ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



DATABASE MANAGEMENT SYSTEM

Theory Assignment #8

**Submitted by:**

Linus Dhakal  
013BSCCSIT022

**Submitted to:**

|  |  |
| --- | --- |
| Er. Sanjay Kumar Yadav  Lecturer  Department of Computer Science |  |

**Date of Submission:** September 27, 2015

# Data Definition Language

Data Definition Language (DDL) is a standard for commands that define the different structures in a database. DDL statements create, modify, and remove database objects such as tables, indexes, and users. Common DDL statements are CREATE, ALTER, and DROP.

## Domain Type in SQL

### Character Domain Types in SQL

* CHAR(n) Fixed length character string, with user-specified length n.

• VARCHAR(n) Variable length character strings, with user-specified maximum length n.

• Null values are allowed in all the domain types. Declaring an attribute to be NOT NULL prohibits null values for that attribute.

### Number Domain Types in SQL

• INT (also: INTEGER) Integer (a finite subset of the integers that is machine-dependent).

• SMALLINT Small integer (a machine-dependent subset of the integer domain type). •

DECIMAL(p,d) Fixed point number, with user-specified precision of p digits, with n digits to the right of decimal point.

• FLOAT(n) Floating point number, with user-specified precision of at least n digits.

• REAL (also: DOUBLE PRECISION) Floating point and double-precision floating point numbers, with machine-dependent precision.

### Date Domain Types of SQL

• DATE dates, containing a (4 digit) year, month and date – E.g. DATE ‘2001-7-27’ (ANSI Syntax) – E.g. TO\_DATE(‘01-JUL-27’,’YY-MON-DD’) (Oracle Syntax)

• TIMESTAMP date plus time of day – E.g. TIMESTAMP ‘2001-7-27 09:00:30.75’

## Schema Definition in SQL

A schema is the organization or structure for a database. The activity of data modeling leads to a schema. The term is used in discussing both relational databases and object-oriented databases. The term sometimes seems to refer to a visualization of a structure and sometimes to a formal text-oriented description.

The good thing about schemas is that when you access a table that you own (in your own schema), you do not have to refer to the schema name. For instance, you could refer to your table as either one of the following:

EMPLOYEE\_TBL

USER1.EMPLOYEE\_TBL

# Data Manipulation Language

A data manipulation language (DML) is a family of syntax elements similar to a computer programming language used for selecting, inserting, deleting and updating data in a database.

## The Select Clause

The SELECT clause lets you project your own record types, referencing table fields, functions, arithmetic expressions, etc. The DSL type provides several methods for expressing a SELECT clause:

-- The SELECT clause

SELECT BOOK.ID, BOOK.TITLE

SELECT BOOK.ID, TRIM(BOOK.TITLE)

// Provide a varargs Fields list to the SELECT clause:

Select<?> s1 = create.select(BOOK.ID, BOOK.TITLE);

Select<?> s2 = create.select(BOOK.ID, trim(BOOK.TITLE));

Some commonly used projections can be easily created using convenience methods:

-- Simple SELECTs

SELECT COUNT(\*)

SELECT 0 -- Not a bind variable

SELECT 1 -- Not a bind variable

// Select commonly used values

Select<?> select1 = create.selectCount();

Select<?> select2 = create.selectZero();

Select<?> select2 = create.selectOne();

See more details about functions and expressions in the manual's section about [Column expressions](http://www.jooq.org/doc/3.0/manual/sql-building/column-expressions/)

[The SELECT DISTINCT clause](http://www.jooq.org/doc/3.0/manual/sql-building/sql-statements/select-statement/select-clause/" \l "N16028)

The DISTINCT keyword can be included in the method name, constructing a SELECT clause

SELECT DISTINCT BOOK.TITLE

Select<?> select1 = create.selectDistinct(BOOK.TITLE);

## The where clause

The SQL WHERE clause is used to specify a condition while fetching the data from single table or joining with multiple tables.

