Compiler Design Lab
Phase-I Report

***Language*: SQL**

Team Members

K.Pranay - AP19110010349
K.Smaran - AP19110010350
M.Gayathri - AP19110010352
E.Sravanthi - AP19110010354
K.Padmini - AP19110010381
A.Nanda Kishore - AP19110010391
K.Bineeth Kumar - AP19110010396

DATA TYPES AVAILABLE IN LANGUAGE:

CHAR or char - The CHAR data type stores character data in a fixed-length field. Data can be a string of single-byte or multibyte letters, numbers, and other characters that are supported by the code set of your database locale. The size of a CHAR column is byte-based, not character-based. For example, if you define a CHAR column as CHAR(10), the column has a fixed length of 10 bytes, not 10 characters.

Example:

```
CREATE TABLE Student(Name VARCHAR(30), Gender CHAR(6));
INSERT into Student VALUES('Herry', 'Male');
INSERT into Student VALUES('Mahi', 'Female');
SELECT LENGTH(Gender) FROM Student;
```

INT or int - The int data type is the primary integer data type in SQL Server. The big int data type is intended for use when integer values might exceed the range that is supported by the int data type. The INTEGER data type accepts numeric values with an implied scale of zero. It stores any integer value between the range 2^ -31 and 2^31 -1. Attempting to assign values outside this range causes an error. If you assign a numeric value with a precision and scale to an INTEGER data type, the scale portion truncates, without rounding.

Example:

```
CREATE TABLE Products (
   ProductNumber INT,
);
```

FLOAT or float - It is used for specifying a floating-point number.

Example:

```
CREATE TABLE Products (
   ProductPrice FLOAT,
);
```

BIT or bit - SQL Server bit data type is an integer data type that can take only one of these values: 0, 1, NULL. With regard to the storage, if there are less than 9 columns of the bit data in the table, they are stored as 1 byte. If there are 9 to 16 such columns, they consume 2 bytes and so on. Thus, SQL Server optimizes the storage of columns of the bit data type. Additionally, string values TRUE and FALSE can be converted to 1 and 0 corresponding to bit values.

Example:

```
CREATE TABLE Products (
   [ProductName] varchar(20),
   [Available] BIT
);
GO
INSERT INTO Products (productname,available) values('A',1)
INSERT INTO Products (productname,available) values('B',0)
GO
SELECT * FROM Products
```

DOUBLE or double - It is a normal size floating-point number. Its size parameter specifies the total number of digits.

Example:

```
CREATE TABLE Products (
   ProductTotalPrice DOUBLE,
);
```

DATE or date - The date is the SQL Server data type that we use to store Date. It stores the Date without time & without time zone information.The date data type in SQL helps us specify the date in a format. Let’s say if we want to store the date, 2 January 2019, then first we will give the year which would be 2 0 1 9, then the month which would be 0 1, and finally, the day which would be 0 2.

Example:

```
SELECT * FROM Orders WHERE OrderDate='2008-11-11'
```

TIMESTAMP or timestamp - It is used to specify the timestamp. Its value is stored as the number of seconds since the Unix epoch(‘1970-01-01 00:00:01’ UTC). The timestamp is a data type and function in Standard SQL that lets us store and work with both date and time data values, usually without specified time zones. The timestamp has a storage size of 8 bytes that can accept date and time values ranging from 4713 BC and 294276 AD and provides a resolution of 1 microsecond or 14 digits. Some SQL databases allow for customization of a timestamp data type where we can specify a timezone for the timestamp so that every time the database is used in a different timezone, it shows and accepts the corresponding date and time.

Example:

```
SELECT TIMESTAMP("2017-07-23",  "13:10:11");
```

VARCHAR(size) - A VARIABLE length string (can contain letters, numbers, and special characters). The size parameter specifies the maximum column length in characters - can be from 0 to 65535.

Example:

```
CREATE TABLE Products (
   ProductCode varchar(10),
);
```

SYNTAX OF VARIABLE DECLARATION IF YOUR LANGUAGE NEEDS PRE-DECLARATION BEFORE ITS USE.OTHERWISE, THE ASSUMPTIONS:

SQL is a database language. In this language, we'll be working with tables a lot more. Creating a simple table involves labeling the table, creating its columns, and specifying the data type for each column.

Syntax for creating a new table:

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

The column parameters define the names of the table's columns. The data type parameter determines the type of data that can be stored in the column (e.g. varchar, integer, date, etc.).

