R S I

ORACLE

SQL LANGUAGE

REFERENCE GUIDE

Oracle Users Guide - Version 2.3

S Q L

LANGUAGE REFERENCE MANUAL

TABLE OF CONTENTS

SQL LANGUAGE - REFERENCE GUIDE

INTRODUCTION

SQL is 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.

SQL was developed by IBM as the main external interface to be supported by System R, IBM's experimental relational database management system. In 1976, a complete BNF syntax for SQL was published in the "IBM Journal of Research and Development." In 1977, RSI began the development of ORACLE incorporating the SQL language.

ORACLE is based on the relational model of data. SQL is a non-procedural language that operates on normalized data. The advantages of the relational model with a non-procedural language are ease of use, maximum data independence, and flexibility. SQL is an easy to learn English-like language that enhances user productivity. It is a high-level non-procedural language offering greater data independence than conventional procedural database languages. SQL allows complete flexibility in the formulation of statements relating data in the database.

SQL

FORMAT NOTATION

This manual uses the following notation to describe the syntax of SQL statements.

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.

QUERY STATEMENTS

SQL Query Statements consist of one or more Query Blocks. A Query Block starts with and must include a SELECT clause and a FROM clause. The SELECT clause specifies what is to be returned as a result of the query block. The FROM clause specifies what tables and/or views are involved in the query.

A Query Block may optionally contain other clauses as follows:

WHERE	to specify selection criteria for the rows.
GROUP BY	for use with built-in functions.
HAVING	for specifying election criteria on groups.
CONNECT BY	for tree-structured access.
START WITH	for tree-structured access.
INCLUDING	for specifying selection criteria for leafs for tree-structured access.

The values resulting from processing a Query Block can be referred to in the WHERE clause of another Query Block. This is accomplished by nesting Query Blocks within a Query Statement.

```
SELECT  ENAME,JOB
FROM    EMP
WHERE   JOB =
        SELECT   JOB
        FROM     EMP
        WHERE    ENAME = 'JONES';
```

Query Blocks can be nested to any level within a Query Statement, and may be combined with other SQL predicates using boolean AND, OR, and NOT. In the syntax specifications of the SQL statements, "SELECT . . ." is used to denote a nested Query Block.

The SELECT Clause

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

The SELECT clause specifies the columns to be returned as the result of a query. The SELECT clause may request: all columns; a list of specific columns; the results of arithmetic expressions or built-in functions; or any combination of columns, expressions, and functions. ORACLE will return all rows that satisfy the WHERE clause of the query block. Duplicate rows are not eliminated unless SELECT UNIQUE is specified. UNIQUE is an option rather than a default because the process of elimination of duplicate values requires extra processing.

UNIQUE	indicates that duplicate rows which satisfy the WHERE clause are to be eliminated from the query result.
*	returns all columns from all of the table(s) and view(s) specified in the FROM clause of the query block.
column	specifies the name of a column contained in a table or view specified in the FROM clause of the query block.
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.*	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.
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.
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. See the GROUP BY clause and Built-in-Functions sections for a more detailed description.
USER	returns the name of the user (as specified in the DEFINE USER command) who is executing this SQL statement.

The FROM Clause

```
FROM  |table      |, . . .
      |table label|
      |table*     |
```

The 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, and HAVING clause.

table	specifies the name of a table or view that contains columns referenced by SELECT, WHERE, GROUP BY, HAVING, or ORDER BY clauses.
table label	specifies that the table or view 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.
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 can not be the first table listed in the FROM clause.

The WHERE Clause

```
WHERE  [NOT]  |column        |  |=       |  |column             |  |AND| . . .
              |table.column  |  |^=      |  |table.column       |  |OR |
              |constant      |  |>       |  |constant           |
              |NULL          |  |>=      |  |generic-constant   |
              |expression    |  |<       |  |NULL               |
              |<column,...>  |  |<=      |  |expression         |
              |USER          |  |BETWEEN |  |<column,...>       |
                                |IN      |  |<expression,...>   |
                                            |SELECT...          |
                                            |USER               |
```

The WHERE clause 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. Multiple predicates within the same WHERE clause can be combined to form logical expressions connected by AND and OR with square brackets [ ] used to establish precedence. NOT may be specified prior to any predicate to negate a predicate or a boolean expression. The absence of a WHERE clause indicates that all rows in the table or view specified in the FROM clause, are to be returned.