If the given condition is satisfied then only it returns specific value from the table. You would use WHERE clause to filter the records and fetching only necessary records.

The WHERE clause is not only used in SELECT statement, but it is also used in UPDATE, DELETE statement, etc., which we would examine in subsequent chapters.

## Syntax:

The basic syntax of SELECT statement with WHERE clause is as follows:

SELECT column1, column2, columnN

FROM table\_name

WHERE [condition]

You can specify a condition using comparison or logical operators like >, <, =, LIKE, NOT, etc. Below examples would make this concept clear.

## Example:

Consider the CUSTOMERS table having the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Following is an example which would fetch ID, Name and Salary fields from the CUSTOMERS table where salary is greater than 2000:

SQL> SELECT ID, NAME, SALARY

FROM CUSTOMERS

WHERE SALARY > 2000;

This would produce the following result:

+----+----------+----------+

| ID | NAME | SALARY |

+----+----------+----------+

| 4 | Chaitali | 6500.00 |

| 5 | Hardik | 8500.00 |

| 6 | Komal | 4500.00 |

| 7 | Muffy | 10000.00 |

+----+----------+----------+

Following is an example, which would fetch ID, Name and Salary fields from the CUSTOMERS table for a customer with name Hardik. Here, it is important to note that all the strings should be given inside single quotes ('') where as numeric values should be given without any quote as in above example:

SQL> SELECT ID, NAME, SALARY

FROM CUSTOMERS

WHERE NAME = 'Hardik';

This would produce the following result:

+----+----------+----------+

| ID | NAME | SALARY |

+----+----------+----------+

| 5 | Hardik | 8500.00 |

+----+----------+----------+

## The From Caluse

The SQL FROM clause is used to list the tables and any joins required for the SQL statement.

## SYNTAX

The syntax for the FROM Clause in SQL is:

FROM table1

[ { INNER JOIN

| LEFT [OUTER] JOIN

| RIGHT [OUTER] JOIN

| FULL [OUTER] JOIN } table2

ON table1.column1 = table2.column1 ]

### Parameters or Arguments

**table1 and table2**

These are the tables used in the SQL statement. The two tables are joined based on table1.column1 = table2.column1.

**Note:**

* When using the FROM clause in a SQL statement, there must be at least one table listed in the FROM clause.
* If there are two or more tables listed in the SQL FROM clause, these tables are generally joined using [INNER or OUTER joins](http://www.techonthenet.com/sql/joins.php).

## EXAMPLE - WITH ONE TABLE

It is difficult to explain the syntax for the SQL FROM clause, so let's look at some examples.

We'll start by looking at how to use the FROM clause with only a single table.

For example:

SELECT \*

FROM suppliers

WHERE city = 'Newark'

ORDER BY city DESC;

In this SQL FROM clause example, we've used the FROM clause to list the table called suppliers. There are no joins performed since we are only using one table.

The SQL FROM clause is used to list the tables and any joins required for the SQL statement.

## SYNTAX

The syntax for the FROM Clause in SQL is:

FROM table1

[ { INNER JOIN

| LEFT [OUTER] JOIN

| RIGHT [OUTER] JOIN

| FULL [OUTER] JOIN } table2

ON table1.column1 = table2.column1 ]

### Parameters or Arguments

**table1 and table2**

These are the tables used in the SQL statement. The two tables are joined based on table1.column1 = table2.column1.

**Note:**

* When using the FROM clause in a SQL statement, there must be at least one table listed in the FROM clause.
* If there are two or more tables listed in the SQL FROM clause, these tables are generally joined using [INNER or OUTER joins](http://www.techonthenet.com/sql/joins.php).

## EXAMPLE - WITH ONE TABLE

It is difficult to explain the syntax for the SQL FROM clause, so let's look at some examples.

We'll start by looking at how to use the FROM clause with only a single table.