The keyword CREATE TABLE tells the database system what you want to perform. In this case, you need to create a new table. The CREATE TABLE command is followed by the table's unique name or identifier.

The list defines each column and tells us what sort of data type it is will be in the brackets.

Using the combination of The statements CREATE TABLE and SELECT we can create a copy of an existing table.

Example:

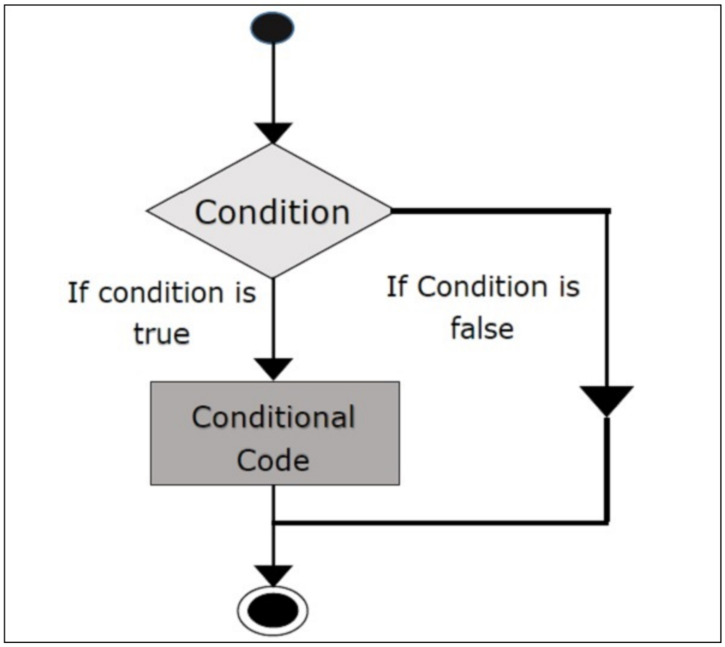
```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

ONE DECISION-MAKING STATEMENT:

1) *Using IF Statement*

```
1 SELECT Age,
2 IF( age>20, "Mature","Immature")
3 As Result
4 FROM student;
```

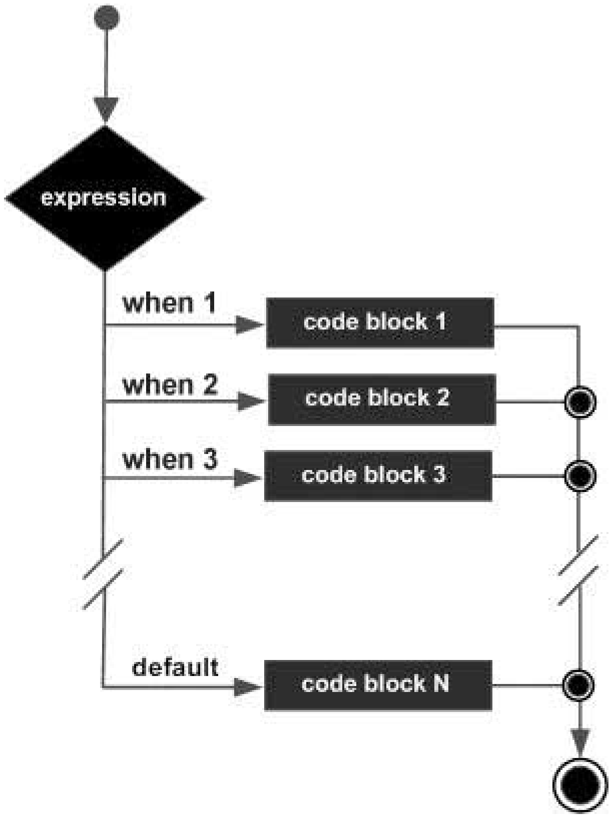
Line 1 and 4: Selects “Age” column from a table called “student”
Line 2: If Age value is greater than 20 then it returns as “Mature” else returns as “Immature”
Line 3: The returned value will be added in a new column called “Result” in the student table.



2) Using CASE Statement:

```
SELECT OrderID, Quantity,
CASE
```

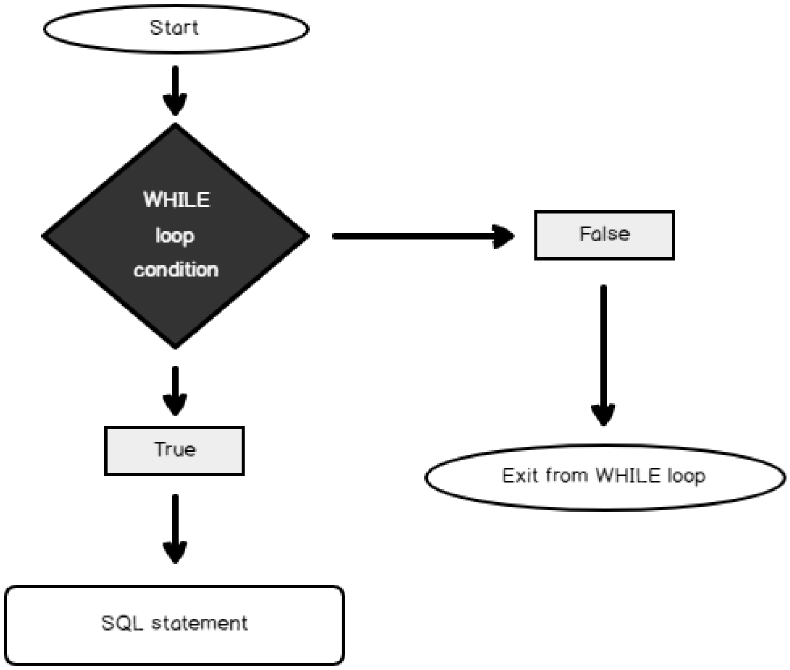
```
WHEN Quantity > 30 THEN 'The quantity is greater than 30'
WHEN Quantity = 30 THEN 'The quantity is 30'
ELSE 'The quantity is under 30'
END AS QuantityText
FROM OrderDetails;
```



AT LEAST TWO ITERATIVE STATEMENTS:

Syntax:

```
WHILE condition
BEGIN
{...statements...}
END
```



1)Using While Loop

Example:

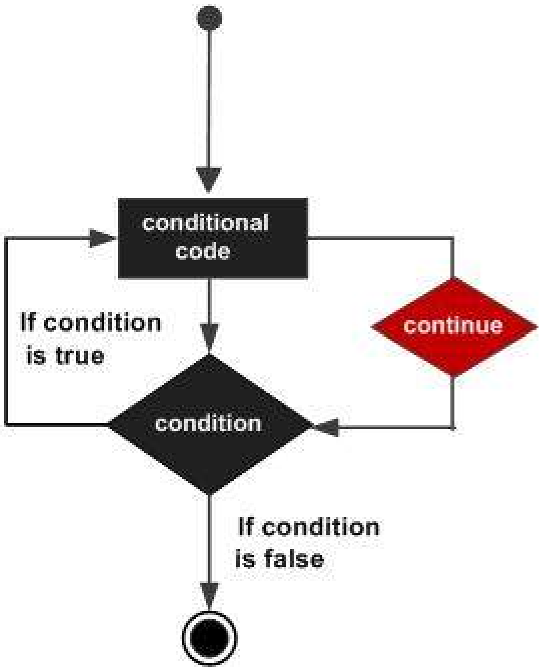
```
DECLARE @count INT;
SET @count = 1;
WHILE @count<= 5
BEGIN
   PRINT @count
   SET @count = @count + 1;
END;
```

2)While loop with continue condition

```
DECLARE @count INT;
DECLARE @mod INT;
SET @count = 1;

WHILE @count<= 10
BEGIN
```

```
   set @mod =  @count % 2
   IF @mod = 1
    BEGIN
    SET @count = @count + 1;
       CONTINUE
    END
   PRINT @count
   SET @count = @count + 1;
END;
```

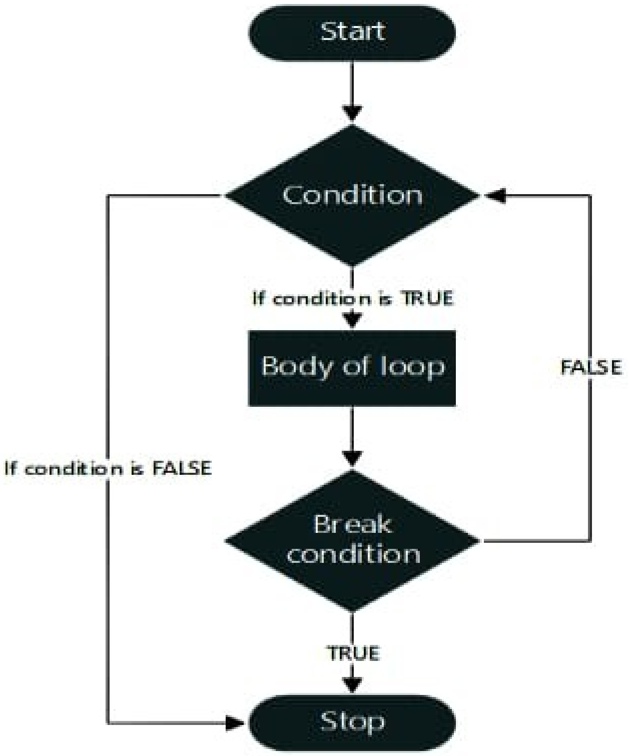


3)While loop with break condition:

```
DECLARE @count INT;
SET @count = 1;

WHILE @count<= 10
BEGIN
          IF @count > 5
BEGIN
         BREAK
```

```
END
PRINT @count
SET @count = @count + 1;
END;
```



CFG FOR AT LEAST FIVE CONSTRUCTS IN YOUR LANGUAGE:

1) CFG for creating tables in a database:

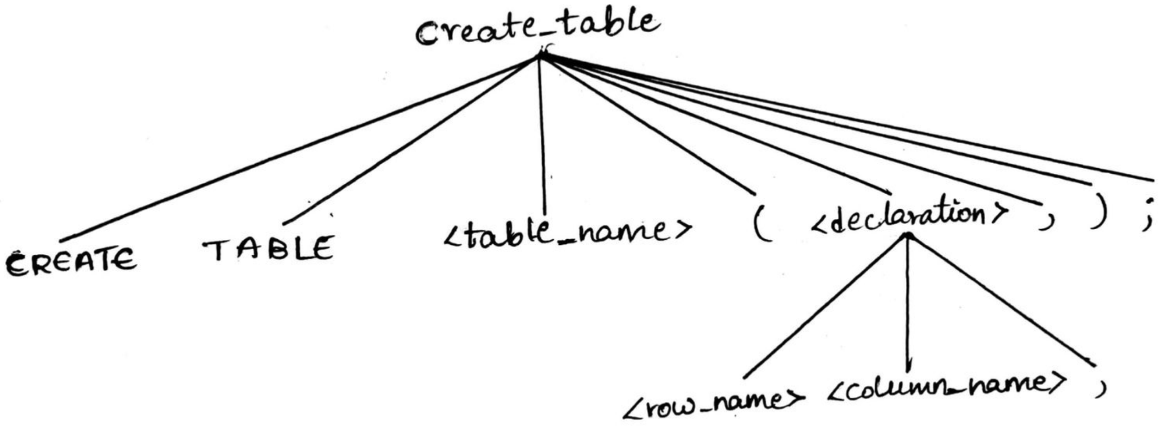
```
database_name → (table name) | (table name)*
database_statement → CREATE DATABASE <database_name>;
data_type → int | char | float | bit | float | double | date | timestamp | varchar(255)
```

table_name → (table name) | (table name)*
variable_name → (variable name) | (variable name)*
declaration → <row_name> <data_type> | (<row_name> <data_type>)*
create_table → <database_statement>
CREATE TABLE <table_name>(
<declaration>,
);

<u>Example:</u>

