|  |
| --- |
|  |
| Using Basic SELECT statement |
|  |
|  |
|  |
| **Preeti Patel** |
|  |
|  |

Week 1: Writing Basic SQL Select Statements

**SQL Fundamentals**

SQL is short for Structured Query Language and is a standard language for accessing and manipulating relational databases. That is, SQL can be used to create, store, retrieve, update and delete data.

The basic structure of data storage in any database is in the form of ***tables***. A table consists of ***rows*** and ***columns***. A particular row represents a set of data related to a particular entity where as a column depicts the characteristics or attributes. SQL helps in creating, maintaining and managing such tables in a database. Therefore SQL comprises of many different statements to perform various actions on the data. These different SQL statements are characterized as follows:

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

We will be discussing the first two categories; which are discussed with their respective statements later in the notes.

**Data Manipulation Language**

The Data Manipulation Language (DML) is used to retrieve, modify and insert information from or into the database. These are the activities which are performed by every database user on the regular basis. The SQL statements which fall under the DML category are as follows:

1. SELECT Statement
2. INSERT Statement
3. UPDATE Statement
4. DELETE Statement

The Select statement is used to retrieve information from the database whereas Insert statement is used to insert data into the database. The Update and Delete statements modify the information in the database either by changing information or by removing it respectively. This week and in the coming few weeks we will be discussing these statements in detail one-by-one. Let’s start with the Select statement.

**SELECT Statement**

The Select statement is the most commonly used SQL statement. It is used to retrieve information from the database table; this information can be based on the specific requirements of the user.

|  |
| --- |
| **Syntax:**    SELECT [\* | DISTINCT] columnname1 [, columnname2]  FROM tablename1 [, tablename2]  [WHERE condition] [And | or | LIKE condition . . .]  [GROUP BY column-list]  [HAVING “conditions”]  [ORDER BY “condition-list” [ASC | DESC]] |

Above is the complete syntax of the Select statement. It is not always used completely but with the combinations of few clause/keywords mentioned in the syntax. In the syntax:

**SELECT** is a keyword to retrieve one or more columns

**\*** selects all columns

**DISTINCT** supresses duplicate data

**Columnname1…N** represents the specific column name to be retrieved

**FROM tablename1…N** specifies the table name which contains the columns

**WHERE condition** specifies the condition for the selection of data

**GROUP BY** allows multiple columns to be grouped

**HAVING** specifies condition for the GROUP BY clause

**ORDER BY** arrange the output data in ascending/descending order

A simpler syntax for Select statement can be

|  |
| --- |
| SELECT columnname1 [, columnname2] FROM tablename1 [, tablename2] |

The section between the [] are optional and are used when required. The uses of all mentioned keywords/clauses in the syntax are discussed later in the notes. This week we are focusing on simple Select statements and the various ways to use them.

SQL statement are *not case sensitive*, to enhance readability all the keywords are mentioned in uppercase throughout the notes, because they resembles the general English words and can be mistaken with them.

SQL statements can be written on one or more lines but the clauses/keywords cannot be abbreviated or split across the line.

Coming back to our Select statement, two very important clauses that has to be there in the statement to execute at-least a simple query are SELECT and FROM ; because they specifies what information is to be retrieve and from where to retrieve it i.e. table name.

**Capabilities of SQL Select Statements**

A SELECT statement retrieves and display information fetched from the database. This particular statement not only does simple fetching of information but also retrieves information based on user requirements and condition. Using a SELECT statement we can make use of its following capabilities:

* Projection: Using the projection capability we can choose the column that we want from the table. We can choose as few or as many columns as of the table as we require.
* Selection: Using the selection capability we can choose the rows from a table which we require. We can also restrict the rows as per our requirements by applying some condition to the Select statement.
* Joining: Using the joining capability we can create a link between the data stored in two different tables.

These are the capabilities of SQL SELECT statement which we will discuss in detail later in the notes.

**Simple Select Statement**

There are several ways of writing a simple Select statement, let’s take a look at the examples below to understand the different ways of writing them.

|  |  |  |
| --- | --- | --- |
| **Example 1.1: Selecting all columns in a table**   |  | | --- | | **SELECT \***  **FROM Author;** | |  | |
|  |

For the explanation of the SQL statement we are using BOOKS database throughout these notes. The database is available in ***Annexure A*** for your reference.