NOT	specifies that the following predicate or boolean expression is to be negated.
column	specifies the name of a column defined in a table or view specified in the FROM clause of the query block. Note that columns specified here, need not be specified in the SELECT 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.

constant	specifies any numeric or character-string constant literal value. Single quotation marks are required around all character-string constants to distinguish them from column names.
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.
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. The evaluation of logical expressions is described in a separate section of this manual which contains the truth tables for three valued logic.
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.

<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.
<expression,...>	specifies a set of constant values or expressions. The predicate in the WHERE clause tests the field for inclusion in the set. example: WHERE DNO IN <5+2,17,11*3>
SELECT...	specifies the use of the result of one query in the WHERE clause of another query. The inner query returns a set of values to the WHERE clause of the outer query. The outer query proceeds as though it were given a set of constants in place of the inner query. Query blocks may be nested to any number of levels.
USER	returns the name of the user (as specified in the DEFINE USER command) who is executing this SQL statement.
=	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 less than comparison operator.
<=	indicates the less than or equal comparison operator.

BETWEEN	indicates the range comparison operator. The range is specified as a pair of constants, expressions, or columns connected by an AND.
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 without changing the meaning of the WHERE clause.
AND	indicates the boolean operator AND. The boolian operators are used to connect predicates to form compound logical expressions within the WHERE clause.
OR	indicates the boolean operator OR.

Built-In Functions

```
|COUNT|   |*            |
|SUM  |   |column       |
|AVG  |   |table.column|
|MAX  |
|MIN  |
```

ORACLE provides five built-in functions as a standard part of the system. These functions may be used in the SELECT clause and the HAVING clause. When a built-in function is used in a SELECT clause, and there is no GROUP BY clause in the query block, the entire table is treated as one group. Only unique attributes of the group may be selected. No Built-in function other than 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.

COUNT	specifies the count of the set of all fields or rows qualified by the WHERE clause. COUNT includes null fields in its total.
SUM	specifies the arithmetic sum of the values of qualifying fields.
AVG	specifies the arithmetic average of the values contained in the set of qualifying fields.
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.
*	specifies the count of all rows that satisfy the WHERE clause. The * may only by used with the COUNT function in the form: COUNT(*).

column	specifies the name of a column defined in a table or view specified in the FROM clause of the query block.
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.

Null Value Function

NVL (column,value)

The ORACLE Null Value Function is used to assign 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.

column	specifies the name of a column within a SELECT, SET or WHERE clause. The column must have been defined as NUMBER within the CREATE TABLE.
value	specifies a temporary numeric value to be assigned to null values encountered during processing.

The GROUP BY Clause

```
GROUP BY  |column       |, . . .
          |table.column|
```

The GROUP BY clause 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 together with a built-in function. 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.

column	specifies the name of a column defined in a table or view specified in the FROM clause of the query block. Note that columns specified here, need not be specified in the SELECT 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.

The HAVING Clause

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

The 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. The HAVING clause may contain any combination of predicates to accept certain groups and disqualify others. The predicates can use a built-in function to compare the aggregate property of the group to a constant value or to another aggregate property of the same group. When a query block has both a WHERE clause and a HAVING clause: first the WHERE clause is applied to qualify rows; then the groups are formed; then the HAVING clause is applied to qualify groups. Multiple predicates within the same HAVING clause form logical expressions connected by AND and OR with square brackets [ ] used to establish precedence. The absence of a HAVING clause indicates that all groups formed are to be returned.

column	specifies the name of a column defined in a table or view specified in the FROM clause of the query block. Note that columns specified here, need not be specified in the SELECT 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.

constant	specifies any numeric or character-string constant literal value. Single quotation marks are required around all character-string constants to distinguish them from column names.
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.
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).
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.
<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.
<expression,...>	specifies a set of constant values or expressions. The predicate in the HAVING clause tests the field for inclusion in the set.