```
CREATE TABLE Persons (
   PersonID int,
   LastName varchar(255),
   FirstName varchar(255),
   Address varchar(255),
   City varchar(255)
);
```

Parse Tree:

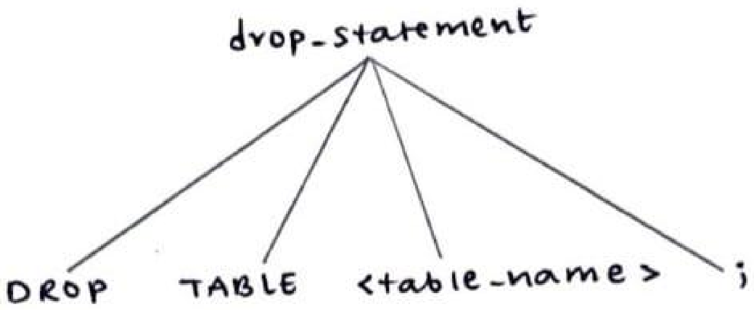


2) CFG for dropping or removing table in a database:

table_name → (table name) | (table name)*
drop_statement → DROP TABLE <table_name>;

<u>Example:</u>
DROP TABLE Persons;

Parse Tree:

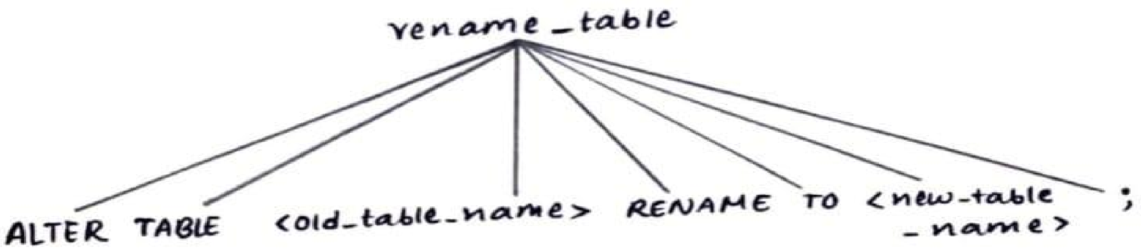


3) CFG for RENAME TABLE in a database:

old_table_name → (old table name) | (old table name)*
new_table_name → (new table name) | (new table name)*
rename_table → ALTER TABLE <old_tabli_name> RENAME TO <new_table_name>;

<u>Example:</u>
ALTER TABLE Persons RENAME TO Person;

Parse Tree:

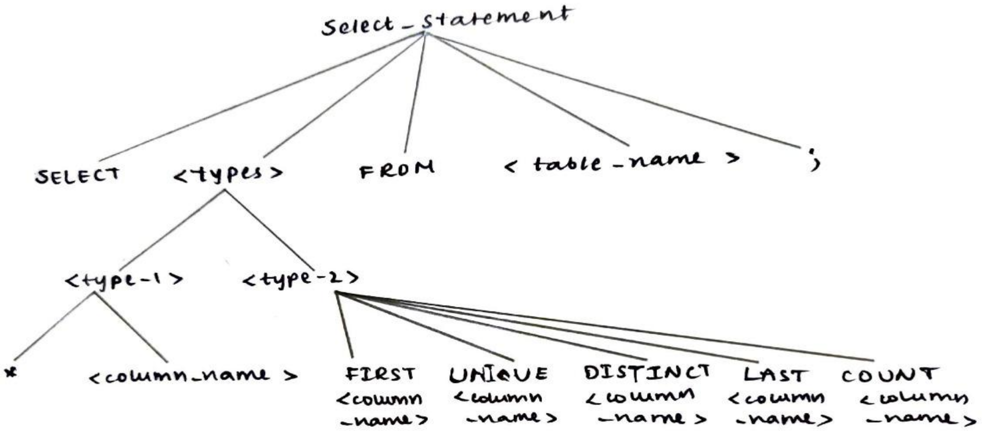


4) CFG for SELECT Table in a database:
table_name → (table name) | (table name)*
column_name → (column name) | (column name)*
type_1 → * | <column_name>
type_2 → UNIQUE <column_name>| DISTINCT <column_name>| FIRST <column_name> | LAST <column_name>| COUNT <column_name>
types → <type_1> | <type_2>
select_statement → SELECT <types> from <table_name>;

Example:

1) SELECT * FROM PERSONS;
2) SELECT Name FROM PERSONS;
3) SELECT FIRST Name FROM PERSONS;
4) SELECT UNIQUE City FROM PERSONS;
5) SELECT DISTINCT City FROM PERSONS;
6) SELECT LAST Name FROM PERSONS;
7) SELECT COUNT Name FROM PERSONS;

Parse Tree:

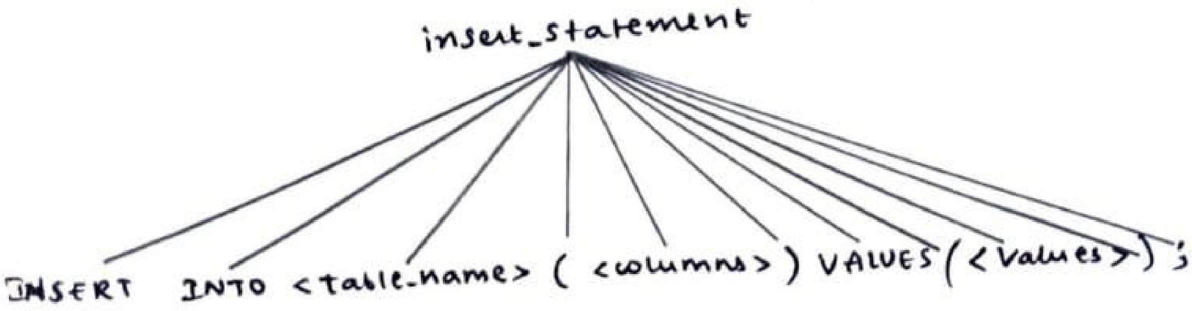


5) CFG for INSERT values in a column/columns of a table:

table_name → (table name) | (table name)*
column_name → (column name) | (column name)*
columns → <column_name> | (<column_name> , <column_name>,)*
value → (a-z)* | (A-Z)* | (A-Z)(a-z)*
values → <value> | (<value>,<value>,)*
Insert_statement → INSERT INTO <table_name> (<columns>)
VALUES(<values>);

<u>Example:</u>
INSERT INTO Customers (CustomerName, City, Country)
VALUES ('Cardinal', 'Stavanger', 'Norway');

Parse Tree:



SEMANTIC ACTIONS:

Semantic analysis is the task of ensuring that the declarations and statements of a program are semantically correct, i.e, that their meaning is clear and consistent with the way in which control structures and data types are supposed to be used.

While Statement:

When the control reaches a while loop, it evaluates the condition to determine whether the loop should start.

- Instructions inside the loop are executed if the condition is true.
- After executing the statement once, control returns to the while condition, which is tested to see if the condition is true before continuing.
- Instructions (loop body) are skipped if the condition is false and control is passed to the statement after the loop.

Let’s now see how the SQL While loop is used to insert dummy records in a database table.

For this we need a simple database “CarShop”:

CREATE DATABASE CarShop

We will create one table i.e. Cars within the CarShop database. The Cars table will have three columns Id, Name, and Price. Execute the following script:

```
CREATE TABLE Cars
(
Id INT PRIMARY KEY IDENTITY(1,1),
Name VARCHAR (50) NOT NULL,
Price INT
)
```

Let’s now use the While loop in SQL to insert 10 records in the Cars table. Execute the following script:

```
DECLARE @count INT;
SET @count = 1;

WHILE @count<= 10
BEGIN
  INSERT INTO Cars VALUES('Car-'+CAST(@count as varchar),
@count*100)
  SET @count = @count + 1;
END;
```

In the script above, we again declare a variable @count and initialize it with 1. Next, a while loop is executed until the value of the @count variable becomes greater than 10, which means that the while loop executes 10 times.

In the body of the while loop, the INSERT query is being used to insert one record into the Cars table. For the Name column, the value of the @count variable is appended with the string Car-. For the Price column of the Cars table, the value of the @count variable is multiplied by 100.

Case Statement (WHEN THEN) :

In this format of a CASE statement in SQL, we can evaluate a condition using comparison operators. Once this condition is satisfied, we get an expression from corresponding THEN in the output.

We can see the following syntax for the Case statement with a comparison operator.

```
CASE
    WHEN Comparison Condition THEN result
    WHEN Comparison Condition THEN result
    ELSE other
 END
```

Suppose we have a salary band for each designation. If employee salary is in between a particular range, we want to get designation using a Case statement.

In the following query, we are using a comparison operator and evaluate an expression.

```
Select EmployeeName,
 CASE
WHEN Salary >=80000 AND Salary <=100000 THEN 'Director'
WHEN Salary >=50000 AND Salary <80000 THEN 'Senior Consultant'
Else 'Director'
END AS Designation
from Employee
```

WHERE:

The SQL WHERE clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables. If the given condition is satisfied, then only it returns a specific value from the table. You should use the WHERE clause to filter the records and fetch only the necessary records.

The WHERE clause is not only used in the SELECT statement but it is also used in the UPDATE, DELETE statement, etc. To get the rows from the table that satisfy one or more conditions, you use the WHERE clause as follows:

```
SELECT
   select_list
FROM
   table_name
WHERE
   search_condition;
```

Example:

```
SELECT
 *
FROM
CITY
WHERE
COUNTRYCODE  = 'USA';
```

SYNTAX OF TARGET LANGUAGE:

We have chosen C language as our target language.

C language:

Basic Syntax of C language:

Header Files:

In C language, header files contain the set of predefined standard library functions. The “#include” preprocessing directive is used to include the header files with “.h” extension in the program.

Main Function:

A main is a predefined keyword or function in C. It is the first function of every C program that is responsible for starting the execution and termination of the program. It is a special function that always starts executing code from the 'main' having 'int' or 'void' as return data type. In other words, a main() function is an entry point of the programming code to start its execution.

Data Type:

A data type defines a set of values and the operations that can be performed on them. Every data type item (constant, variable …etc) in a C – program has a data type associated with it. C supports several types of data, each of which may be represented differently within the computer's memory.

There are mainly 5 types of data types used in the Turbo – C Compiler. They are:

1. Primary (or) Scalar (or) Standard (or) Simple data types.
2. Secondary (or) Derived (or) Structured data type.
3. User defined (or) Enumerated (or) Typedef data type.
4. Empty (or) Void data type.
5. Pointer data type.

1. Scalar (or) Standard data type: A scalar data type is used for representing a single value only. It is also called as Simple or Fundamental data type. These are used at primary level so these are also called as Primary data type. These are sub divided into 4 categories:

⇒ Integer data type:
It must have at least one digit.
It should not contain decimal point.
-ve or +ve values are allowed.
Its range is – 32768 to 32767.
It ranges for unsigned int be 0 to 65535.
It occupies 2bytes of memory.
Default sign is positive.
The format string specifier is %d or %i.

⇒ Float (or) Single precision real (or) Real data type:

It is of very small real type.

Default sign is positive.

It should contain at least one digit with a decimal point.

-ve or +ve values are allowed.

Its range is 3.4E – 38 to 3.4E +38.

Before ‘E’ is called mantissa after ‘E’ part is called Exponential part.

It occupies 4bytes of memory.

The format string specifier is %f.

⇒ Double Precision real (or) Double (or) Long float data type:

It is a very large floating value.

Its range is 1.7E – 308 to 1.7E +308.

There are long double and long float are also available.

It occupies 8bytes of memory.

The format string specifier is %lf.

⇒ Character data type:

This is a single character or a group of characters (called String).

It may be signed or unsigned.

It ranges for signed int be – 128 to + 127.

It ranges for unsigned int be 0 to 255.

It occupies 1bytes of memory.

The format string specifier is %c.

Type	**Size (Bytes)**	**Range**
char (or) signed char	1	-128 to 127

unsigned char	1	0 to 255
int (or) signed int	2	-32768 to 32767
unsigned int	2	0 to 65535
short int (or) signed short int	1	-128 to 127
unsigned short int	1	0 to 255
long int (or) signed long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295
float	4	3.4E $^{-38}$ to 3.4E $^{+38}$
Double	8	1.7E $^{-308}$ to 1.7E $^{+308}$
long double	10	3.4E $^{-4932}$ to 3.4E $^{+4932}$

2. Structured (or) Derived data type: Derived data types are derived from the primary data types by adding additional relationship

with the elements of the primary data types. These are used to represent a single or multiple values. These are also called as structured or secondary data types. These are divided into 3 categories:
⇒ Arrays and Strings
⇒ Structures
⇒ Unions

3. Enumerated (or) User Defined data type: This is also used for type definition i.e. it allows the users to define a variable or an identifier, which is used to represent existing data types. In other way it allows us to define new data types. These are defined as below:
Syntax: enum identifier {v1, v2, v3,….., vn};
(or)
enum identifier {v1, v2, v3,…., vn} variable;
Example: enum month {Jan, Feb, Mar,…, Dec};
(or)
enum year {1999, 2000, 2001,…., 2005}y;
There is another user defined data type typedef. It is also used to represent the existing data type and it can be used in place of the old data type anywhere in a C program. It is defined as:
Syntax: typedef data type identifier;
Example: typedef int pay;
typedef float salary;
Here pay tells us the link with int and salary with the float and these can be used for the declaration of the variable as:
pay p1, p2;
salary s1, s2, s3;

4. Void (or) Empty Data type: It is used in the user defined function or sub programs. It is used when a function has not any argument and the function does not return a value.
Syntax: void function name (int, int);void function name (void);
Example: void add (10, 20); void mul (void);

Variable:

It is a data name which is used to store data and may change during program execution. It is opposite to constant. Variable name is a name given to memory cells location of a computer where data is stored.

The following is the syntax for the declaration of variable:

Syntax: data type variable name;

Example: int a;   (or)   float b;

Variables of different data type must be declared in different statements. But variables of same data type can be declared in a single statement, separated by comma.

Syntax: data type variable name1, variable name 2,………, variable name n;

