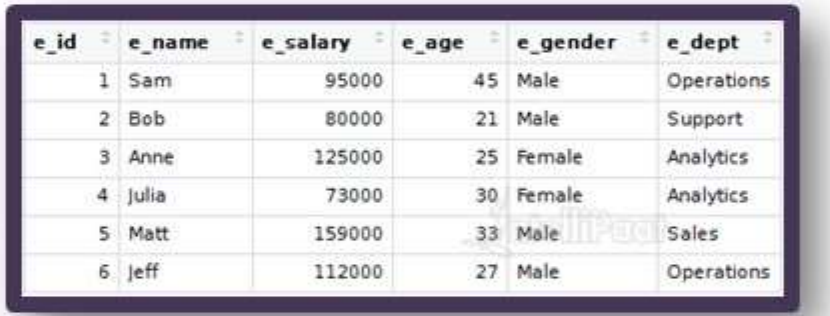
SQL Tutorial

SQL

Agenda

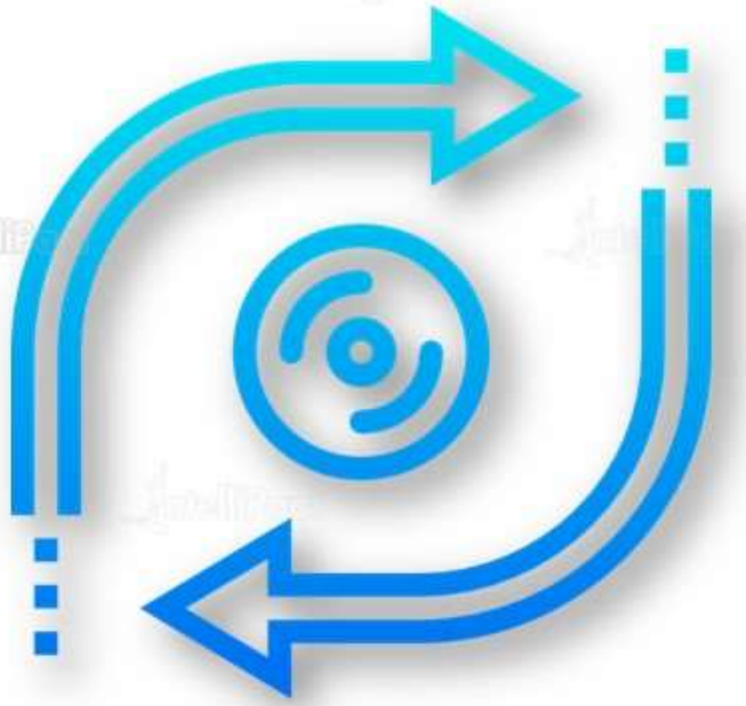
- Alter Table
- Temporary Table
- SQL Server Functions
- Case Statement
- IIF Function in SQL
- User-defined Functions
- Programming Constructs



e_id	e_name	e_salary	e_age	e_gender	e_dept
1	Sam	95000	45	Male	Operations
2	Bob	80000	21	Male	Support
3	Anne	125000	25	Female	Analytics
4	Julia	73000	30	Female	Analytics
5	Matt	159000	33	Male	Sales
6	Jeff	112000	27	Male	Operations

Alter Table

Alter Table statement is used to add, delete, or modify columns in a table.