SELECT...	specifies the use of the result of one query in the HAVING clause of another query. The inner query returns a set of values to the HAVING clause of the outer query. The outer query proceeds as though it were given a set of constants in place of the inner query. Query blocks may be nested to any number of levels.
USER	returns the name of the user (as specified in the DEFINE USER command) who is executing this SQL statement.
=	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 less than comparison operator.
<=	indicates the less than or equal comparison operator.
BETWEEN	indicates the range comparison operator. The range is specified as a pair of constants, expressions, or columns connected by an AND.
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 without changing the meaning of the WHERE clause.
AND	indicates the boolean operator AND.
OR	indicates the boolean operator OR.

The CONNECT BY Clause

```
CONNECT BY [PRIOR] major-column = [PRIOR] minor-column
```

In ORACLE, a table may be used to represent tree-structured data. Consider a parts table consisting of assemblies (major) and components (minor). There is one row in the table for each combination of a component and assembly. One assembly can be a component of another assembly, etc. The table contains one column identifying component-number and another column identifying assembly-number. This table then represents a complete bill of materials.

The CONNECT BY clause specifies the selection of rows according to their tree-structure relationship. The clause requires specification of the major and minor columns. The PRIOR keyword is positioned before the major column to indicate that the rows are to be selected going up the tree, or before the minor column to indicate the rows are to be selected going down the tree.

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).
major-column	specifies the name of the assembly column.
minor-column	specifies the name of the component column.

The START WITH Clause

```
START WITH [NOT] |column        |  |=      |  |column              |  |AND| . . .
                 |table.column|   |^=     |  |table.column        |  |OR |
                 |constant      |  |>      |  |constant            |
                 |NULL          |  |>=     |  |generic-constant|
                 |expression    |  |<      |  |NULL                |
                 |<column,...>|   |<=     |  |expression          |
                 |USER          |  |BETWEEN|  |<column,...>        |
                                   |IN     |  |<expression,...>|
                                              |SELECT...           |
                                              |USER                |
```

The START WITH clause 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.

NOT	specifies that the following predicate or boolean expression is to be negated.
column	specifies the name of a column defined in a table or view specified in the FROM clause of the query block.
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.
constant	specifies any numeric or character-string constant literal value. Single quotation marks are required around all character-string constants to distinguish them from column names.

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.
NULL	indicates the absence of a value in the database.
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.
<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.
<expression,...>	specifies a set of constant values or expressions.
SELECT...	specifies that the result of a query block is to be used in the START WITH clause.
USER	returns the name of the user (as specified in the DEFINE USER command) who is executing this SQL statement.
=	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 less than comparison operator.
<=	indicates the less than or equal comparison operator.
BETWEEN	indicates the range comparison operator. The range is specified as a pair of constants, expressions, or columns connected by an AND.
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.
AND	indicates the boolean operator AND. The boolian operators are used to connect predicates to form compound logical expressions.
OR	indicates the boolean operator OR.

The INCLUDING Clause

```
INCLUDING [NOT] |column        |  |=      |  |column              |  |AND| . . .
                |table.column  |  |^=     |  |table.column        |  |OR |
                |constant      |  |>      |  |constant            |
                |NULL          |  |>=     |  |generic-constant    |
                |expression    |  |<      |  |NULL                |
                |<column,...>  |  |<=     |  |expression          |
                |USER          |  |BETWEEN|  |<column,...>        |
                                  |IN     |  |<expression,...>    |
                                             |SELECT...           |
                                             |USER                |
```

The INCLUDING clause is used with queries on tree-structured tables. The INCLUDING clause is used in conjunction with the WHERE clause to determine which rows are to be returned as a result of the query. Any rows which are excluded by virtue of not satisfying the WHERE clause, result in exclusion of entire "branches" of the tree structure. Any rows which are excluded by virtue of not satisfying the INCLUDING clause, result only in that row being excluded. In other words, the WHERE clause causes exclusion before the CONNECT BY is applied; the INCLUDING clause causes exclusion after the CONNECT BY is applied.

