ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

**Maitighar, Kathmandu**

****

**Database Management System**

**Theory Assignment #8**

**SUBMITTED BY:**

Siddhant Rimal

013BSCCSIT039

**SUBMITTED TO**

|  |  |
| --- | --- |
| **Er. Sanjay Kr. Yadav**  **( Lecturer )** |  |
| **Department of Computer Science** | |

Date of Submission:

8.1. 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. The Data Definition Language (DDL) is used to create and destroy databases and database objects. These commands will primarily be used by database administrators during the setup and removal phases of a database project.

**8.1.1. Domain Type in SQL**

Use of domains is an effective aid to design and implement consistent databases. Domains insure that all attributes defined by a domain share the same data type and value constraints.

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)

**8.1.2. Schema Definition in SQL**

SQL-Schema Statements provide maintenance of catalog objects for a schema -- tables, views and privileges. This subset of SQL is also called the Data Definition Language for SQL. There are 6 SQL-Schema Statements:

**8.1.3. CREATE TABLE Statement**

The CREATE TABLE Statement creates a new base table. It adds the table description to the catalog. A base table is a logical entity with persistence. The logical description of a base table consists of:

* Schema -- the logical database *schema* the table resides in
* Table Name -- a name unique among tables and views in the Schema
* Column List -- an ordered list of column declarations (name, data type)
* Constraints -- a list of constraints on the contents of the table

The CREATE TABLE Statement has the following general format:

**CREATE TABLE table-name   
({column-descr | constraint} [,{column-descr | constraint}]...)**

*table-name* is the new name for the table. *column-descr* is a column declaration. *constraint* is a *table* constraint.

**8.1.4. CREATE VIEW Statement**

The CREATE VIEW statement creates a new database view. A view is effectively a SQL query stored in the catalog. The CREATE VIEW has the following general format:

**CREATE VIEW view-name [ ( column-list ) ] AS**

**query-1[WITH [CASCADED|LOCAL] CHECK OPTION ]**

*view-name* is the name for the new view. *column-list* is an optional list of names for the columns of the view, comma separated. *query-1* is any SELECT statement without an ORDER BY clause. The optional WITH CHECK OPTION clause is a constraint on *updatable* views.

*column-list* must have the same number of columns as the select list in *query-1*. If *column-list* is omitted, all items in the select list of *query-1* must be named.

**8.1.5. DROP TABLE Statement**

The DROP TABLE Statement removes a previously created table and its description from the catalog. It has the following general format:

**DROP TABLE table-name {CASCADE|RESTRICT}**

*table-name* is the name of an existing base table in the current schema. The CASCADE and RESTRICT specifiers define the disposition of other objects dependent on the table.

**8.1.6. DROP VIEW Statement**

The DROP VIEW Statement removes a previously created view and its description from the catalog. It has the following general format:

**DROP VIEW view-name {CASCADE|RESTRICT}**

*view-name* is the name of an existing view in the current schema. The CASCADE and RESTRICT specifiers define the disposition of other objects dependent on the view.

**8.1.7. GRANT Statement**

The GRANT Statement grants access privileges for database objects to other users. It has the following general format:

**GRANT privilege-list ON [TABLE] object-list TO user-list**

*privilege-list* is either ALL PRIVILEGES or a comma-separated list of properties: SELECT, INSERT, UPDATE, DELETE. *object-list* is a comma-separated list of table and view names. *user-list* is either PUBLIC or a comma-separated list of user names.

**8.1.7. REVOKE Statement**

The REVOKE Statement revokes access privileges for database objects previously granted to other users. It has the following general format:

**REVOKE privilege-list ON [TABLE] object-list FROM user-list**

The REVOKE Statement revokes each privilege in *privilege-list* for each object (table) in *object-list* from each user in *user-list*. All privileges must have been previously granted.

The user-list may specify PUBLIC. This must apply to a previous GRANT TO PUBLIC.

8.2. Data Manipulation Language

**8.2.1. The Select Clause**

SQL **SELECT** statement is used to fetch the data from a database table which returns data in the form of result table. These result tables are called result-sets.

The basic syntax of SELECT statement is as follows:

**SELECT column1, column2, columnN FROM table\_name;**

**OR,**

**SELECT \* FROM table\_name;**

Example:

**SELECT ID, NAME FROM CUSTOMERS;**

**8.2.2. 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 statements.

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

**SELECT column1, column2, columnN FROM table\_name WHERE [condition]**

Example:

**SELECT ID, NAME, SALARY**

**FROM CUSTOMERS**

**WHERE NAME = 'Hardik';**

**Example**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Name | Address | Phone | Salary |
| 1 | Ram | Kathmandu | 553344 | 5000 |
| 2 | Shyam | Pokhara | 667788 | 3000 |
| 3 | Hari | Bhaktapur | 445566 | 7000 |
| 4 | Bir | Chitwan | 223344 | 2000 |

SQL Select query:  **SELECT id, name FROM Info WHERE salary > 4000**

The query above is processed and following result appears:

|  |  |
| --- | --- |
| Id | Name |
| 1 | Ram |
| 3 | Hari |

**8.2.3. The From Clause**

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

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 ]**

Example:

**SELECT \* FROM PENDING\_SALES JOIN FINISHED\_SALES;**

**8.2.4. The Rename Operation**

This statement renames one or more tables. The rename operation is done atomically, which means that no other session can access any of the tables while the rename is running.

**RENAME TABLE tbl\_name TO new\_tbl\_name [, tbl\_name2 TO new\_tbl\_name2] ...**

Example:

**RENAME TABLE old\_table TO new\_table;**

**8.2.5. Tuple Variable**

Tuple variables can be used in SQL, and are defined in the **from** clause. These variables can then be used throughout the expression.

The basic syntax is:

**SELECT column\_name AS alias\_name**

**FROM table\_name**

**WHERE [condition];**

Example:

**SELECT C.ID, C.NAME, C.AGE, O.AMOUNT**

**FROM CUSTOMERS AS C, ORDERS AS O**

**WHERE C.ID = O.CUSTOMER\_ID;**

**8.2.6. String Operations**

SQL 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 |

**8.2.7. Ordering the Display of Tuples**

The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns. Some database sorts query results in ascending order by default.

The basic syntax of ORDER BY clause is as follows:

**SELECT column-list**

**FROM table\_name**

**[WHERE condition]**

**[ORDER BY column1, column2, .. columnN] [ASC | DESC];**

Example:

**SELECT \* FROM CUSTOMERS**

**ORDER BY NAME, SALARY;**

**8.2.8. Duplicate Tuples**

The SQL **DISTINCT** keyword is used in conjunction with SELECT statement to eliminate all the duplicate records and fetching only unique records.

The basic syntax of DISTINCT keyword to eliminate duplicate records is as follows:

**SELECT DISTINCT column1, column2,.....columnN**

**FROM table\_name**

**WHERE [condition]**

Example:

**SELECT DISTINCT SALARY FROM CUSTOMERS ORDER BY SALARY;**