In example 1.1 the Select statement is displaying ALL the records available in the Author table. The asterisk (\*) is used when we want to display all the available rows and columns in a particular table. If we know the column names of a table we can also use them to retrieve the information. In our Author table we have three columns so a simple select statement can also be written as:

*SELECT Author\_id, Fname, Lname*

*FROM Author;*

This will also produce the similar result which we get in example 1.1. It is a good way of retrieving the information in the case where we don’t have lots of columns in a table or if the column names can be easily recalled. But if we forgot column name, then we can always use asterisk (\*) to display them.

|  |  |  |
| --- | --- | --- |
| **Example 1.2: Selecting specific columns in a table**   |  | | --- | | **SELECT P\_Name, City**  **FROM Publisher;** | |  | |
|  |

There are several occasions when we want some specific data to be displayed instead of the complete table. Example 1.2 is dealing with one such situation. Here we just want the Publishers name and the city where the publisher is located from the Publisher table, although there are more columns present in this table but for the moment we just need these two columns to be displayed.

**Select Statement with Aliases**

Aliases in SQL statements are used to provide some temporary names to table columns in order to make the output more meaningful. In the Publisher table the P\_Name store the value of Publishers name, but for the person who doesn’t know about it will make no meaning. Therefore with example below we can easily assign a name to that column to make it more understandable.

|  |  |  |
| --- | --- | --- |
| **Example 1.3: Assigning alias to a column in a table**   |  | | --- | | **SELECT P\_Name AS “Publisher’s Name”, City**  **FROM Publisher;** | |  | |
|  |

As we can see in the output for the example 1.3 it displays Publisher’s Name despite P\_Name.

If the alias is of single word we can avoid the use of double quotation marks. Similarly we can alias to a table as well. Table aliases are very useful especially if we want to select data from multiple tables, we will see table aliases in Week 4 notes.

**Select Statement with Operators**

Often we want to perform some calculations on our existing data and display it and sometimes we want to display our data in some specified manner, to make it possible with SQL we use operators. A simple Select statement is capable of using various operators. The list of operators is given below:

* Arithmetic Operators
* Concatenation Operator
* Set Operators

1. **Arithmetic Operator:** As the name suggests these operators are used for numeric values. It is contains a set of arithmetic expression which can be used to perform arithmetic calculation on the columns data.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| **+ -** | Unary operators represents positive & negative data items |
| **\*** | Multiplication |
| **/** | Division |
| **+** | Addition |
| **-** | Subtraction |

All the operators have got some priority over each other, in case if have more than one operator present in a Select statement then it will be handled according to its priority i.e. highest to lowest. The unary operators have got the highest position in priority list. Multiplication and division are next; addition and subtraction are at the lowest level.

|  |  |  |
| --- | --- | --- |
| **Example 1.4: Using arithmetic operators on a column in table**   |  | | --- | | **SELECT B\_Code, Title, Price, Price\*10/100 AS “Student Discount”**  **FROM Books;** | |  | |
|  |

As we can see in example 1.4 we are using both alias and operators, because this particular column does not exist in our actual table. Here in this example both the operators are of same priority, in such a case an expression is always evaluated from left-to-right to produce output.

This statement can be written in a different way and will produce same result as for example 1.4. We can do it by putting the part of expression in brackets (). These brackets will force the expression inside it to be executed first. The above statement can be re-written as:

*SELECT B\_Code, Title, Price, (Price\*10)/100 AS “Student Discount”*

*FROM Book;*

1. **Concatenation Operator:** The concatenation operator is used to join two character (text) strings which results into a new character string.

|  |  |  |
| --- | --- | --- |
| **Example 1.5: Using concatenation operator on two columns in table**   |  | | --- | | **SELECT Fname || Lname AS “Author’s Name”**  **FROM Author;** | |  | |
|  |

As we can see the output for example 1.5, the first name and the Last name of the author are combined together to show as a single string. It is done by placing the concatenation operator (||) in between the two columns.

1. **Set Operators:** The set operators combine the result of two queries into a single result. Queries containing the set operators are called combined queries. List of set operators is given below:

|  |  |
| --- | --- |
| **OPERATOR** | **RETURNS** |
| **UNION** | All rows selected by either query |
| **UNION ALL** | All rows selected by either query, including all duplicates. |
| **INTERSECT** | All distinct rows selected by both queries. |
| **MINUS** | All distinct rows selected by the first query but not the second. |

**Select Statement with Literals**

Literals are a value that represents a fixed value. A literal can be a character, a number or a date which can be included in the Select statement. They are printed with each row which is being retrieved. Date and character literals must be enclosed within single quotation marks, whereas it is not required for the numbers.

|  |  |  |
| --- | --- | --- |
| **Example 1.6: Using literal in a table**   |  | | --- | | **SELECT Title || ‘ published by ‘ || Publisher AS “Books Details”**  **FROM Books;** | |  | |
|  |

In example 1.6 the title and the publisher of the books are displayed in this example we have used concatenation operator as well as literals to show that we can use as many combinations of operators to make our output more readable and presentable.