For example:

SELECT \*

FROM suppliers

WHERE city = 'Newark'

ORDER BY city DESC;

In this SQL FROM clause example, we've used the FROM clause to list the table called suppliers. There are no joins performed since we are only using one table.

## EXAMPLE - TWO TABLES WITH INNER JOIN

Let's look at how to use the FROM clause with two tables and an [INNER JOIN](http://www.techonthenet.com/sql/joins.php).

For example:

SELECT products.product\_name, inventory.quantity

FROM products

INNER JOIN inventory

ON products.product\_id = inventory.product\_id

WHERE products.product\_id < 1000;

This SQL FROM clause example uses the FROM clause to list two tables - products and inventory. And we are using the FROM clause to specify an INNER JOIN between the products and inventory tables based on the product\_id column in both tables.

# SQL RENAME Statement

With RENAME statement you can rename a table.

Some of the relational database management system (RDBMS) does not support this command, because this is not standardizing statement.

For example renaming a table through MS SQL Server you must use storage procedure SP\_RENAME.

### ****Syntax for SQL RENAME is:****

RENAME TABLE {tbl\_name} TO {new\_tbl\_name};

**Where {tbl\_name} table that exists in the current database, and {new\_tbl\_name} is new table name.**

**Tuple Variables**

1. Tuple variables can be used in SQL, and are defined in the **from** clause:
2. **select distinct** *cname, T.loan#*
3. **from** *borrower* ***as*** *S, loan* ***as*** *T*
4. **where** *S.loan# = T.loan#*

Note: The keyword **as** is optional here.

1. These variables can then be used throughout the expression. Think of it as being something like the rename operator.

Finds the names of all branches that have assets greater than at least one branch located in Burnaby.

**select distinct** *T.bname*

**from** *branch S, branch T*

**where** *S.bcity=``Burnaby''* **and** *T.assets > S.assets*

# String Operation

QL string functions are used primarily for string manipulation. The following table details the important string functions:

|  |  |
| --- | --- |
| **Name** | **Description** |
| [**ASCII()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_ascii) | Returns numeric value of left-most character |
| [**BIN()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_bin) | Returns a string representation of the argument |
| [**BIT\_LENGTH()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_bit-length) | Returns length of argument in bits |
| [**CHAR\_LENGTH()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_char-length) | Returns number of characters in argument |
| [**CHAR()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_char) | Returns the character for each integer passed |
| [**CHARACTER\_LENGTH()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_character-length) | A synonym for CHAR\_LENGTH() |
| [**CONCAT\_WS()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_concat-ws) | Returns concatenate with separator |
| [**CONCAT()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_concat) | Returns concatenated string |
| [**CONV()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_conv) | Converts numbers between different number bases |
| [**ELT()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_elt) | Returns string at index number |
| [**EXPORT\_SET()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_export-set) | Returns a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string |
| [**FIELD()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_field) | Returns the index (position) of the first argument in the subsequent arguments |
| [**FIND\_IN\_SET()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_find-in-set) | Returns the index position of the first argument within the second argument |
| [**FORMAT()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_format) | Returns a number formatted to specified number of decimal places |
| [**HEX()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_hex) | Returns a string representation of a hex value |
| [**INSERT()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_insert) | Inserts a substring at the specified position up to the specified number of characters |
| [**INSTR()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_instr) | Returns the index of the first occurrence of substring |
| [**LCASE()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_lcase) | Synonym for LOWER() |
| [**LEFT()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_left) | Returns the leftmost number of characters as specified |
| [**LENGTH()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_length) | Returns the length of a string in bytes |
| [**LOAD\_FILE()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_load-file) | Loads the named file |
| [**LOCATE()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_locate) | Returns the position of the first occurrence of substring |
| [**LOWER()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_lower) | Returns the argument in lowercase |
| [**LPAD()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_lpad) | Returns the string argument, left-padded with the specified string |
| [**LTRIM()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_ltrim) | Removes leading spaces |
| [**MAKE\_SET()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_make-set) | Returns a set of comma-separated strings that have the corresponding bit in bits set |
| [**MID()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_mid) | Returns a substring starting from the specified position |
| [**OCT()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_oct) | Returns a string representation of the octal argument |
| [**OCTET\_LENGTH()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_octet-length) | A synonym for LENGTH() |
| [**ORD()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_ord) | If the leftmost character of the argument is a multi-byte character, returns the code for that character |
| [**POSITION()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_position) | A synonym for LOCATE() |
| [**QUOTE()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_quote) | Escapes the argument for use in an SQL statement |
| [**REGEXP**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#operator_regexp) | Pattern matching using regular expressions |
| [**REPEAT()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_repeat) | Repeats a string the specified number of times |
| [**REPLACE()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_replace) | Replaces occurrences of a specified string |
| [**REVERSE()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_reverse) | Reverses the characters in a string |
| [**RIGHT()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_right) | Returns the specified rightmost number of characters |
| [**RPAD()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_rpad) | Appends string the specified number of times |
| [**RTRIM()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_rtrim) | Removes trailing spaces |
| [**SOUNDEX()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_soundex) | Returns a soundex string |
| [**SOUNDS LIKE**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#operator_sounds-like) | Compares sounds |
| [**SPACE()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_space) | Returns a string of the specified number of spaces |
| [**STRCMP()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_strcmp) | Compares two strings |
| [**SUBSTRING\_INDEX()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_substring-index) | Returns a substring from a string before the specified number of occurrences of the delimiter |
| [**SUBSTRING(), SUBSTR()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_substring) | Returns the substring as specified |
| [**TRIM()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_trim) | Removes leading and trailing spaces |
| [**UCASE()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_ucase) | Synonym for UPPER() |
| [**UNHEX()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_unhex) | Converts each pair of hexadecimal digits to a character |
| [**UPPER()**](http://www.tutorialspoint.com/sql/sql-string-functions.htm#function_upper) | Converts to uppercase |

## Ordering the Display of Tuples

1. SQL allows the user to control the order in which tuples are displayed.
   * **order by** makes tuples appear in sorted order (ascending order by default).
   * **desc** specifies descending order.
   * **asc** specifies ascending order.
2. **select** \*
3. **from** *loan*
4. **order by** *amount* **desc**, *loan#* **asc**

Sorting can be costly, and should only be done when needed.

Duplicate Tuples

Microsoft SQL Server tables should never contain duplicate rows, nor non-unique primary keys. For brevity, we will sometimes refer to primary keys as "key" or "PK" in this article, but this will always denote "primary key." Duplicate PKs are a violation of entity integrity, and should be disallowed in a relational system. SQL Server has various mechanisms for enforcing entity integrity, including indexes, UNIQUE constraints, PRIMARY KEY constraints, and triggers.  
  
Despite this, under unusual circumstances duplicate primary keys may occur, and if so they must be eliminated. One way they can occur is if duplicate PKs exist in non-relational data outside SQL Server, and the data is imported while PK uniqueness is not being enforced. Another way they can occur is through a database design error, such as not enforcing entity integrity on each table.

There are various times when we need to find duplicate records in SQL Server. It is possible to find duplicates using **DISTINCT, ROW NUMBER as well as the GROUP BY** approach.

Duplicate records can create problems sometimes when displaying reports or performing a Multiple Insert update. Finding duplicate records in a database needs further investigation. In some cases, duplicate records are positive, but it all depends on the data and the database design as well.

For example, if a customer has ordered the same product twice on the same date with the the same shipping and billing address, then this may result in a duplicate record.

Let us create a table **Customer** with First Name, Last Name, and Mobile Number fields.