Alter Table: Add Column

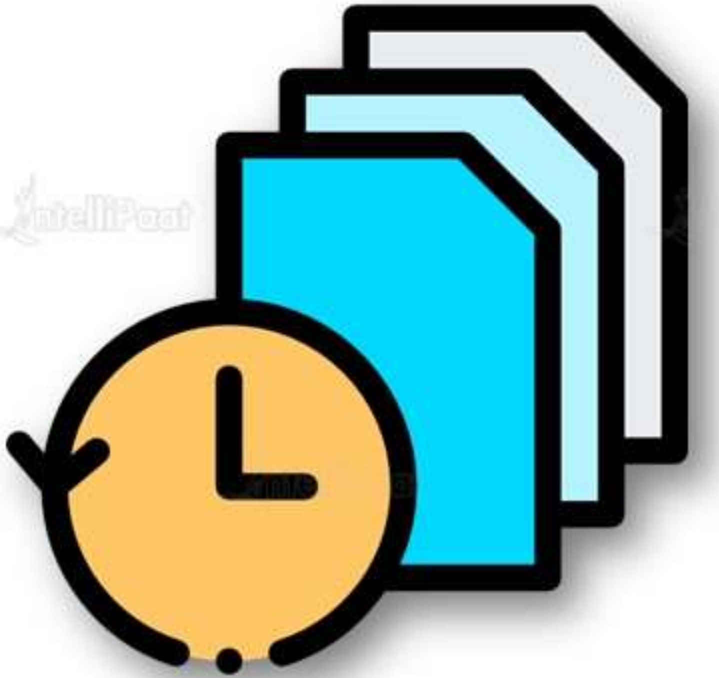
```
ALTER TABLE table_name
ADD column_name datatype;
```

Alter Table: Drop Column

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Temporary Table

Temporary tables are created in tempDB and deleted as soon as the session is terminated.



Syntax to Create Temporary Table

```
CREATE TABLE #table_name(
);
```

SQL Server Functions

SQL Server Functions

- **Aggregate Functions**
- Date Functions
- String Functions
- System Functions
- Window Functions

Σ

Aggregate functions are built in SQL Server functions and applied to sets of records rather than to a single record

<null>

Except for COUNT(*), aggregate functions ignore null values.

Aggregate functions are often used with the GROUP BY clause of the SELECT statement.

SQL Server Functions

Aggregate Functions

Date Functions

String Functions

System Functions

Window Functions

Aggregate function	Description
AVG	Calculates the average of non-NULL values in a set.
COUNT	Returns the number of rows in a group, including rows with NULL values.
MAX	Returns the highest value (maximum) in a set of non-NULL values
MIN	Returns the lowest value (minimum) in a set of non-NULL values.
SUM	Returns the summation of all non-NULL values a set.

SQL Server Functions

- Aggregate Functions
- Date Functions
- String Functions
- System Functions
- Window Functions

Aggregate function	Description
CHECKSUM_AGG	Calculates a checksum value based on a group of rows.
COUNT_BIG	Returns the number of rows (with BIGINT data type) in a group, including rows with NULL values
STDEV	Returns the statistical standard deviation of all values provided in the expression based on a sample of the data population.
STDEVP	Returns the standard deviation for all values in the provided expression, but does so based on the entire data population.
VAR	Returns the summation of all non-NULL values a set.
VARP	Returns the statistical variance of values in an expression but does so based on the entire data population.

SQL Server Functions

Aggregate Functions

Date Functions

String Functions

System Functions

Window Functions

AVG()

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

SQL Server Functions

Aggregate Functions

Date Functions

String Functions

System Functions

Window Functions

COUNT()

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

SQL Server Functions

Aggregate Functions

Date Functions

String Functions

System Functions

Window Functions

MAX()

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

SQL Server Functions

- **Aggregate Functions**
- Date Functions
- String Functions
- System Functions
- Window Functions

MIN()

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

SQL Server Functions

- **Aggregate Functions**
- Date Functions
- String Functions
- System Functions
- Window Functions

SUM()

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

SQL Server Functions

Aggregate Functions

Date Functions

String Functions

System Functions

Window Functions

CHECKSUM_AGG()

```
SELECT
CHECKSUM_AGG(column_name)
FROM table_name
```

SQL Server Functions

Aggregate Functions

Date Functions

String Functions

System Functions

Window Functions

COUNT_BIG()

```
SELECT COUNT_BIG(column_name)
FROM table_name;
```

SQL Server Functions

Aggregate Functions

Date Functions

String Functions

System Functions

Window Functions

STDEV()

```
SELECT STDEV(column_name)
FROM table_name;
```

SQL Server Functions

Aggregate Functions

Date Functions

String Functions

System Functions

Window Functions

STDEVP()

```
SELECT STDEVP(column_name)
FROM table_name;
```

SQL Server Functions

- **Aggregate Functions**
- Date Functions
- String Functions
- System Functions
- Window Functions

