Exp No. 2 Date : 28-07-2019

INTRODUCTION TO SQL

History of SQL

Dr. E. F. Codd published the paper, "A Relational Model of Data for Large Shared Data Banks", in June 1970 in the Association of Computer Machinery (ACM) journal, Communications of theACM. Codd's model is now accepted as the definitive model for relational database management systems (RDBMS). The language, Structured English Query Language ("SEQUEL") was developed by IBM Corporation, Inc., to use Codd's model. SEQUEL later became SQL (still pronounced "sequel"). In 1979, Relational Software, Inc. (now Oracle Corporation) introduced the first commercially available implementation of SQL. Today, SQL is accepted as the standard RDBMS language.

How SQL Works

The strengths of SQL provide benefits for all types of users, including application programmers, database administrators, managers, and end users. Technically speaking, SQL is a data sublanguage. The purpose of SQL is to provide an interface to a relational database such as Oracle, and all SQL statements are instructions to the database. In this SQL differs from general-purpose programming languages like C and BASIC. Among the features of SQL are the following:

1. It processes sets of data as groups rather than as individual units.
2. It provides automatic navigation to the data.
3. It uses statements that are complex and powerful individually, and that therefore stand alone.

Flow-control statements were not part of SQL originally, but they are found in the recently accepted optional part of SQL, ISO/IEC 9075-5: 1996. Flow-control statements are commonly known as "persistent stored modules" (PSM), and Oracle's PL/SQL extension to SQL is similar to PSM.

Essentially, SQL lets you work with data at the logical level. You need to be concerned with the implementation details only when you want to manipulate the data. For example, to retrieve a set of rows from a table, you define a condition used to filter the rows. All rows satisfying the condition are retrieved in a single step and can be passed as a unit to the user, to another SQL statement, or to an application. You need not deal with the rows one by one, nor do you have to worry about how they are physically stored or retrieved. All SQL statements use the optimizer, a part of Oracle that determines the most efficient means of accessing the specified data. Oracle also provides techniques you can use to make the optimizer perform its job better.

SQL provides statements for a variety of tasks, including:

1. Querying data
2. Inserting, updating, and deleting rows in a table
3. Creating, replacing, altering, and dropping objects
4. Controlling access to the database and its objects
5. Guaranteeing database consistency and integrity

SQL unifies all of the above tasks in one consistent language.

Common Language for All Relational Databases

All major relational database management systems support SQL, so you can transfer all skills you have gained with SQL from one database to another. In addition, all programs written in SQL are portable. They can often be moved from one database to another with very little modification.

Summary of SQL Statements

SQL statements are divided into these categories:

1. Data Definition Language (DDL) Statements
2. Data Manipulation Language (DML) Statements
3. Transaction Control Statements (TCL)
4. Session Control Statement
5. System Control Statement

Managing Tables
A table is a data structure that holds data in a relational database. A table is composed of rows and columns.

A table can represent a single entity that you want to track within your system. This type of a table could represent a list of the employees within your organization, or the orders placed for your company's products.

A table can also represent a relationship between two entities. This type of a table could portray the association between employees and their job skills, or the relationship of products to orders. Within the tables, foreign keys are used to represent relationships.

Creating Tables

To create a table, use the SQL command CREATETABLE.

Syntax:
CREATE TABLE <TABLE NAME>(<FIELD NAME ><DATA TYPE><[SIZE]>,.....)

Altering Tables

Alter a table in an Oracle database for any of the following reasons:

1. To add one or more new columns to the table
2. To add one or more integrity constraints to a table
3. To modify an existing column's definition (datatype, length, default value, and NOTNULL
4. integrity constraint)
5. To modify data block space usage parameters (PCTFREE, PCTUSED)
6. To modify transaction entry settings (INITRANS, MAXTRANS)
7. To modify storage parameters (NEXT, PCTINCREASE, etc.)
8. To enable or disable integrity constraints associated with the table
9. To drop integrity constraints associated with the table

When altering the column definitions of a table, you can only increase the length of an existing column, unless the table has no records. You can also decrease the length of a column in an empty table. For columns of data type CHAR, increasing the length of a column might be a time consuming operation that requires substantial additional storage, especially if the table contains many rows. This is because the CHAR value in each row must be blank-padded to satisfy the new column length.

