec :: = PASSWORD password
```

```
grant : : = GRANT [ auth ] table-name TO user-list
         [ WITH GRANT OPTION ]
auth : : = ALL RIGHTS ON
        | operation-list ON
        | ALL BUT operation-list ON
user-list : : = user-name
             | user-list , user-name
             | PUBLIC
operation-list : : = operation
                  | operation-list , operation
operation : : = READ
             | INSERT
             | DELETE
             | UPDATE [ ( field-name-list ) ]
             | EXPAND
revoke : : = REVOKE [ auth ] table-name FROM user-list
begin trans : : = BEGIN TRANSACTION [ tran-number ]
                 ON TABLE  table-name  trans-type
tran-number : : = ( integer )
trans-type : : = UPDATE
              | READ
end trans : : = END TRANSACTION [ tran-number ]
```

This Section was extracted from the ORACLE USER'S GUIDE ,
pp. 2-51 - 2-54.

SECTION 3. UNIQUE SQL/DS FEATURES

SQL/Data System is being developed by IBM for use on the 370 series or 4300 computers under DOS/VSE. SQL/DS had a Beta test in August, 1981, but is not projected to be ready for commercial installation until February, 1982. SQL/DS offers significant flexibility in data definition and modification; high-level capabilities; and fairly simple user access facilities. This system is designed to provide ease in programming and use for both the user and the programmer.

User access to data is also easier in SQL/DS. IBM calls it "automatic navigation" which means that the user can access data by indicating what data he needs, rather than specifying how to find it. SQL/DS does not require that the user know how the data is stored. The user view of the database is two dimensional. The extract function that is built-in to SQL/DS allows it to copy data from a DL/I database into its tabular form.

This system has the capability to allow an application program to "accept and execute a user entered command at execution time, thereby providing for the possibility of program control of user queries." The host language preprocessor stores object code access codes modules, which are executed at run time by application programs, in the data dictionary. This feature eliminates the need for program recompilation when access paths are changed.

A further, significant feature of SQL/DS is its direct bridge capability. The DL/1 DOS/VSE extract facility, queues and executes requests at specified times for data from a DL/1 database using

VSE/POWER. The facility has a DL/1 database description capability in SQL/DS, a DL/1 extract component, and an SQL/DS load component in which the SQL/DS target relations have been defined.

Additional features of interest on this system include: control of free space with a parameter in the ACQUIRE DBSPACE command; that archiving may be done during regular operation; automatic roll-back; and a defined hierarchy of security authorizations.

The SQL/DS system when it is commonly available, will have several significant features. It is, however, important to note that it shares most of its capabilities with INGRES and ORACLE. As E. Martin Dieckmann noted in his article "Three Relational DBMS," "The three systems are striking in their similarities. They differ more in the degree to which they have implemented certain facilities and capabilities than in the array of facilities offered."

SECTION 4. IDL DESCRIPTION

IDL (Intelligent Database Language) is a general-purpose query language that translates easily into IDM-internal form developed by Britton-Lee, Inc. The intelligent terminal raises the query; it translates the user command to the IDM-internal form without the IDM ever seeing the original user-generated command. Several front-end systems are capable of taking a user-generated database command and translating it to the IDM-internal form. Hardware, software and data requirements should be used to select the command language suitable for translation application. IDL is used in this comparative analysis because it describes IDM commands easily.

The following symbols are used in IDL commands. They are extracted from the IDM 500 Software Reference Manual Version 1.3, September, 1981.

`(`, `) - Parentheses are necessary, and must appear literally in the command.

`[`, `]` - Anything included in square braces is optional.

`|` - A vertical bar indicates that a choice of words is presented.

`{`, `}` - Curly braces indicate that the word may appear 0 or more times.

`/*`, `*/` - Words between these symbols are explanatory comments.

`<`, `>` - Words in angle braces are meta-symbols.

All other words are key words and must appear literally.

4.1 Query Statements. In order to display data from relations present in the database, a range statement must first be provided. The range statement associates a variable name to a relation name. Most IDM commands require the range variable, not the actual relation name. Next the command retrieve and the names of the attributes to be found are listed. This is called the "target list." This list is qualified by an instruction (called a qualification or a where clause) that specifies which tuples to get the data from. Expressions that appear on the target list must be named so that the front-end program can display the name when the value is sent by the IDM. Expressions can appear in the target list and in the qualification.

Qualifications also determine which objects are affected by a command. They are boolean expressions of relational clauses. In fact, a relational clause may only appear in a qualification, where operands may be in any expression.