**Select Statement with DISTINCT Keyword**

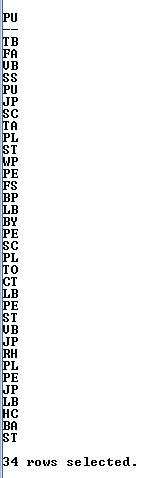
The SQL DISTINCT clause is used together with SQL SELECT to retrieve and display a dataset with unique entries. In simpler words the DISTINCT keyword is used to eliminate any duplicate data entries and make the output more precise and accurate.

For example if we want to display the names of publisher for all the books from the Book table our select statement will be:

*SELECT Publisher*

*FROM Books;*

The output for the above statement will be:



We can see that many of the Publishers name are repeated in the list and the total number of rows selected are also high in number. To eliminate this duplication of records we use DISTINCT keyword, example 1.7 shows the use of DISTINCT keyword and the output we got after executing it.

|  |  |  |
| --- | --- | --- |
| **Example 1.7: Distinct records in a table**   |  | | --- | | **SELECT DISTINCT Publisher**  **FROM Books;** | |  | |
|  |

As we can see now there is no duplication of name all the publisher are listed without any duplication of record and the number of rows selected have reduced considerably as well.

We can specify multiple columns after the DISTINCT keyword. It ensures that all the columns select are unique. Uniqueness is verified against the complete row, not the first column and the result is every unique combination of the column.

**Displaying Table Structure**

We have a fair good idea how we can retrieve and display data from a table in a database. But sometimes we need to know the structure of the table we are using. It is useful to know about the various column names and their data type. To know the structure of a table we use the following syntax:

|  |
| --- |
| **Syntax:**    DESC[RIBE ] tablename |

In the syntax:

**DESCRIBE** Is the keyword which instruct to get the structure of the table.

**Tablename** Is the name of any existing table, which is accessible to user.

We can use either **DESCRIBE** as a whole or we can also use **DESC** to get the structure of the table. This can be used to know the structure of Views, sequences or any synonyms created by user or accessible by the users. We can also see all the tables and views present in our schema by writing the following select statement:

SELECT \* FROM TAB;

|  |  |  |
| --- | --- | --- |
| **Example 1.8: Displaying table structure**   |  | | --- | | **DESCRIBE Books;** | |  | |
|  |

Example 1.8 shows the use of DESCRIBE keyword, as we can see it displays the structure of Book table. Similarly we can find the structure of all the tables available in the database.

**The DUAL Table**

The DUAL table is a dummy table available to all the users in the database. It contains one row and one column. It used to select system variables or to evaluate an expression.

|  |  |  |
| --- | --- | --- |
| **Example 1.9: Using DUAL table**   |  | | --- | | **SELECT SYSDATE, USER**  **FROM dual;** | |  | |
|  |

Example 1.9 shows the use of dual table, it can also be used to evaluate various functions available in SQL.

***i*SQL \*Plus – An Overview**

*i*SQL \*Plus is an Oracle tool that recognizes and submits SQL statements to the Oracle server for execution and contains its own command language. It is a browser- based interface which uses SQL \*Plus processing engine in a three- tier model comprising:

* Client (Web Browser)
* Middle Tier (Application Server)
* Database (Oracle Database)

ORACLE NET

HTTP TCP/IP

Application Server

Oracle Database

Web Browser

Database Tier

Middle Tier

Client Tier

The *i*SQL \*Plus server is located on the same machine as the Application server; the client may or may not be on the same machine. The middle tier coordinates interactions and resources between the client tier and the database tier.

**Features of *i*SQL \*Plus**

* Accessed from a browser
* Accepts ad hoc entry of statements
* Provides online editing for modifying SQL statements
* Controls environmental settings
* Formats query results into basic report
* Accesses local and remote databases

***i*SQL \*Plus Commands Versus SQL Statements**

|  |  |
| --- | --- |
| **SQL** | ***i*SQL \*Plus** |
| * It is a language for communicating with the Oracle server to access data. | * It is an environment which recognizes SQL statements and sends them to the server. |
| * It is based on American National Standards Institute (ANSI) standard SQL. | * It is an Oracle proprietary interface for executing SQL statements |
| * It can manipulate data and table definitions in the database as the previously executed statement is always available in the SQL buffer. | * It does not allow any manipulation of values in the database as SQL \*Plus command are not stored into the buffer. |
| * It does not need a continuation character if the command exceeds one line. | * It needs a dash (-) as a continuation character if the command is longer than one line. |
| * SQL statements are terminated with a semicolon (;) or a blank line. | * SQL \*Plus statement are entered on a single line so they don’t require any termination character. Pressing *ENTER* executes the SQL \*Plus command. |
| * Here keywords cannot be abbreviated | * Here keywords can be abbreviated. |
| * It uses functions to perform some formatting. | * It uses commands to format data. |

\*\*\* Chapter Ends \*\*\*