The INCLUDING clause may contain 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.

NOT	specifies that the following predicate or boolean expression is to be negated.
column	specifies the name of a column defined in a table or view specified in the FROM clause of the query block.
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.
constant	specifies any numeric or

	character-string constant literal value. Single quotation marks are required around all character-string constants to distinguish them from column names.
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.
NULL	indicates the absence of a value in the database.
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.
<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.
<expression,...>	specifies a set of constant values or expressions.
SELECT...	specifies that the result of a query block is to be used in the INCLUDING clause.

USER	returns the name of the user (as specified in the DEFINE USER command) who is executing this SQL statement.
=	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 less than comparison operator.
<=	indicates the less than or equal comparison operator.
BETWEEN	indicates the range comparison operator. The range is specified as a pair of constants, expressions, or columns connected by an AND.
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.
AND	indicates the boolean operator AND. The boolian operators are used to connect predicates to form compound logical expressions.
OR	indicates the boolean operator OR.

The ORDER BY Clause

```
ORDER BY  |column       |   |ASC |, . . .
          |table.column|   |DESC|
          |expression   |
```

The ORDER BY clause 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. Each sort field may specify ascending or descending order. An ORDER BY clause is not part of a query block, and may only be associated with the first query block of a SQL query statement.

column	specifies the name of a column defined in a table or view specified in the FROM clause of the query block. Note that columns specified here, need not be specified in the SELECT 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.
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.
ASC	indicates ascending sort order. If no sort direction is specified for a field, ascending is assumed.
DESC	indicates descending sort order.

DATA MANIPULATION STATEMENTS

The INSERT INTO and DELETE clauses provide for addition and deletion of rows of a table. The combination of the UPDATE and SET clauses allows modification of column values within a row or set of rows within a table.

Nested Query Blocks may be used with the INSERT INTO clause to copy data from another table. A WHERE clause is used with the DELETE and UPDATE clauses to specify sets of records to be processed.

Refer to the section on Concurrency Control Statements for information about locking during the execution of data manipulation statements.

The following SQL clauses are provided for Data Manipulation:

INSERT INTO	for adding rows to a table.
DELETE	for deleting rows from a table.
UPDATE	for specifying a table whose rows are to be updated.
SET	for specifying the updates to be made to columns of a row.

The INSERT INTO Clause

```
INSERT INTO  table(column,...): |<constant,...>    |
                                |NULL              |
                                |USER              |
                                |SELECT . . .      |
```

The INSERT statement 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. If the row to be inserted has all its fields present in the correct order, the list of column names may be omitted.

table	specifies the name of the table into which the rows are to be inserted.
(column,...)	specifies the names of the columns of the table in the order the values will appear in the INSERT statement. If values are being provided for all columns of a row (with any missing values being indicated by the keyword NULL), and the columns are in the order that they are defined in the CREATE TABLE, then the column list may be omitted.
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.
NULL	indicates that the column associated with the NULL is to be null in the database.
USER	returns the name of the user (as specified in the DEFINE USER command) who is executing this SQL statement.

SELECT...	specifies that the result of a query is to be inserted into a table in the database. Query blocks specified within an INSERT statement can be nested to any number of levels.

The DELETE Clause

```
DELETE  table
```

The 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. The WHERE clause in a DELETE statement is identical to the WHERE clause in a query and may contain nested query blocks.

table	specifies the name of the table that contains the rows to be removed from the database.