Aggregate functions are strong elements of IDL. They are designed to return a set of values. A scalar aggregate is an arithmetic expression that operates over one or more functions and returns a single value. In the IDM, the following are aggregate operators: MIN, MAX, COUNT, SUM, ADG, and ANY. "ANY" returns Ø if no tuples qualify; otherwise "ANY" returns 1. COUNT, SUM and AVG (average) may use the modifier unique. If that option is selected, only non-duplicated values of the expression will be included in the aggregate. The result of the aggregate must also be given a name, so that the answer (result) can be identified. Qualifications are written inside the parentheses next to the object of the aggregate. In this way, the qualification refers to the objects being operated upon, not to the entire query. This distinction allows considerable flexibility. An aggregate is always a self-contained query embedded inside another query.

In IDL, the group by operator is called the "by" clause. It is this clause that distinguishes the syntax between aggregate functions and simple aggregates. When the qualification appears outside the aggregate function parentheses it is not being used to evaluate the function. It is, instead, used to specify which answers to print out. The qualification is serving as a general where clause. When the by clause is global to the whole query, the names on the target list are the same as the names in the by clause. They are referring to the same tuple in the qualification list.

The "order by" clause is included, by the user, to specify the order of the data. Use of this clause is the only way to assure that the data will be returned in a specific order. Absence of an order by clause allows the IDM to return tuples in the order the IDM finds most efficient for processing.

4.2 Data Definition Statements. The command "create" is used to set up a relation in IDL. Basically, the command sets up an empty relation in the database currently open. Attribute types and maximum attribute size must be specified in the create statement. To create a new database, the command is "create database." This command sets up a database that is empty except for the system relations. If parameters are to be included they must be specified here, otherwise the IDM assumes no parameters are to be included, and will use its default values. "Demand" specifies the desired size of the database. The database will not be allowed to grow beyond the size specified.

"Retrieve into" is the command that creates a new relation from one or more old ones. This command causes the new relation to be filled with the data specified including any data conversion that has been specified. When the "retrieve into" command is finished executing, a copy of the new data is in the new relation and the old relation should be removed with the destroy command. This process redefines the data structures to meet the changing needs of the database.

There is a data dictionary built into the IDM's data management system. It was designed to enable users to interactively define the data schema, and to look up that schema once it has been defined. Three of the relations that perform the data dictionary functions are described further.

The "relation" relation holds a list of all the relations in the database identified by the IDM-assigned relation id (relid), relation names, relation owners, number of tuples, and other information needed by the IDM to process commands on the relation.

The "attribute" relation contains information on each attribute of each relation in the database such as: attribute name, type, relid, IDM-assigned attribute id (attid) and all other attribute information needed by the IDM.

The "descriptions" relation associates one or more descriptions with a relid/attid pair. If the attid is zero the description is associated only with the relid. Descriptions may be up to 255 characters long.

The relation and attribute relation are automatically updated when a new relation is added to a database. The user has the option to update the description relation to store information about the relation. When a relation is destroyed, the related tuples in the relation, attribute and description relations are automatically deleted.

Adding a column to a table in IDL requires that a new table be created. The procedure requires that a new table consisting of the old table plus the new column be formatted. The old table is then destroyed and the new table takes the name of the old table.

"Create view" is a command used to set up a virtual relation, one that is an unreal entity. The view is composed of parts of one or more relations (called close relations), or other views. Views may be preserved or destroyed just as relations are. They may also be updated if the update can unambiguously be applied to one of the base relations.

The "define" statement defines the following stored commands: retrieve, append, replace, delete, begin transaction and end transaction. In this command, a parameter can be used in any place a constant could be used. The "define program" statement is used in programs and is referenced with a 4-byte number used to refer to the stored command. Each "define program" is associated with a program name and held physically near other define program commands using the same program name.

The "destroy" command eliminates relations, files, views and stored commands. This command removes the entire object from the system. Its space is then freed for use within the current database. If there are views or stored commands dependent on a database due to be destroyed, they must be destroyed first. Only the owner or database administrator may destroy an object.

4.3 Data Manipulation Statements. Data may be inserted into relations through the "append" command. These commands cause the IDM to store the data in the specified relations. Basically, the command adds zero or more tuples to a relation or a view. The names and values of attributes must be specified at this juncture. An attribute with no assigned value is given a default value: blanks for character attributes and zero for numeric attributes.