VAR()

```
SELECT VAR(column_name)
FROM table_name;
```

SQL Server Functions

Aggregate Functions

Date Functions

String Functions

System Functions

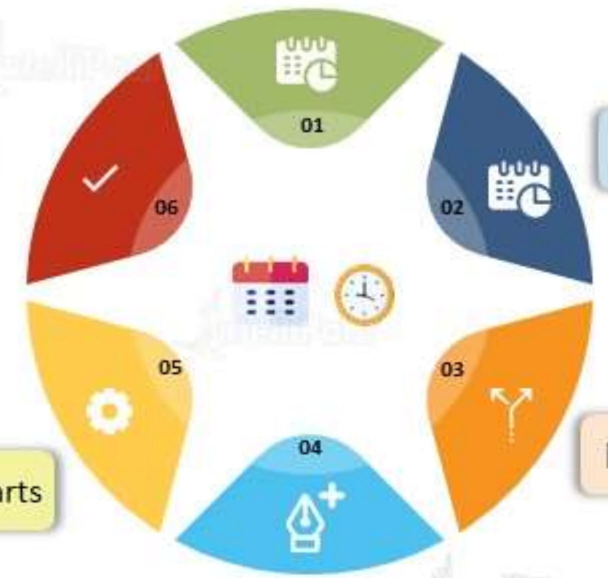
Window Functions

VARP()

```
SELECT VARP(column_name)
FROM table_name;
```

SQL Server Functions

- Aggregate Functions
- **Date Functions**
- String Functions
- System Functions
- Window Functions


Returning the current date and time
01
Returning the date and time Parts
02
Returning a difference between two dates
03
Modifying dates
04
Constructing date and time from their parts
05
Validating date and time values
06

SQL Server Functions

- Aggregate Functions
- **Date Functions**
- String Functions
- System Functions
- Window Functions

Returning the current date and time

Function	Description
CURRENT_TIMESTAMP	Returns the current system date and time without the time zone part
GETUTCDATE	Returns a date part of a date as an integer number
GETDATE	Returns the current system date and time of the operating system on which the SQL Server is running
SYSDATETIME	Returns the current system date and time with more fractional seconds precision than the GETDATE() function
SYSUTCDATETIME	Returns the current system date and time in UTC time
SYSDATETIMEOFFSET	Returns the current system date and time with the time zone

SQL Server Functions

Aggregate Functions

Date Functions

String Functions

System Functions

Window Functions

Returning the date and time Parts

Function	Description
DATENAME	Returns a date part of a date as a character string
DATEPART	Returns a date part of a date as an integer number
DAY	Returns the day of a specified date as an integer number
MONTH	Returns the month of a specified date as an integer
YEAR	Returns the year of the date as an integer

SQL Server Functions

- Aggregate Functions
- **Date Functions**
- String Functions
- System Functions
- Window Functions

Returning a difference between two dates

Function	Description
DATEDIFF	Returns a difference in date part between two date

SQL Server Functions

- Aggregate Functions
- **Date Functions**
- String Functions
- System Functions
- Window Functions

Modifying dates

Function	Description
DATEADD	Adds a value to a date part of a date and return the new date value
EOMONTH	Returns the last day of the month containing the specified date, with an optional offset
SWITCHOFFSET	Changes the time zone offset of a DATETIMEOFFSET value and preserves the UTC value
TODATETIMEOFFSET	Transforms a DATETIME2 value into a DATETIMEOFFSET value

SQL Server Functions

- Aggregate Functions
- **Date Functions**
- String Functions
- System Functions
- Window Functions

Constructing date and time from their parts

Function	Description
DATEFROMPARTS	Return a DATE value from the year, month, and day
DATETIME2FROMPARTS	Returns a DATETIME2 value from the date and time arguments
DATETIMEOFFSETFROMPARTS	Returns a DATETIMEOFFSET value from the date and time arguments
TIMEFROMPARTS	Returns a TIME value from the time parts with the precisions