CREATE TABLE CUSTOMER

(

FirstName VARCHAR(50),

LastName VARCHAR(50),

MobileNo VARCHAR(15)

);

INSERT INTO CUSTOMER VALUES ('Niraj','Yadav',989898);

INSERT INTO CUSTOMER VALUES ('Chetan','Gadodia',959595);

INSERT INTO CUSTOMER VALUES ('Chetan','Gadodia',959595);

INSERT INTO CUSTOMER VALUES ('Atul','Kokam',42424242);

INSERT INTO CUSTOMER VALUES ('Atul','Kokam',42424242);

INSERT INTO CUSTOMER VALUES ('Vishal','Parte',9394453);

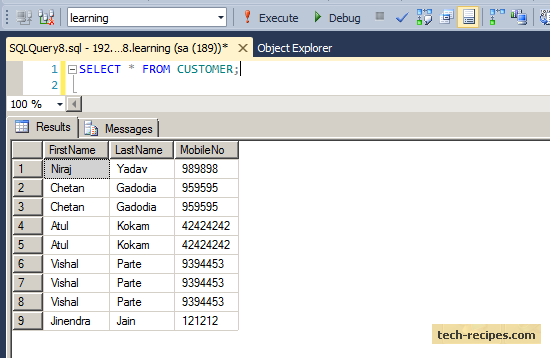
INSERT INTO CUSTOMER VALUES ('Vishal','Parte',9394453);

INSERT INTO CUSTOMER VALUES ('Vishal','Parte',9394453);

INSERT INTO CUSTOMER VALUES ('Jinendra','Jain',121212);

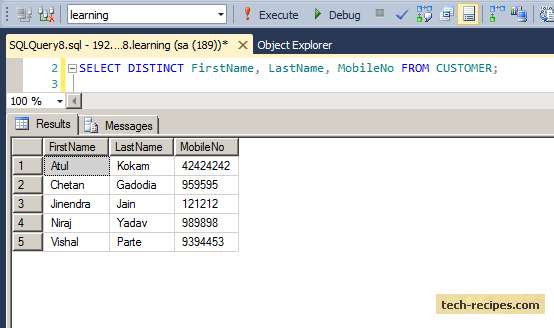
INSERT INTO CUSTOMER VALUES ('Jinendra','Jain',121212);

SELECT \* FROM CUSTOMER;



Using the **DISTINCT**approach, we can quickly get unique rows in a table.

SELECT DISTINCT FirstName, LastName, MobileNo FROM CUSTOMER;



However, this does not show how many times a row has been duplicated. Using the GROUP BY approach, we can find this.

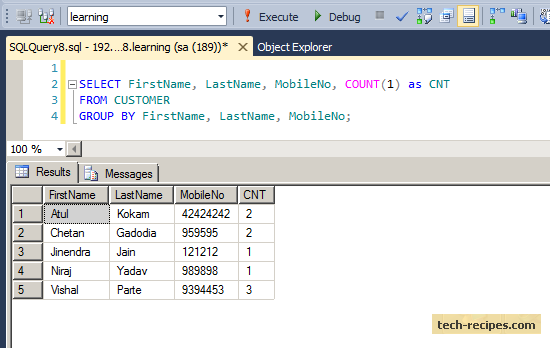
**Finding Duplicates Using GROUP BY**

Adding grouping and a counting field to our display of FirstName, LastName and MobileNo combination shows how many times each customer’s name appears.

SELECT FirstName, LastName, MobileNo, COUNT(1) as CNT

FROM CUSTOMER

GROUP BY FirstName, LastName, MobileNo;



**GROUP BY will show just one record for each combination of FirstName, LastName and MobileNo.**

The count CNT shows how many times the row has been duplicated.  
CNT = 1 indicates that row appears only once.

Let us filter out using the **Having clause** to exclude rows that appear only once.

SELECT FirstName, LastName, MobileNo, COUNT(1) as CNT

FROM CUSTOMER

GROUP BY FirstName, LastName, MobileNo

HAVING COUNT(1) > 1;