If you change the datatype (for example, from VARCHAR2 to CHAR), then the data in the column does not change. However, the length of new CHAR columns might change, due to blank-padding requirements.

Altering a table has the following implications:

1. If a new column is added to a table, then the column is initially null. You can add a column with a NOT NULL constraint to a table only if the table does not contain any rows.
2. If a view or PL/SQL program unit depends on a base table, then the alteration of the base table might affect the dependent object, and always invalidates the dependent object.

Privileges Required to Alter a Table
To alter a table, the table must be contained in your schema, or you must have either the ALTER object privilege for the table or the ALTER ANY TABLE system privilege.

Dropping Tables
Use the SQL command DROP TABLE to drop a table. For example, the following statement drops the EMP_TAB table:

If the table that you are dropping contains any primary or unique keys referenced by foreign keys to other tables, and if you intend to drop the FOREIGN KEY constraints of the child tables, then include the CASCADE option in the DROP TABLE command.

Oracle Built-In Datatypes
A datatype associates a fixed set of properties with the values that can be used in a column of a table or in an argument of a procedure or function. These properties cause Oracle to treat values of one datatype differently from values of another datatype. For example, Oracle can add values of sNUMBER datatype, but not values of RAW datatype.

Oracle supplies the following built-in data types:character data types

- CHAR
- NCHAR
- VARCHAR2 and VARC
- NVARCHAR2
- CLOB
- NCLOB
- LONG
    1. NUMBER datatype
    2. DATE datatype
    3. Binary datat ypes
- BLOB
- BFILE
- RAW
- LONG RAW

Another datatype, ROWID, is used for values in the ROWIDpseudocolumn, which represents the unique address of each row in a table.

Table summarizes the information about each Oracle built-in datatype.

Summary of Oracle Built-In Data types

Data Type	Description	Column Length and Default
CHAR (size)	Fixed-length Character data of length size bytes	Fixed for every row in the table (with trailing blanks); byte per row. Consider the character set (one-byte or multibyte) before setting size.
VARCHAR2 (size)	Variable-length Character data	Variable for each row, up to 4000 bytes per row:Consider the character set (one-byte or multibyte)before setting size: A maximum size must be specified.

NCHAR(size)	Fixed Length Character of length size , characters, bytes, depending on the character set	Fixed for every row in the table(with trailing blanks).Coloum size is the number of characters for a fixed width national character set or the number of bytes for a varying width national character set. Maximum size is determined ny the number of bytes required to store one character, with an upper limit of 2000 bytes per row. Default is 1 character or 1 byte, depending on the character set.
NVARCHAR2 (Size0	Variable-length Character data of length size, characters or bytes,depending on National character set: A maximum size must be specified	Variable for each row. Column size is the number of characters for a fixed-width national character set or the number of bytes for a varying-width national character set. Maximum size is determined by the number of byte, depending on the character set.
CLOB	Single-byte character data	
LONG	Variable-length Character data	Variable for each row in the table, up to 2^31 - 1 bytes, or 2 gigabytes, per row. Provided for backward
NUMBER(p, s)	Variable-length Numericdata.: Maximum precision p and/or scale s is 38	Variable for each row. The maximum space required
DATE	Fixed-length date and time data, ranging from Jan. 1, 4712 B.C.E. to Dec. 31, 4712 C.E.	Fixed at 7 bytes for each row in the table. Default format is a string (such as DD-MON-YY) specified by
BLOB	Unstructured binary data	
BFILE	Binary data stored in an external file	
RAW (size)	Variable-length Raw	Variable for each row in the table, up to 2000 bytes per row. A maximum size must be specified. Provided for binary data
LONG RAW	Variable-length Raw binary data	Variable for each row in the table, up to 2^31 - 1 bytes, or 2 gigabytes, per row. Provided for backward

ROWID	Binary data representing row addresses	Fixed at 10 bytes (extended ROWID) or 6 bytes (restricted ROWID) for each row in the table.
MLSLABEL	Trusted Oracle data type	

Using Character Data types

Use the character data types to store alphanumeric data.