SQL Server Functions

Aggregate Functions

Date Functions

String Functions

System Functions

Window Functions

Validating date and time values

Function	Description
ISDATE	Check if a value is a valid date, time, or datetime value

SQL Server Functions

- Aggregate Functions
- Date Functions
- **String Functions**
- System Functions
- Window Functions

LTRIM()	Removes blanks on the left side of the character expression
LOWER()	Converts all characters to lower case letters
UPPER()	Converts all characters to upper case letters
REVERSE()	Reverses all the characters in the string
SUBSTRING()	Gives a substring from the original string

SQL Server Functions

- Aggregate Functions
- Date Functions
- String Functions
- **System Functions**
- Window Functions

Function	Description
CAST	Cast a value of one type to another
CONVERT	Convert a value of one type to another
CHOOSE	Return one of the two values based on the result of the first argument
ISNULL	Replace NULL with a specified value
ISNUMERIC	Check if an expression is a valid numeric type
IIF	Add if-else logic to a query

SQL Server Functions

- Aggregate Functions
- Date Functions
- String Functions
- **System Functions**
- Window Functions

Function	Description
TRY_CAST	Cast a value of one type to another and return NULL if the cast fails
TRY_CONVERT	Convert a value of one type to another and return the value to be translated into the specified type. It returns NULL if the cast fails
TRY_PARSE	Convert a string to a date/time or a number and return NULL if the conversion fails
Convert datetime to string	Show you how to convert a datetime value to a string in a specified format
Convert string to datetime	Describe how to convert a string to a datetime value
Convert datetime to date	Convert a datetime to a date

SQL Server Functions

Aggregate Functions

Date Functions

String Functions

System Functions

Window Functions

Function	Description
CUME_DIST	Calculate the cumulative distribution of a value in a set of values
DENSE_RANK	Assign a rank value to each row within a partition of a result, with no gaps in rank values
FIRST_VALUE	Get the value of the first row in an ordered partition of a result set
LAG	Provide access to a row at a given physical offset that comes before the current row
LAST_VALUE	Get the value of the last row in an ordered partition of a result set

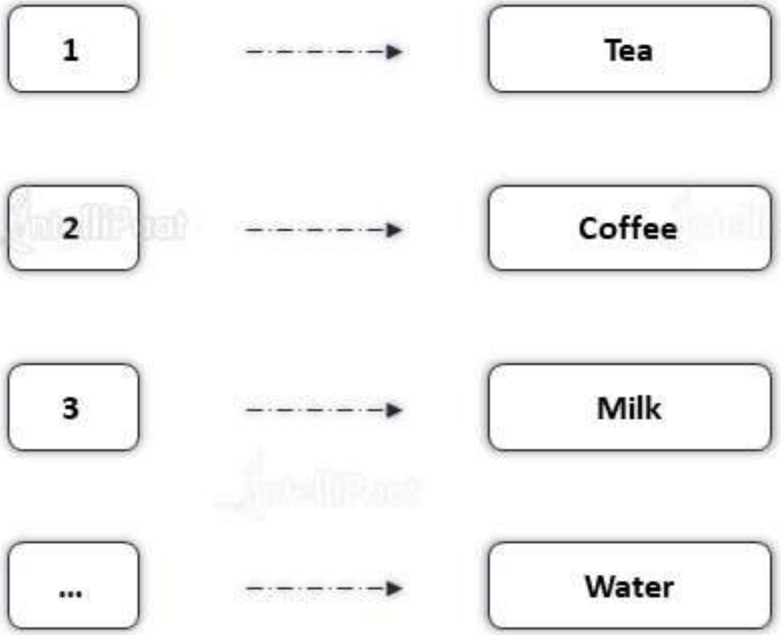
SQL Server Functions

- Aggregate Functions
- Date Functions
- String Functions
- System Functions
- Window Functions