The UPDATE Clause

UPDATE table

The UPDATE clause 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. The specific row or rows to be modified are qualified by means of a WHERE clause. The WHERE clause in a UPDATE statement is identical to the WHERE clause in a query and may contain nested query blocks. Primary keys may not be modified by an UPDATE statement (see CREATE TABLE).

table	specifies the name of a table that is to be modified.

The SET Clause

```
SET    column = |constant   |, . . .
                |expression|
```

The SET clause 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 stated as constants or expressions.

constant	specifies any numeric or character-string constant literal value as the new value for the field. Single quotation marks are required around all character-string constants to distinguish them from column names.
expression	specifies the use of the result of an arithmetic expression as the new value for the field. An arithmetic expression can be made up of columns and constants that are connected by the operators +, -, *, /. Parenthesis ( ) are used to establish precedence.

DATA DEFINITION STATEMENTS

The SQL Data Definition Statements provide for establishing and modifying data definitions within the ORACLE Data Dictionary. The execution of these statements does not require any reorganization activity.

The following statements are provided:

CREATE TABLE	for defining a new TABLE in the database.
EXPAND TABLE	for defining a new COLUMN for an existing TABLE.
DEFINE VIEW	for defining a new VIEW.
DROP	for removing a TABLE or VIEW definition.

The CREATE TABLE Statement

```
CREATE TABLE table

  column(|CHAR(len) [VAR]| [NONULL] [UNIQUE] [UC] [IMAGE]), . . .
        |NUMBER         |
```

The CREATE TABLE statement 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. The presence of null or duplicate values within a column may be restricted. High-performance access paths may be specified on any columns.

ORACLE automatically maintains an index (IMAGE) for the first column defined in the table. To optimize sequential processing the rows of the table are stored in physical sequence based on this index. This column is automatically treated as a required (NONULL) item.

table	specifies the name of the table that is being defined. The name must be unique within the database. The maximum length of the table name is 30 characters. The first character must be alphabetic.
column	specifies the name of a column defined within the table. Column names must be unique within a table. The column name can have a maximum length of 30 characters. The first character must be alphabetic.
CHAR	indicates the column is to contain alpha-numeric character string 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.

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.
NUMBER	indicates the column is to contain numeric values. Numeric values are stored internally in base 256 format to maintain maximum precision.
NONULL	indicates that null values are not permitted in the column.
UNIQUE	indicates that no two fields within this column can have the same value. UNIQUE can only be specified if IMAGE is also specified.
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.
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.

The EXPAND TABLE Statement

```
EXPAND TABLE table

 ADD COLUMN column(|CHAR(len)  [VAR]|[NONULL]|[UNIQUE]  [UC]  [IMAGE]|)
                  |NUMBER           |
```

The EXPAND TABLE statement 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. Queries and views that were written in terms of the existing table are not affected by the expansion. EXPAND TABLE specifies the name of the table to be enlarged and defines the new column with a syntax identical to that used in the CREATE TABLE statement. The presence of null or duplicate values within the column may be restricted. A high-performance access path (IMAGE) may be specified. EXPAND TABLE is a instantaneous operation. No physical reorganization of any part of the database takes place.

table	specifies the name of the table that is being expanded.
column	specifies the name of a column being added to the table. Column names must be unique within a table. The column name can have a maximum length of 30 characters. The first character must be alphabetic.
CHAR	indicates the column is to contain alpha-numeric character string 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.

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.
NUMBER	indicates the column is to contain numeric values. Numeric values are stored internally in base 256 format to maintain maximum precision.
NONULL	indicates that null values are not permitted in the column.
UNIQUE	indicates that no two fields within this column can have the same value. UNIQUE can only be specified if IMAGE is also specified.
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.
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.

The DEFINE VIEW Statement

```
DEFINE VIEW  view [(column, . . .)] AS  SELECT . . .
```

The DEFINE VIEW statement creates an alternative view of data stored in tables in the database. The definition of a view is similar to the process of stating a query, because the result of any query on one or more tables is itself a table. Therefore, any query formulation can be used in the definition of a view. 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.

