ST.XAVIER’S COLLEGE

**Maitighar, Kathmandu**



**DBMS LAB ASSIGNMENT**

**Submitted By:**

**Narayan Panthi**

013BSCCSIT024

**Submitted To:**

**Er. Sanjay Kumar Yadav** ………..…………….

**Lecturer**

**Department of Computer Science**

**Date of Submission:- *2015 -*** *\_\_****\_-*** *\_\_\_*

# Data Definition Language

Data Definition Language (DDL) is a vocabulary used to define data structures in SQL Server .

It 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 Types in SQL

1. The SQL-92 standard supports a variety of built-in domain types:
   * **char**(n) (or **character**(n)): fixed-length character string, with user-specified length.
   * **varchar**(n) (or **character varying**): variable-length character string, with user-specified maximum length.
   * **int** or **integer**: an integer (length is machine-dependent).
   * **smallint**: a small integer (length is machine-dependent).
   * **numeric**(*p, d*): a fixed-point number with user-specified precision, consists of *p* digits (plus a sign) and *d* of *p* digits are to the right of the decimal point. E.g., **numeric**(*3, 1*) allows 44.5 to be stored exactly but not 444.5.
   * **real** or **double precision**: floating-point or double-precision floating-point numbers, with machine-dependent precision.
   * **float**(n): floating-point, with user-specified precision of at least *n* digits.
   * **date**: a calendar date, containing four digit year, month, and day of the month.
   * **time**: the time of the day in hours, minutes, and seconds.
2. SQL-92 allows arithmetic and comparison operations on various numeric domains, including, **interval** and *cast* (*type coercion*) such as transforming between *smallint* and *int*. It considers strings with different length are compatible types as well.
3. SQL-92 allows **create domain** statement, e.g.,

**create domain** *person-name* **char**(20)

# DATA MANIPULATION LANGUAGE

* 1. **THE SELECT CLAUSE**

The DISTINCT keyword can be used to return only distinct (different) values.

**SQL SELECT DISTINCT Syntax:**

SELECT DISTINCT *column\_name*,*column\_name*  
FROM *table\_name*;

* 1. **THE WHERE CLAUSE**

The WHERE clause is used to extract only those records that fulfill a specified criterion.

**SQL WHERE Syntax:**

SELECT *column\_name*,*column\_name*  
FROM *table\_name*  
WHERE *column\_name operator value*;

**SELECT/FROM/WHERE**

* **SELECT** − This is one of the fundamental query command of SQL. It is similar to the projection operation of relational algebra. It selects the attributes based on the condition described by WHERE clause.
* **FROM** − This clause takes a relation name as an argument from which attributes are to be selected/projected. In case more than one relation names are given, this clause corresponds to Cartesian product.
* **WHERE** − This clause defines predicate or conditions, which must match in order to qualify the attributes to be projected.
  + - **The rename clause**

SQL provides a way of renaming the attributes of a result relation. It uses the **as** clause, taking the form:

*old-name* **as** *new-name*

The ‘**as’** clause is particularly useful when renaming relations. One reason to rename a relation is to replace a long relation name with a shortened version that is more convenient to use elsewhere in the query. To illustrate, we rewrite the query “For all instructors in the university who have taught some course, find their names and the course ID of all courses they taught.”

**select** *T*.*name*, *S*.*course id*

**from** *instructor* **as** *T*, *teaches* **as** *S*

**where** *T*.*ID*= *S*.*ID*;

Another reason to rename a relation is a case where we wish to compare tuples in the same relation. We then need to take the Cartesian product of a relation with itself and, without renaming, it becomes impossible to distinguish one tuple from the other. Suppose that we want to write the query “Find the names of all instructors whose salary is greater than at least one instructor in the Biology department.”We can write the SQL expression:

**Select distinct** *T*.*name*

**from** *instructor* **as** *T*, *instructor* **as** *S*

**where** *T.salary > S.salary* **and** *S.dept name* = ’Biology’;

* + - **Tuple Variable**

Tuple variables can be used in SQL, and are defined in the **from** clause:

**select distinct** *cname, T.loan#*

**from** *borrower* ***as*** *S, loan* ***as*** *T*

**where** *S.loan# = T.loan#*

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*