Function	Description
LEAD	Provide access to a row at a given physical offset that follows the current row
NTILE	Distribute rows of an ordered partition into a number of groups or buckets
PERCENT_RANK	Calculate the percent rank of a value in a set of values
RANK	Assign a rank value to each row within a partition of a result set
ROW_NUMBER	Assign a unique sequential integer to rows within a partition of a result set, the first row starts from 1t

Case Statement

Case Statement helps in multi-way decision-making



Case Statement: Syntax

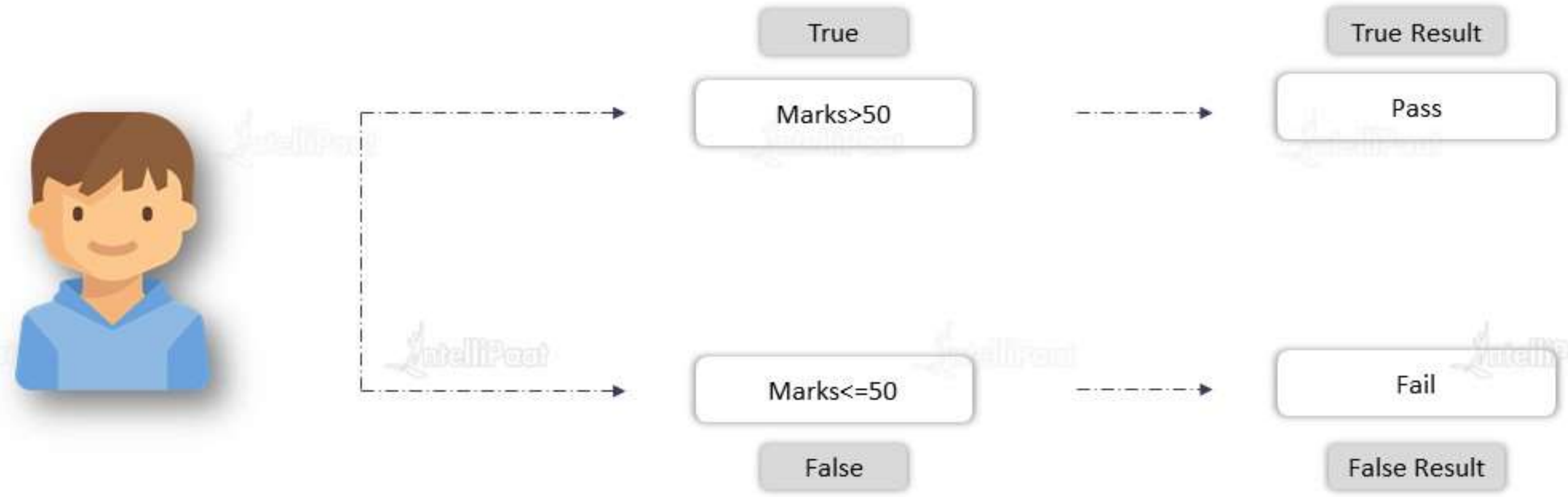
```
CASE
   WHEN condition1 THEN result1
   WHEN condition2 THEN result2
   WHEN conditionN THEN resultN
   ELSE result
END;
```