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.
column	specifies the name of a column defined within the view. Column names must be unique within a view. The column name can have a maximum length of 30 characters. The first character must be alphabetic. A view's column names may be drawn from the SELECT clause of the query defining the view if the column names in the SELECT clause are unique.
SELECT...	specifies the use of the result of a query as a view on the database. Any valid query block can be used as a database view. The query blocks may be nested to any number of levels.

The DROP Statement

```
DROP    |TABLE|   name
        |VIEW |
```

The 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.

TABLE	indicates the system entity to be dropped is a table. A table may not be dropped until all rows in that table have been deleted.
VIEW	indicates the system entity to be dropped is a view.
name	specifies the name of the table or view to be dropped.

DATA CONTROL STATEMENTS

The SQL Data Control Statements provide for Security and Concurrency Control.

The following SQL statements are provided for Security Control:

DEFINE USER	to define a user of a database and his password.
GRANT	to give privileges on a TABLE or VIEW to a user.
REVOKE	to remove privileges on a TABLE or VIEW from a user.
PASSWORD	to allow a user to change his password.

The following SQL statements are provided for Concurrency Control:

BEGIN TRANSACTION	Lock a resource.
END TRANSACTION	Unlock a resource.

The DEFINE USER Statement

```
DEFINE USER  user-name/password
```

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

Initially, the user who creates the database is the only authorized user of that database. Thereafter, the creating user can define additional users via the DEFINE USER command. These new users may themselves define additional users, etc.

Users defined by means of the DEFINE USER command are authorized to log on to a secure database and create tables. These users are not allowed any access to any data stored within the data base without data access privileges. Data access privileges are given to a user via the GRANT command.

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.
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.

The GRANT Statement

```
GRANT|privilege[,...]          |ON table TO|user-name[,...]|
     |ALL RIGHTS               |           |PUBLIC         |
     |ALL BUT privilege[,...]  |

                                              [WITH GRANT OPTION]
```

It is the responsibility of the user who creates a table or view to control access to it. When a user creates a table, only that creating user is privileged to access that table. The creating user may allow other users access privileges on his table via the GRANT command.

The following privileges may be granted:

READE
INSERT
DELETE
UPDATE (by column)
EXPAND

Only the READ privilege may be specified for a view.

Users that have been granted the right to exercise a privilege may or may not have the right to grant the same privilege to other users. The grantor of privileges may permit the grantee to grant the listed privileges to other users by including the clause WITH GRANT OPTION.

privilege	specifies the type of operations that are to be authorized for the table.
ALL RIGHTS	indicates that all privileges are to be granted.
ALL BUT	indicates that all privileges except those listed in the GRANT command are to be granted.
table	specifies the name of the table or view for which the privileges apply. If a view is specified, only the READ privilege may be granted.

user-name	specifies the name of the user or users that are to receive the privileges. User-name is the user-name field specified in the DEFINE USER command.
PUBLIC	indicates that all users of the database are to receive the privileges listed.
WITH GRANT OPTION	specifies that the grantee may grant the privileges listed to other users.

The REVOKE Statement

```
REVOKE |privilege[,...]         | ON table FROM |user-name[,...]|
       |ALL RIGHTS              |               |PUBLIC         |
       |ALL BUT privilege[,...] |
```

Privileges that have been granted by means of the GRANT command may be withdrawn through the use of the REVOKE command. The named privileges are removed from the grantee and from all users to whom he has granted them.

The following privileges may be revoked:

READE
INSERT
DELETE
UPDATE (by column)
EXPAND

Only the READ privilege may be specified for a view.

privilege	specifies the type of operation that is no longer authorized for the table.
ALL RIGHTS	indicates that all privileges are to be revoked.
ALL BUT	indicates that all privileges except those listed in the REVOKE command are to be withdrawn.
table	specifies the name of the table or view for which the privileges are to be revoked. If a view is specified, only the READ privilege may be revoked.

user-name	specifies the name of the user or users whose privileges are to be revoked. User name is the user-name field specified in the DEFINE USER command.
PUBLIC	indicates that all users of the database are to have the listed privileges revoked.