The command "delete" is used to remove one or more tuples from a relation. Only a user with write permission may make deletions from a relation.

The command "replace" substitutes one or more attributes in zero or more tuples of a relation. The variable is located outside the target list since only one relation may be affected by a single replace command. "Replace" may access more than one relation to calculate what is to be updated and how it is to change.

4.4 Security and Concurrency Control Statements. This section contains the security and concurrency control statements used in IDL. The statements, permit and deny, provide the framework for security control. Begin transaction and end transaction are the fundamental concurrency control statements.

The command "permit" is a protection control command. It allows designated users access to a relation, view, file or stored command.

User names are recorded in the "users" relation by the Database Administrator. If no names are recorded, everyone may access the information. Read, write or "all" capabilities may be specified in the <protect mode> of relations, views or files. Execute must be specified for stored commands. Relations, views, files and stored commands default to no access allowed by anyone except the owner. The DBA may grant permission to use the create, create index and create database commands.

"Deny" is the command used to refuse access to users. Access may be denied to a relation, file, view or stored command by user names or group names. If no users are specified, the protection applies to everyone. Read and write apply to relations, views and files. "All" denies both read and write capability. A deny command has precedence over previous permit commands. Only the owner of an object or the DBA can deny access. The DBA may also deny rights to use the create, create index and create database commands.

The "begin transaction" command is used whenever multiple IDM commands are to be treated as a single transaction. The "end transaction" command is given whenever a set of commands that commenced with "begin transaction" is completed. This allows the user to make the results of the transaction known to the rest of the system.

SECTION 5. IDM COMMAND RELATIONSHIP TO IDL

The IDM 500 Database Machine does not have a machine language. It is programmed through a series of high level commands and interpretations of the results. Each command begins with a command token, an op-code. The last byte of the command is always the END OF COMMAND token which tells the IDM that the command is complete and that processing can begin. Parameters of commands are defined with other tokens. In this section of the report, Britton-Lee's nomenclature and architecture are followed explicitly. Document tokens are written in capital letters. Numbers to the right of the token are one-byte length specifiers. These numbers are sent after the token and are followed by as many bytes of data as are associated with the token. The examples that follow show both the IDL form and the resulting IDM command notation. Examples are used at several junctures to clarify formats. The symbols defined in the introduction to Section 3 are used in the command definitions.

5.1 Query/Parse Tree

The meta-symbol notation for a query tree is:

<query tree>:LT-list>= =><rootnode>L= =Lq-list>

where the components are defined as:

<rootnode>:ROOT<byte1><byte2)

where

<byte 1>=range no. for result variable

<byte 2>=status bits for unique/non-unique,

retrieve/retrieve into,

create/destroy index, permit/deny

This is the syntax for the IDM's internal-form command language. The language consists of all valid trees which correspond to IDM commands. After the command tree is constructed it is sent in postfix order to the IDM.

The process of translating the query is straightforward.

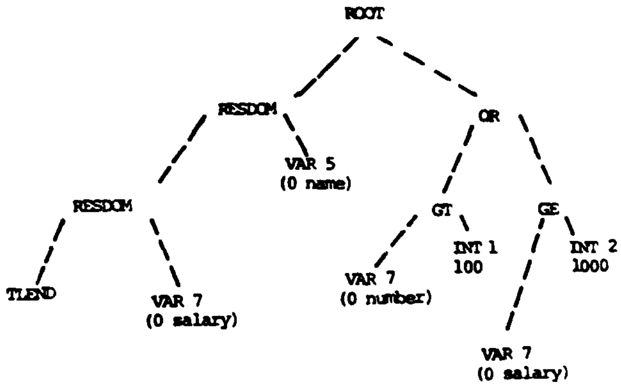
1. A user at a terminal types in a query.

2. The translating program puts the user command into IDM-internal form.

3. This parsing procedure evolves into a parse tree. The left side of the parse tree is the target list. The right side of the tree is the qualification. An example of a parse tree in IDM format follows. Its IDL equivalent (using the standard employee file example) is shown at the bottom of the page.

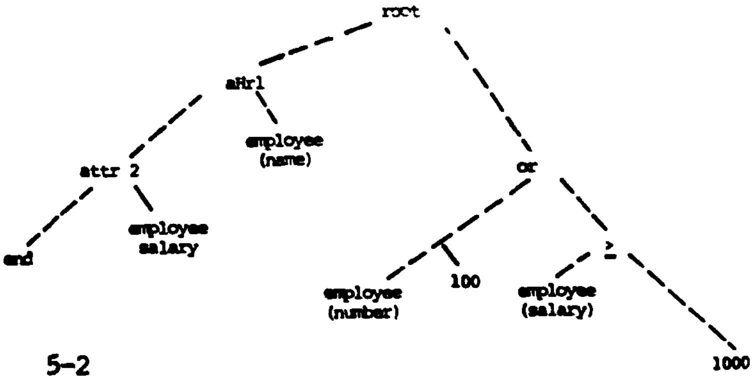
IDM

FIGURE 5.1 IDM PARSE TREE



IDL

FIGURE 5.2 IDL PARSE TREE



5.2 Query Statements

Query statements generally include target lists, qualification, range variables and expressions. Note, that for all of the following examples, trees are sent using a post order traversal, and that items in parentheses below the nodes are data associated with those nodes, and are sent after the token value and length.

The target list is the list of objects that are affected by the command. In a retrieve command the order of the elements on the target list will determine the order in which data is retrieved. In a query tree, the target list which is on the left of the ROOT node end with the TLEND (target list end) node.

<target list element>:

<name>=<expression> <attribute>

Target lists can be simple, as in the case above, where the target list was composed of a single relation variable and an attribute name. Arbitrary expressions, as defined below, can also be included in the target list. For example:

retrieve (e.name, wages = e.salary * e.hours)

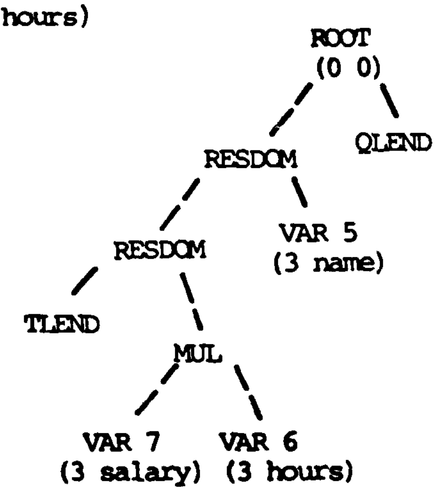


FIGURE 5.3 IDM TARGET LIST

In the above query, the target list is composed of the attribute "e.name" and the expression "e.salary * e.hours". The expression must be given a name in order to be displayed to the user; the name assigned in the above query is "wages".

The qualification is the part of the database command that specifies which objects are affected by the command.