IIF() Function

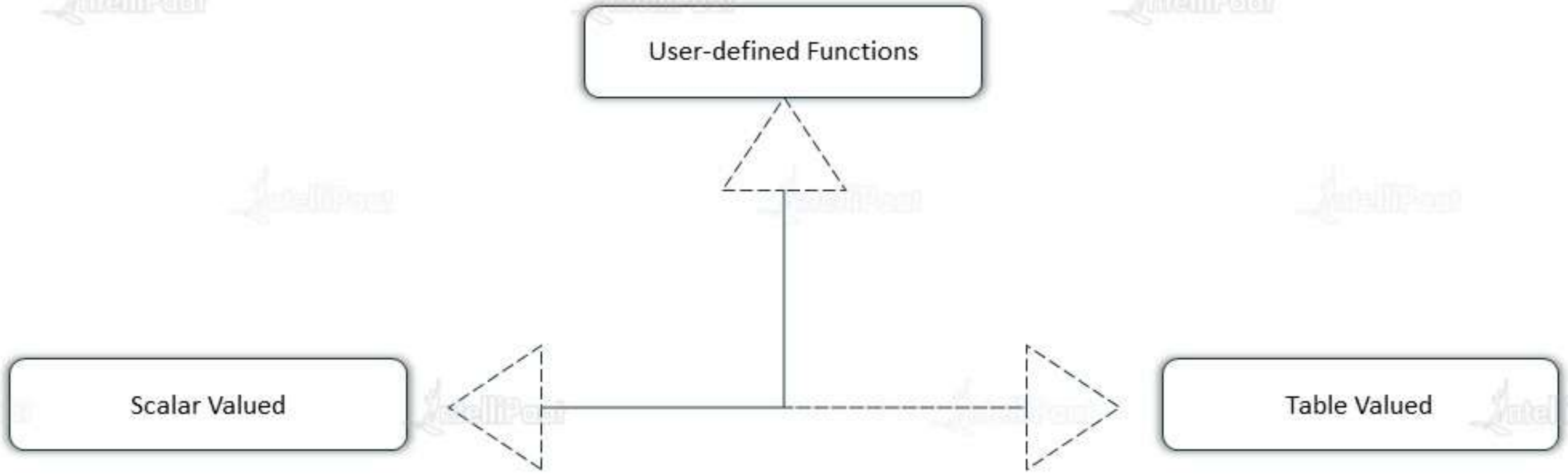
IIF() function is an alternative for the case statement.

```
IIF (boolean_expression, true_value,
false_value )
```

IIF() Function



Types of User-defined Functions



Scalar Valued Function

Scalar valued function always returns a scalar value.

```
CREATE FUNCTION function_name(@param1 data_type, @param2
data_type…)
RETURNS return_datatype
AS
BEGIN
    -----Function body
RETURN value
 END
```

Table Valued Function

Table valued function returns a table.

```
CREATE FUNCTION function_name(@param1 data_type, @param2
data_type…)
RETURNS table
AS
RETURN (SELECT column_list FROM table_name WHERE [condition] )
```

Programming Constructs

Using Variables

The DECLARE keyword enables you to declare several variables with a single statement (although this device can sometimes look confusing when you look at your code later). An example of this type of statement appears here:

```
1> declare @emp_name char(30), @id int
2> go
```

Using Variables to Store Data

Variables are available only within the current statement block. To execute a block of statements using the Transact-SQL language, the go statement is executed. (Oracle uses the semicolon for the same purpose.) The scope of a variable refers to the usage of the variable within the current Transact-SQL statement.

You cannot initialize variables simply by using the = sign. Try the following statement and note that an error will be returned.

```
1> declare @name char(30)
2> @name = "Billy Brewster"
3> go
```

Using Variables to Store Data

Variables are available only within the current statement block. To execute a block of statements using the Transact-SQL language, the go statement is executed. (Oracle uses the semicolon for the same purpose.) The scope of a variable refers to the usage of the variable within the current Transact-SQL statement.

You should have received an error informing you of the improper syntax used in line 2. The proper way to initialize a variable is to use the SELECT command. (Yes, the same command you have already mastered.) Repeat the preceding example using the correct syntax:

```
1> declare @name char(30)
2> select @name = "Billy Brewster"
3> go
```

Retrieving Data into Local Variables

Variables often store data that has been retrieved from the database. They can be used with common SQL commands, such as SELECT, INSERT, UPDATE, and DELETE. Example 19.1 illustrates the use of variables in this manner.

This example retrieves the name of the player in the BASEBALL database who has the highest batting average and plays for the Portland Beavers

```
1> declare @team_id int, @player_name char(30), @max_avg float
2> select @team_id = TEAM_ID from TEAMS where CITY = "Portland"
3> select @max_avg = max(AVERAGE) from BATTERS where TEAM =
@team_id
4> select @player_name = NAME from BATTERS where AVERAGE =
@max_avg
5> go
```