Example: int a, b, c;

Assigning values to variables: values to variables can be assigned in two ways:

⇒ Assigning values at the time of declaration

⇒ Assigning values after declaration

Assigning values at the time of declaration:

We can assign value to the variable at the time of declaring variables. These values act like default values of the variables, when the variable is accessed.

Syntax: data type variable name = assigning value;

Example: int a = 5;

Assigning values after the declaration:

We can assign value to the variable after the declaration of the variable. Accepting values from the user using input statements or assigning a value to the variable after the declaration statement are known as assigning values after declaration.

Syntax: data type variable name;

Variable name = assigning value;

Example: int a;

int a = 5;

Rules for declaring variables:

- Each variable must be preceded by a data type.
- Variables of different data types can be declared in a single statement, but they must be separated by semicolon.
- Variables of the same data type can be declared in a single statement separated by comma.
- Variables must be declared before their first use.
- Re-declaring variables is not allowed.
- Variable names are case sensitive.
- First character should be a letter or alphabet.
- Keywords are not allowed to be used as a variable name.
- White space is not allowed.
- C is case sensitive i.e. UPPER and lower case are significant.
- Only underscore, special symbols are allowed between two characters.
- The length of identifier may be upto 31 characters but only the first 8 characters are significant by compiler.

Arithmetic Operators:

- (+) This operator is used to add two operands.
- (-) This operator is used to subtract the second operand from the first.
- (*) This operator is used to multiply both operands

- (/)   This operator is used to divide numerator by de-numerator.
- (%) This operator is a Modulus Operator and the remainder after an integer division.
- (++) The increment operator increases the integer value by one.
- (--)   The decrement operator decreases the integer value by one.

Relational Operators:

- (==) This operator checks if the values of the two operands are equal or not. If yes, then the condition becomes true
- (!=)   This operator checks if the values of the two operands are equal or not. If the values are not equal, then the condition becomes true.
- (>)   This operator checks if the value of the left operand is greater than the value of the right operand. If yes, then the condition becomes true.
- (<)   This operator checks if the value of the left operand is less than the value of the right operand. If yes, then the condition becomes true.
- (>=) This operator checks if the value of the left operand is greater than or equal to the value of the right operand. If yes, then the condition becomes true.
- (<=) This operator checks if the value of the left operand is less than or equal to the value of the right operand. If yes, then the condition becomes true.

Logical Operators:

- (&&) This operator is called a logical AND operator. If both the operands are non-zero, then the condition becomes true.
- (||)   This operator is called a logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.
- (!)   This operator is called a logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false.

Assignment Operators:

- (=) This is a simple assignment operator. It assigns values from right side operands to left side operands.
- (+=) This is a simple add AND assignment operator. It adds the right operand to the left operand and assigns the result to the left operand.
- (-=) This is a subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.
- (*=) This is a multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.
- (/=) This is a divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.
- (%=) This is a M\modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.
- (<<=) This is the left shift AND assignment operator.
- (&=) This is a bitwise AND assignment operator.
- (^=) This is a bitwise exclusive OR and assignment operator.
- (|=) This is a bitwise inclusive OR and assignment operator.

Miscellaneous Operators:

- sizeof() It is used to return the size of a variable.
- (*) This is a pointer to a variable
- (?:) This is a C\conditional Expression.

Comments:

In the C Programming Language, you can place comments in your source code that are not executed as part of the program.

Comments provide clarity to the C source code allowing others to better understand what the code was intended to accomplish and greatly helping in debugging the code. Comments are especially important in large projects containing hundreds or thousands of lines of source code or in projects in which many contributors are working on the source code.

A comment starts with a slash asterisk /* and ends with a asterisk slash */ and can be anywhere in your program. Comments can span several lines within your C program. Comments are typically added directly above the related C source code.

Adding source code comments to your C source code is a highly recommended practice. In general, it is always better to over comment C source code than to not add enough.

An example code of while loop in C language:

```
#include <stdio.h>
int main () {
  /* local variable definition */
  int a = 10;
  /* while loop execution */
  while( a < 20 ) {
    printf("value of a: %d\n", a);
    a++;
  }
  return 0;
}
```