```
<qualification>:                          (<qualification>)
        |      not <qualification>
        |      <qualification> and <qualification>
        |      <qualification> or <qualification>
        |      <clause>
```

A qualification is a boolean expression of relational clauses. Relational clauses may only appear in a qualification. Operands may be in any expression. For the example:

delete emp where emp.salary >24000

the following tree would be used.

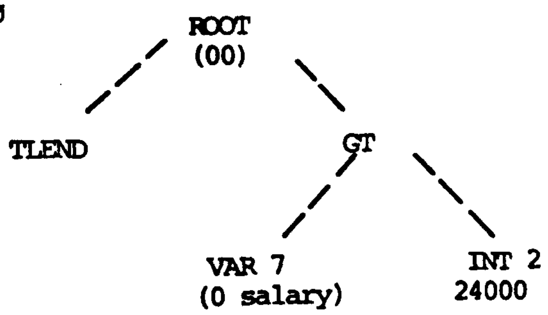


FIGURE 5.4 QUERY TREE
WITH QUALIFICATION

Expressions are supported by the IDM in both the target list and qualification for most commands. The following terms which are explained in great detail in the IDM manual are all operable expressions.

```
<expression>:                       <aggregate>
```

```
|    <attribute>
|    <constant>
|    <expression><arithop><expression>
|    -<expression>
|    (<expression>)
|    <constant function>
|    <unary function> (<expression>)
|    (with length function>(<intl>,<expression>)
|    <bmary function>(<expression>,<expression>)
|    <ternary finction>(<intl>,<intl>,<expression>)
```

Arithmetic operators are supported by the IDM only for integer and BCD expressions.

The range command associates a variable name with the name of a relation or view. Most IDM commands require the range variable, not the actual relation name. The IDM requires the statements on every command in which a variable is used. The IDL syntax for range is:

```
range of <variable> is <object name>
```

The retrieve command causes data to be sent to the host. This command can reference up to 15 relations, although they must all be in the same database. Use of an "order by" clause will specify the sort order of the returned data. The IDL syntax for retrieve is:

```
retrieve [unique] [into]<object name>]
          (<target list>)
          [order[by]<order-spec>[:ald]
          [,<order-spec>[:a|d]]
```

```
                    [where<qualification>]
```

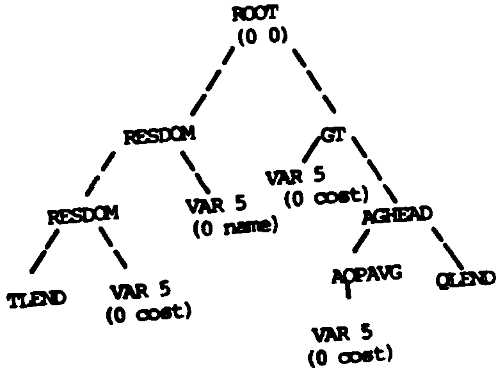
Given the example:

```
range of p is parts

retrieve (p.name,p.cost)
         order by cost:descending

         where p.cost>avg(p.cost)
```

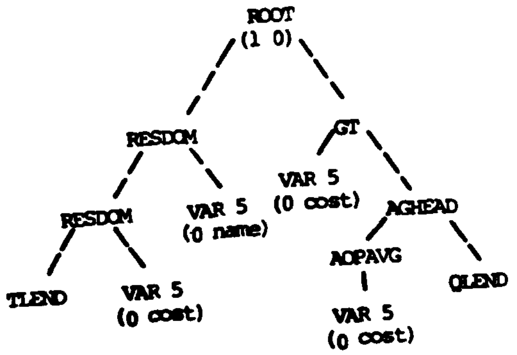
FIGURE 5.5 IDM RETRIEVE COMMAND TREE



RETRIEVE

RANGE 6 Ø parts

FIGURE 5.6 IDM RETRIEVE COMMAND TREE



RETRIEVE

RANGE 6 Ø parts

RANGE 1Ø 1 exp parts

The IDM command notation is extracted from the IDM Software Reference Manual Version 1.3, section 7.

```
RETRIEVE
RANGE <lenf> <rno> relnamel
  .
  .
  .
RANGE <lenf> <rno> relnamek
<rootnode>:  ROOT Ø <byte2>
                /* <byte2>= 1 if retrieve unique
                and <byte2>= Ø if retrieve */
<T-list>:    TLEND = =>
             || "<resattnode>
                | RESDOM
                | ORDERDOM " <= =<attnode>||
<Q-list>:    <Q-subtree>  |  QLEND
[ORDERA x] |  [ORDERED y]    /* where x and y are attribute **
           .                 ** numbers on the target list  **
           .                 ** of the query tree on which  **
           .                 ** the result is sorted
<options>
ENDOFCOMMAND
```

5.3 Data Definition Statements

This section includes IDL notation and the IDM commands for the operations create, create database, retrieve into, create view, define, and destroy.

The "create" command sets up an empty relation in the open database. Several optional specifications, such as size, updates and space may be selected. These options are detailed in the *IDM Software Reference Manual Version 1.3*. The IDL syntax is:

```
create<object name>(<name>=<format>{
 ,<name>=,<format>})[with<options>]
```

The IDM create command appears as follows:

```
CREATE
RANGE <lenf> <rno> <name>
<rootnode>:        ROOT <rno> Ø
<T-list>:          TLEND ||= => <resattnode> <= = <typenode>||
<Q-list>:          [<= = logspec] [<= =<qspec>]
                   <= =[<demandspec>] |QLEND
<demandspec>:      | <allocspec> <= =<dskspec>
                   | <dskspec>
<typenode>:        TYPE INT1 | TYPE INT2 | TYPE INT 4 |
                   TYPE FLT4 | TYPE FLT8|
                   TYPE CHAR len | TYPE FCHAR len |
                   TYPE BCD  len | TYPE FBCD  len |
                   TYPE BCDFLT len | TYPE FBCDFLT len |
                   TYPE BINARY len | TYPE FBINARY len
logspec:           QLEND = => WITH 4
<qspec>:           INT1 val | INT2 val | INT4 val| = => WITH 1
<allospec>:        INT1 val | INT2 val | INT4 val
```

```
                   = => WITH "2 | 5"
<dskspec>:         CHAR <lenf> <diskname> = => WITH 3
                   /* where len is an attribute width in **
                   ** bytes and <diskname> is a virtual  **
                   ** or physical disk name              */
<options>
ENDOFCOMMAND
```

"Create database" sets up an empty database on a framework of system relations. The database options that may be specified include: the number of blocks to allocate; the disk the database should be allocated to; and the disk on which to write the transactions log. Note that if there is no space on the specified disk the database will not be created. The syntax is:

```
                   create database <name>[with<options>]
```

The IDM command structure for "create database" is:

```
DBCREATE
RANGE <lenf> <rno> <dbname>
<rootnode>;        ROOT <rno> Ø
<T-list>:          TLEND = =>
<Q-list>:          [<= = logspec] <= = [<demandspec>] | QLEND
logspec:           QLEND = => WITH 4
<qspec>:           INT1 val | INT2 val | INT4 val = => WITH 1
<demandspec>:      <allocspec>
                   | <allocspec> <= =dskspec>
                   | <dskspec>
```

```
<allocspec>:        INT1 val | INT2 val | INT4 val = =>
                    WITH 2|5
<dskspec>:          CHAR <lenf> <diskname> = => WITH 3
<options>
ENDOFCOMMAND
```

When the user requires a new relation, and wants to put the results of the query currently being retrieved into the new relation "retrieve into" is used. The IDL syntax and IDM command structure take the following formats:

```
retrieve into exp_parts (p.name, p.cost)
                             order by cost:descending
                            where p.cost>avg (p.cost)

 RET_INTO
RANGE <lenf> <rno> relname1
   .

   .

   .