The PASSWORD Statement

```
PASSWORD  password
```

The PASSWORD statement is used to redefine a user's password. It can only be used by a user to redefine his own password.

password	specifies the name of the new password for the currently logged on user. A password can have a maximum length of 20 characters.

CONCURRENCY CONTROL STATEMENTS

ORACLE allows multiple users to concurrently UPDATE the same table in a database. Yet, ORACLE does not require a user to issue locking statements or statements of intent to update. The setting and clearing of locks are the responsibility of ORACLE. No explicit lock requests are required to insure that concurrent UPDATE operations do not read the same row of a table and attempt to write back that row. ORACLE automatically places locks on individual records (rows) in order to synchronize UPDATE operations, thus preventing the updates from overwriting each other.

A user may not wish to operate on data that has been modified by an operation that is still in progress. To accomplish this serialization the user can place his SQL request between BEGIN TRANSACTION and END TRANSACTION statements. In addition, placing several SQL statements inside a transaction causes ORACLE to execute these SQL statements as a "atomic act" without permitting interference (UPDATES to the same tables) by other users during the transaction.

ORACLE's automatic update synchronization locks a record (row of a table) at a time. USER requested transactions lock a table at a time.

Whereas transaction level control will often be desired in UPDATE operations, it is also useful in READ-only situations when it is required that data being retrieved not be subject to modification during the period of retrieval.

The BEGIN TRANSACTION Statement

```
BEGIN TRANSACTION  [tran-id]  ON TABLE  table  | UPDATE |, . . .
                                               | READ   |
```

The BEGIN TRANSACTION statement is used to identify 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.

Transactions may be nested. When transactions are nested, the BEGIN TRANSACTION statements must be numbered beginning with 1.

tran-id	specifies an integer value. Tran-id must be specified when transactions are nested.
table	specifies the name of a table which will be updated or read.
UPDATE	specifies that the table should be locked for all other update and read transactions.
READ	specifies that the table should be locked to update transactions. Read transactions may concurrently access the table.

The END TRANSACTION Statement

END TRANSACTION [tran-id]

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

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

S Q L

PUNCTUATION AND CODING RULES

The following general rules of punctuation apply in writing SQL statements:

Blank Spaces	are used as general purpose delimiters. The number of blank spaces used is optional and will not change the meaning of a SQL statement.
, Comma	is used to separate items in a list.
. Period	is used to separate qualifiers in a qualified name.
; Semicolon	is used to indicate the end of a query block. The semicolon may be omitted in query statements containing only one query block.
: Colon	is used as a general purpose terminator within a SQL statement.
[ ] Square Brackets	are used to establish precedence within logical expressions in WHERE and HAVING clauses.
< > Angle Brackets	are used to enclose sets of literal values.
( ) Parentheses	are used to establish precedence within arithmetic expressions. Parentheses are also used for general purpose enclosure within SQL.

LOGICAL EXPRESSIONS

SQL WHERE and HAVING clauses contain logical expressions. Logical expressions are made up of predicates connected by the boolean operators AND and OR. ORACLE tests fields in a given row in the database to determine if the row satisfies the predicates in the logical expression.

ORACLE allows unknown or null values in the database. Therefore, the evaluation of logical expressions requires the use of three valued logic.

Rows that contain null fields tested by the WHERE clause are assigned an unknown truth value (?). The truth value of a row against the entire logical expression is then evaluated using the three valued logic truth tables depicted below.

AND	T	F	?
T	T	F	?
F	F	F	F
?	?	F	?

OR	T	F	?
T	T	T	T
F	T	F	?
?	T	?	?

NOT	
T	F
F	T
?	?

Only those rows whose overall truth value is true are considered to have satisfied the WHERE clause. If the row's overall truth value is false or unknown the row is disqualified by the WHERE clause.

Null values are ignored in the evaluation of arithmetic expressions.

S Q L

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 ]
```