1. CHAR and NCHAR data types store fixed-length character strings.
2. VARCHAR2 and NVARCHAR2 data types store variable-length character strings. (The VARCHAR dataty is synonymous with the VARCHAR2 datatype.)
3. CLOB and NCLOB data types store single-byte and multi byte character strings of up to four gigabytes.
4. The LONG datatype stores variable-length character strings containing up to two gigabytes, but with many restrictions.
5. This data type is provided for backward compatibility with existing applications; in general, new applications should use CLOB and NCLOB data types to store large amounts of character data.

When deciding which datatype to use for a column that will store alphanumeric data in a table, consider the following points of distinction:

Space Usage

1. To store data more efficiently, use the VARCHAR2 datatype. The CHAR data type blank-pads and stores trailing blanks up to a fixed column length for all column values, while the VARCHAR2 datatype does not blank-pad or store trailing blanks for column values.
2. Use the CHAR data type when you require ANSI compatibility in comparison semantics (when trailing blanks are not important in string comparisons). Use the VARCHAR2 when trailing blanks are important in string comparisons.

Comparison Semantics

Use the CHAR data type when you require ANSI compatibility in comparison semantics (when trailing blanks are not important in string comparisons). Use the VARCHAR2 when trailing blanks are important in string comparisons.

Future Compatibility

1. The CHAR and VARCHAR2 data types are and will always be fully supported. At this time, the VARCHAR datatype automatically corresponds to the VARCHAR2 datatype and is reserved for future use.

CHAR, VARCHAR2, and LONG data is automatically converted from the database character set to the character set defined for the user session by the NLS_LANGUAGE parameter, where these are different.

Using the NUMBER Datatype

Use the NUMBER datatype to store real numbers in a fixed-point or floating-point format. Numbers using this data type are guaranteed to be portable among different Oracle platforms, and offer up to 38 decimal digits of precision. You can store positive and negative numbers of magnitude 1 x 10^-130 to 9.99...x10^125, as well as zero, in a NUMBER column.

For numeric columns you can specify the column as a floating-point number:

Column_name NUMBER

Or, you can specify a precision (total number of digits) and scale (number of digits to the right of the decimal point):

Column_name NUMBER (<precision>, <scale>)

Although not required, specifying the precision and scale for numeric fields provides extra integrity checking on input. If a precision is not specified, then the column stores values as given. Table shows examples of how data would be stored using different scale factors.

Using the DATE Datatype
Use the DATE datatype to store point-in-time values (dates and times) in a table. The DATE datatype stores the century, year, month, day, hours, minutes, and seconds.

Oracle uses its own internal format to store dates. Date data is stored in fixed-length fields of seven bytes each, corresponding to century, year, month, day, hour, minute, and second.

Date Format
For input and output of dates, the standard Oracle default date format is DD-MON-YY.

For example: '13-NOV-92'

To change this default date format on an instance-wide basis, use the NLS_DATE_FORMAT parameter. To change the format during a session, use the ALTER SESSION statement. To enter dates that are not in the current default date format, use the TO_DATE function with a format mask.

For example:

TO_DATE ('November 13, 1992', 'MONTH DD, YYYY')

If the date format DD-MON-YY is used, then YY indicates the year in the 20th century (for example, 31-DEC-92 is December 31, 1992). If you want to indicate years in any century other than the 20th century, then use a different format mask, as shown above.

Time Format
Time is stored in 24-hour format #HH:MM:SS. By default, the time in a date field is 12:00:00 A.M. (midnight) if no time portion is entered. In a time-only entry, the date portion defaults to the first day of the current month. To enter the time portion of a date, use the TO_DATE function with a format mask indicating the time portion, as in:

INSERT INTO Birthdays_tab (bname, bday) VALUES ('ANNIE',TO_DATE('13-NOV-92 10:56 A.M.','DD-MON-YY HH:MI A.M.'));

To compare dates that have time data, use the SQL function TRUNC if you want to ignore the time component. Use the SQL function SYSDATE to return the system date and time. The FIXED_DATE initialization parameter allows you to set SYSDATE to a constant; this can be useful for testing.