RANGE <lenf> <rno> relnamek
<rootnode>:         ROOT <rno> <byte2>
                    /* <byte2>= 1 if retrieve unique into;      **
                    ** <byte2>= Ø if retrieve into              */
<T-list>:           TLEND = => || " <resattnode> | ORDERDOM "
                             <= = <attnode> ||
<Q-list>:           <Q-subtree>  | QLEND
[ORDERA x]  | [ORDERD y]  /*  where x and y are attribute       **
                             **  numbers on the target list      **
```

```
        .                    ** of the query tree on which     **
        .                    ** the result is sorted           */
<options>
ENDOFCOMMAND
```

"Create view" is a command used to set up a virtual relation, one that is not a physical entity. A view is made of parts of one or more relations (base relations) or other views. A view may be protected or destroyed, and may be updated if the update can unambiguously be applied to one of the base relations. The IDL syntax for "create view" is:

```
create view <object name>(<target list>)
          [where<qualification>]
```

An example of a short "create view" query tree is reproduced here from the IDM manual.

```
range of p is parts
range of pr is products
create view mine (p.name, p.cost, pr.quan)
where pr.name = "TV"
and pr.part = p.name
```

```
VIEW
RANGE 6 Ø parts
RANGE 9 1 products
RANGE 5 2 mine
```

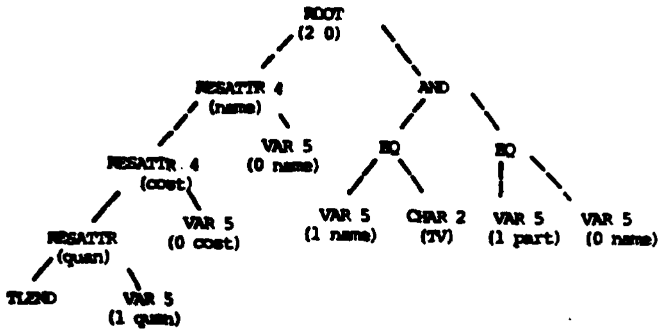


FIGURE 5.7 CREATE VIEW QUERY TREE

The IDM command format for "create view" is:

```
 VIEW
RANGE <lenfl> <rnol>
   .
   .
   .
RANGE <lenfk> <rnok>
RANGE <lenf> <viewname>
<rootnode>:           ROOT <rno> Ø
<T-list>:             TLEND || = => RESATTR <= = <attnode> ||
<Q-list>:             <Q-subtree> | QLEND
<options>
ENDOFCOMMAND
```

The define statement is used to define one of the following stored commands; retrieve, append, replace, delete, begin transaction or end transaction. The command may employ parameters anywhere that a constant would normally be acceptable. The IDL syntax is:

{define <name><command>{<command>}end define.

The IDM command mode requires that in constructing a "define tree: replace proper <varnode> which will become a parameter by corresponding <paramnod> in each command tree which appears in a stored command".

The command "destroy" is used to eliminate relations, files, views, and stored commands. The entire object is removed from the system, freeing the space for another object. Only the owner or the DBA can destroy objects. Objects with dependent views or stored commands must have those destroyed before the object may be destroyed. The destroy command may be implemented two ways: first, by specifying the object name and; second, by specifying relation names through the use of a target disk.

The IDL syntax is:

```
                    destroy<object name>{,<object name>}
               destroy (<target list>)[where<qualification>]
```

The IDM command format is:

```
DESTROY
<rootnode>:      ROOT Ø Ø
<T-list>:        TLEND || = => RESDOM <= = <attnode> ||
<Q-list>         <Q-subtree>
<options>
ENDOFCOMMAND
```

5.4 Data Manipulation Statements

The commands append, delete, and replace are data manipulation statements. These statements are designed to add, delete or modify column or row values within relations or views

The append command adds tuples to a relation or a view. Attributes are named and their values specified at this juncture. An attribute without a specified value is assigned a default value. The IDL syntax for the append command is:

```
                  append [to] <object name>(<target list>)
                              [where<qualification>]
```

The IDM command format follows:

```
APPEND

RANGE <lenf> <rno> relnamel

    .

    .

    .

RANGE <lenf> <rno> relnamek

<rootnode>:        ROOT <rno> Ø

<T-list>:          TLEND = => { <resattnode> <= = <attnode> }

<Q-list>:          <Q-subtree> | QLEND

<options>

ENDOFCOMMAND
```

The command "delete" is used to remove one or more tuples from a relation. Permission to delete is granted with "write" permission. The IDL syntax is:

```
delete <variable>[where<qualification>]
```

The IDM command structure for "delete" is:

```
DELETE

RANGE <lenf> <rno> relnamel

    .

    .

    .

RANGE <lenf> <rno> relnamek
<rootnode>:        ROOT <rno> Ø       /*  where <rno>=range no.    **
                                      **for the result variable    */
<T-list>:          TLEND

<Q-list>:          <Q-subtree> | QLEND
```

```
<options>

ENDOFCOMMAND
```

"Replace" is a command that replaces one or more attributes in zero or more tuples of a relation. A single replace command affects one relation. The variable is outside the target list. "Replace" may access one or more relations to calculate what is to be updated or changed. The IDL syntax is:

```
                    replace <variable>(<target list>)
                                   [where <qualification>]
```

The IDM command set structure for replace is:

```
REPLACE
RANGE <lenf> <rno> relname1
  .
  .
  .
RANGE <lenf> <rno> relnamek
<rootnode>          ROOT <rno> <byte2>
<T-list>:           TLEND = => || <resattnode> <= = <attnode> ||
<Q-list>:           <Q-subtree> | QLEND
<options>
ENDOFCOMMAND
```

5.5 Security and Concurrency Control Statements

This section focuses on security and concurrency control statements. In IDL the statements "permit" and "deny" are the framework for database security. Begin Transaction and End Transaction are the fundamental concurrency control statements.

The command "permit" allows specific user(s) or groups access to a relation, database, or index. Permission to read, write or "all", may be granted. Secure "users" names are recorded in the "users" relation by the DBA. Without user specifications, anyone may access the information. Permission to create, create index, and create database may also be granted. The IDL syntax is:

```
                 permit <protect mode>[[on|of]<object name>
                      {(<attlist>)][to<user>{,<user>}]
```

The IDM command syntax is: PERMIT

```
RANGE <lenf> <rno> <name>
<rootnode>:          ROOT <rno> mode
                     /* on read mode=Ø1 */
                     /* on write mode=Ø2 */
                     /* on all mode=Ø3 */
                     /* on execute mode-Ø34 */
<T-list>:            TLEND = => "RESDOM <= = <varnode>"
<Q-list>:            || CHAR <lenf> <username> = => QUALDOM ||
                     | QLEND
                     | QLEND
<username>:          /* a user to whom the object is denied */
<options>
ENDOFCOMMAND
```

"Deny" is a command used to bar access to users. Access may be denied to a relation, view, file or stored command. If there are no users specified, the protection applies to everyone. Access may be denied to read, write or all. Deny commands which contradict earlier permit commands take precedence. The DBA may also deny rights to use the create, create index, and create database commands. The IDL syntax is:

```
deny<protect mode>[[on|of]<object name>
         [ (<attlist>)]][to <user>{,<user>}]
```

The IDL command format is:

```
DENY
RANGE <lenf> <rno> <name>
<rootnode>:         ROOT <rno> <mode>
                    /* on read mode=Ø1 */
                    /* on write mode=Ø2 */
                    /* on all mode=Ø3 */
                    /* on execute mode=Ø34 */
<T-list>:           TLEND || = => RESDOM <= = <varnode> ||
<Q-list>:           || CHAR <lenf> <username> = => QUALDOM ||
                    <= =QLEND
<username>:         /* a name to whom the object is denied */
<options>
ENDOFCOMMAND
```

The command "begin transaction" is given whenever multiple IDM commands are to be treated as a single transaction. The IDL syntax is:

```
begin transaction.
```

The IDM command structure is:

```
BEGINLXACT
```

<options>

ENDOFCOMMAND

The command "end transaction" is used whenever a set of commands that commenced with a "begin transaction" is complete, and the user wishes to make the results of the transaction known to the rest of the system. The IDL syntax is

end transaction.

The IDM command structure is:

ENDXACT

<options>

ENDOFCOMMAND

SECTION 6. CORRELATION BETWEEN SQL AND IDM FEATURES

In the earlier sections of this analysis the features of ORACLE SQL (hereafter referred to as SQL) and IDM/IDL were described in detail. In this section, examples of the constructs of each language will be placed side by side, so that the reader can get an understanding for how SQL might be translated to IDL. Those features of SQL that are not available with the IDM are discussed later in this section. Please note that all of the examples use the classic database example of personnel/employee files. Information such as employee: name, number, salary, department, location and title is manipulated to show the capabilities of both systems. The SQL formats are capitalized, and the IDL formats are presented in the lower case.

6.1 Query Facilities

```
ORACLE SQL                                 IDL
```

1. Find the names of the employees in department 50. (Query command)

```
SELECT   NAME                     range of e is emp
FROM     EMP                      retrieve (e.name) where e.deptno=50
WHERE    DEPTNO=50
```

2. Find the names of the employees in departments 25, 47 and 53. (Where clause)

```
SELECT   NAME                     range of e is emp
FROM     EMP                      retrieve (e.name) where e.deptno=25
WHERE    DEPTNO IN (25,47,53)     OR e.deptno=47 or e.deptno=53.
```

3. List the names of all employees who earn between $1,200 and $1,400 (Range of values)

```
SELECT   NAME                     range of e is emp
FROM     EMP                      retrieve (e.name) where e.sal ≥
WHERE SAL BETWEEN 1200 and 1400     1200 and e.sal ≤1400
```

4. Find the names of employees who work for departments in New York (Nested queries)

```
SELECT    NAME                      range of e is emp
FROM      EMP                       range of d is dept
WHERE     DEPTNO IN                 retrieve (e.name) where e.deptno=
          SELECT DEPTNO               d.deptno
          FROM DEPT                 and d. loc = "New York"
          WHERE LOC = "NEW YORK"
```

5. List the names of employees who have the same job and salary as `SMITH` (Range variables on the same table)

```
SELECT    NAME                      range of e is emp
FROM      EMP                       range of f is emp
WHERE     <JOB,SAL>=                retrieve (e.name) where f.name=
  SELECT        JOB,SAL              "SMITH"
  FROM          EMP                 and f.job=e.job and f.sal=e.sal
  WHERE         NAME=`SMITH`
```

6. List all of the departments and the average salary within each of them (GROUP BY (SQL)/BY(IDL)clause)

```
SELECT    DEPT NO, AVG (SAL)        range of e is emp
FROM      EMP                       retrieve (e.deptno, asal=avg
GROUP     BY DEPTNO                   (e.sal by e.deptno))
```

7. List the departments in which the average employee salary is less than 10,000 (HAVING clause (SQL)/WHERE clause (IDL))

```
SELECT    DEPTNO                    range of e is emp
FROM      EMP                       retrieve (e.deptno) where avg
GROUP     BY DEPTNO                   (e.sal by e.deptno) < 10000
HAVING    AVG (SAL) < 10000
```

8. List the departments that employ more than ten clerks (COUNT function)

```
SELECT    DEPTNO                    range of e is emp
FROM      EMP                       retrieve (e.deptno) where count
WHERE     JOB = `CLERK`               (e.job by e.deptno where e.job =
GROUP     BY DEPTNO                   "CLERK") > 10
HAVING    COUNT (*) > 10
```

9. Determine the number of different jobs held by employees in department 50 (UNIQUE aggregates)

```
SELECT    COUNT (UNIQUE JOB)        range of e is emp
FROM      EMP                       retrieve (temp = count unique
WHERE     DEPTNO = 50               (e.job where e.deptno=50))
```

10. When a CREATE TABLE statement is used in SQL with the IMAGE option

defined on a column, it indicates that an index is to be maintained for the values in that column. In IDL, a clustered or nonclustered index can be created for an attribute or a group of attributes in a relation. The IDM can sort the relation by its "parts" on the part number, and then create a directory that relates the part number to the physical location of the associated part tuple. That command is stated:

```
create clustered index on parts (number)
```

6.2 Data Manipulation Facilities

```
        SQL                                       IDL
```

1. Insert a new employee information into the employee table. (INSERT INTO (SQL), append (IDL) clause)

```
INSERT INTO EMP(EMPNO,NAME,JOB,SAL,       append to emp(empno=7989,
  COMM,DEPTNO): <7989, `CARTER`,          name = "CARTER",
  `SALESMAN`, 1500,0,30>                  job="SALESMAN", SAL=1500,
                                          comm=0, deptno=30)
```

2. Delete the employee tuple with employee number 561 from the EMP table (DELETE clause)

```
DELETE EMP                                range of e is emp
WHERE EMPNO = 561                         delete e where e.empno=561
```

6.3 Data Definition Facilities

1. Create a new table to contain department, name and location information (CREATE TABLE statement)

```
        SQL                                       IDL

CREATE TABLE DEPT                         creat dept (deptno=UC2, name=C12,
(DEPTNO (CHAR(2), NONULL),DNAME           loc = c20)
(CHAR(12)VAR),LOC(CHAR(20)VAR))
```

Note that names are limited to 12 characters in IDL, and 30 characters in ORACLE.

2. Define a view called PROGS consisting of the names and salaries of all programmers and the locations of their departments (DEFINE statement)

```
DEFINE VIEW PROGS                         range of e is emp
```

```
(NAME,SALARY,HOMEBASE)AS                range of d is dept
SELECT EMP.NAME,EMP.SAL,DEPT.LOC        create view progs (name=e.name,
FROM EMP,DEPT                           salary = e.sal, homebase=d.loc)
WHERE EMP.DEPTNO=DEPT.DEPTNO            where e.deptno=d.deptno and
AND EMP JOB=`PROGRAMMER`                e.job="programmer"
```

3. Add a new column called NEMPS, of integer type, to the table DEPT (EXPAND TABLE statement)

```
EXPAND TABLE DEPT                       range of d is dept
ADD COLUMN NEMPS(INTEGER)               retrieve into ndept (deptno=
                                        d.deptno, d.name=d.dname,
                                        loc=d.loc,nemps=Ø)
                                        destroy dept
                                        rename ndept, dept
```

Note that in IDL to add a column it is necessary to: create a new expanded table destroy the old table, and use the name of the old table for the new table.

4. Destroy the view D5O (DROP (SQL), destroy (IDL), command)

```
DROP VIEW D5Ø                           destroy d5Ø
```

6.4 Security and Concurrency Control Facilities

1. Authorize user GEORGE to READ the DEPT table (GRANT(SQL)perm-t(IDL) statement)

```
            SQL                                     IDL

       GRANT    READ                    permit read of dept to george
       ON       DEPT
       TO       GEORGE
```

2. Revoke from user GEORGE the right to WRITE in the DEPT table. (REVOKE (SQL), deny(IDL)statement)

```
       REVOKE   WRITE                   deny write on dept to george
       ON       DEPT
       FROM     GEORGE
```

3. Calculate the average salaries of each job position within the EMP table (BEGIN and END transaction)

```
       BEGIN TRANSACTION                range of e is emp
       ON TABLE EMP READ                begin transaction
                                        retrieve (e.job, a=avg(e.sal by
SELECT JOB, AVG(SAL)          e.job))
FROM EMP                      end transaction
GROUP BY JOB
END TRANSACTION
```

Certain features implemented in ORACLE/SQL are not implemented on the IDM. A brief list of these capabilities follows. For expanded descriptions see Section 2 of this analysis or the ORACLE USER'S GUIDE - Version 2.3.

1. In ORACLE/SQL names may be 30 characters long. On the IDM, the maximum name length is 12 characters.

2. ORACLE/SQL has the capability for retrieval operations on tree-structured tables. This capability is not implemented in the IDM.

3. ORACLE automatically maintains an index (IMAGE) for the first column defined in a table. The IDM requires the user to explicitly indicate on which columns to define an index.

4. In ORACLE/SQL columns can be added to the right side of existing tables by means of the EXPAND TABLE statement. With the IDM, the user must build a new expanded table, destroy the old table, and rename the new table with the name of the old table.

5. The ORACLE/SQL GRANT command enables users to grant the following privileges to other users: READ, INSERT, DELETE, UPDATE and EXPAND. The IDM permits READ, WRITE, INSERT and UPDATE.

6. In ORACLE/SQL the Null Value Function is used. The IDM has no equivalent.

SECTION 7. ADDITIONAL IDM FEATURES

The IDM has both software and special-purpose hardware features that are significant for their range, speed and depth. These features such as the transaction management functions, the random access file system, and the complete relational database management system are all described in detail in the Britton-Lee Product Description. Some of the unique features of the IDM are highlighted below.

Within the IDM data management system is the capacity to use a stored command. A stored command is one defined earlier by the user, and stored in a partially processed form in the IDM. From this point on, the command can be referred to by the user with a short name or number. Both the transmission and execution times are minimized because the command is stored in the IDM.

Using the stored query feature is of critical importance for front-end programs. This function allows the internal form of the query to be stored in the IDM. The front-end program sends the query name and appropriate parameters. This reduces the amount of information that needs to be transmitted to run a query. It also reduces the size of the front-end program, leading to more efficient programs.

The IDM has a 1-31 decimal digit BCD (binary coded decimal) and 1, 2 and 4-byte integers. When a relation is created, the maximum length of the BCD and character attributes are specified. The IDM automatically compresses data to save storage space.

Tuples are accessed by values not by position. Therefore the structure of a relation can change. Attributes can be added, and the relation can be reorganized with very little impact on end-user programs. The values are specified in the qualification.

The IDM has two special constant functions in addition to the standard aggregate functions. The "time" function supplies the time of day in a 4-byte integer. The "date" function provides the date in a 4-byte integer.

The IDM has the capability to handle complex aggregate functions using built in functions within WHERE clauses, a capability that is not available in SQL. The following example, extracted from The Preliminary Performance Report is a good illustration.

> "For each account get the name of the account and the average balance for this account type for those account types whose average balance is greater than twice the minimum of all account types.
>
> range of a is accounts
>
> retrieve (a.name, avg = ave (a.balance by a.type))
>
> where avg (a.balance by a.type) > (min(a.balance by a.type)*"

The IDM also has the capability to handle an aggregate within an aggregate. The following is an example:

```
retrieve (a=avg(max(e.sal by e.deptno)))
```

Using a qualification within an aggregate is also a special capability of the IDM.

```
retrieve (a=avg(max(e.sal by e.deptno where e.yrs>5)))
```

SECTION 8. UNIMPLEMENTED FEATURES OF FULL SEQUEL

Certain features of SEQUEL as defined in the article, "SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control" that appeared in the IBM Journal of Research and Development, (Vol. 20 November, 1976) were not implemented in ORACLE. These features are noted briefly in the following section and illustrated with examples.

1. In SEQUEL query-expr nonterminal defines set operations. ORACLE does not support set operations.

```
SEQUEL                                 ORACLE

  query::=query-expr                     query::=query-block[ORDER BY ord-spec-list]
  [ORDER BY ord-spec-list]

  query-expr::=query-block
          |query-expr set-up query block
          |(query-expr)
```

2. The SEQUEL syntax includes an INTO clause, which is used for the host language interface. This capability is not included in the ORACLE syntax. The ORACLE syntax permits a HAVING clause without a GROUP BY clause which is forbidden by the SEQUEL syntax.

```
  query-block::=select-clause          query-block::= select-clause
               [INTO target-list]                       FROM from-list
                                                      [WHERE boolean]
               FROM from-list                         [GROUP BY field-spec-list]
               [WHERE boolean]                        [HAVING boolean]
               [GROUP BY field-spec-list]             [CONNECT BY[PRIOR]
               [HAVING boolean]]                        field-spec=field-spec]
                                                       [START WITH boolean]
                                                       [INCLUDING boolean]
```

3. SEQUEL permits an OLD or NEW qualifier, to be used with both assertions and triggers. This concept is not supported by ORACLE. The ORACLE NVL provides a default value to be used in place of null values. SEQUEL does not provide an NVL function.

```
SEQUEL                                        ORACLE

primary::=[OLD!NEW]field-spec             primary::=field-spec
          !set-fn([UNIQUE]expr)                      !set-fn(expr)
          !count(*)                                  !count(*)
          !constant                                  !NVL(field-spec, constant)
          !(expr)                                    !constant
                                                     !(expr)
```

4. SEQUEL syntax permits named objects to be differentiated by their creator. ORACLE syntax does not support this capability.

```
name::=[creator.]identifier               name::=identifier
```

5. The SEQUEL syntax permits the implementation of special purpose user defined set functions which are added to a special program library. ORACLE does not support this.

```
set-fn::=AVG!MAX!MIN!SUM!COUNT!identifier   set-fn::=AVG!MAX!MIN!SUM!COUNT
```

6. The SEQUEL syntax provides host-location and CURSOR references for the host language interface. ORACLE does not support these or the USER/DATE functions.

```
SEQUEL                                        ORACLE

constant::= quoted-string                     constant::= quoted string
           !number                                       !number
           !host-location                                !NULL
           !NULL
           !USER
           !DATE
           !field-name OF CURSOR
            cursor-name ON table name
```

7. SEQUEL uses parentheses to grasp boolean operations while ORACLE uses brackets.

```
boolean-primary::= predicate                  boolean-primary::= predicate
                  !(boolean)                                    ![boolean]

                                              *NOTE-brackets are terminal symbols
                                                   here
```

8. SEQUEL supports set operations with IS NOT IN clauses, set functions, and comparisons between two tables. It also supports an IF a THEN b construction. Neither of these capabilities are present within ORACLE.

```
predicate::=expr comparison expr
         |expr BETWEEN expr AND expr
         |expr comparison table-spec
         |<field-spec-list>=table-spec
         |<field-spec-list>[IS][NOT]
          IN table spec
         |IF predicate THEN predicate
         |SET(field-spec-list)comparison
          table spec
         |SET(field-spec-list)comparison
           SET (field-spec-list)
         |table-spec comparison table-
          spec
```

```
predicate::=expr comparison expr
         |expr BETWEEN expr AND
          expr
         |expr comparison table-
          spec
         |<field-spec-list>=
           table-spec
         |<field-spec-list>
          [IS]IN table spec
```

9. SEQUEL supports the set operations CONTAINS, DOES NOT CONTAIN, and as previously noted, IS NOT IN. ORACLE does not support these operations.

SEQUEL

```
comparison::=comp-op
          |CONTAINS
          |DOES NOT CONTAIN
          |[IS]IN
          |[IS]NOT IN
```

ORACLE

```
comparison::=comp-op
          |[IS]IN
```

10. SEQUEL uses angle brackets to delimit tuples, and parentheses to delimit lists of tuples or scalar constants. ORACLE uses angle brackets for both of these purposes.

```
literal::=(lit-tuple-list)
       |lit-tuple
       |(entry-list)
       |constant
```

```
literal::=<lit-tuple-list>
       |lit-tuple
       |constant
```