BEGIN and END Statements

Transact-SQL uses the BEGIN and END statements to signify the beginning and ending points of blocks of code. Other languages use brackets ({}) or some other operator to signify the beginning and ending points of functional groups of code. These statements are often combined with IF...ELSE statements and WHILE loops. Here is a sample block using BEGIN and END:

```
BEGIN
 statement1
 statement2
 statement3...
END
```

IF...ELSE Statements

One of the most basic programming constructs is the IF...ELSE statement. Nearly every programming language supports this construct, and it is extremely useful for checking the value of data retrieved from the database. The Transact-SQL syntax for the IF...ELSE statement looks like this:

```
if (condition)
begin
    (statement block)
end
else if (condition)
begin
    statement block)
end
.
.
else
begin
    (statement block)
end
```

IF...ELSE Statements

One of the most basic programming constructs is the IF...ELSE statement. Nearly every programming language supports this construct, and it is extremely useful for checking the value of data retrieved from the database. The Transact-SQL syntax for the IF...ELSE statement looks like this:

Note that for each condition that might be true, a new BEGIN/END block of statements was entered. Also, it is considered good programming practice to indent statement blocks a set amount of spaces and to keep this number of spaces the same throughout your application. This visual convention greatly improves the readability of the program and cuts down on silly errors that are often caused by simply misreading the code.

EXISTS Condition

The EXISTS keyword ensures that a value is returned from a SELECT statement. If a value is returned, the IF statement is executed. Example 19.5 illustrates this logic.

In this example the EXISTS keyword evaluates a condition in the IF. The condition is specified by using a SELECT statement.

```
1> if exists (select * from TEAMS where TEAM_ID > 5)
2> begin
3>     print "IT EXISTS!!"
4> end
5> else
6> begin
7>     print “NOT EXISTS!"
8> end
```

WHILE Loop

Another popular programming construct that Transact-SQL supports is the WHILE loop. This command has the following syntax:

```
WHILE logical_expression
   statement(s)
```

WHILE Loop

The WHILE loop continues to loop through its statements until the logical expression it is checking returns a FALSE. This example uses a simple WHILE loop to increment a local variable (named COUNT).

```
1> declare @COUNT int
2> select @COUNT = 1
3> while (@COUNT < 10)
4> begin
5>     select @COUNT = @COUNT + 1
6>     print "LOOP AGAIN!"
7> end
8> print "LOOP FINISHED!"
```

BREAK Command

You can issue the BREAK command within a WHILE loop to force an immediate exit from the loop. The BREAK command is often used along with an IF test to check some condition.

Notice the placement of the BREAK statement after the evaluation of the first condition in the IF

```
declare @COUNT int
select @COUNT = 1
while (@COUNT < 10)
begin
    select @COUNT = @COUNT + 1
    if (@COUNT = 8)
    begin
        break
    end
    else
    begin
        print "LOOP AGAIN!"
    end
end
print "LOOP FINISHED!"
```

CONTINUE Command

CONTINUE command is also a special command that can be executed from within a WHILE loop. The CONTINUE command forces the loop to immediately jump back to the beginning, rather than executing the remainder of the loop and then jumping back to the beginning

Notice the placement of the CONTINUE statement after the evaluation of the first condition in the IF

```
1> declare @COUNT int
2> select @COUNT = 1
3> while (@COUNT < 10)
4> begin
5>     select @COUNT = @COUNT + 1
6>     if (@COUNT = 8)
7>     begin
8>         continue
9>     end
10>    else
11>    begin
12>        print "LOOP AGAIN!"
13>    end
14> end
15> print "LOOP FINISHED!"
```

Quiz

Quiz

In which database are the temporary tables created and stored?

A	master
B	tempDB
C	model
D	msdb

Solution

In which database are the temporary tables created and stored?

- A master
- [✓ B tempDB]
- C model
- D msdb

Quiz

Which of the following statement will add a column ‘F_name’ to the STUDENT table?

A `ALTER TABLE Student add column ( F_name varchar(20));`

B `ALTER TABLE Student add (F_name varchar(20));`

C `ALTER TABLE Student add column (F_name);`

D `ALTER TABLE Student add F_name varchar(20);`

Solution

Which of the following statement will add a column ‘F_name’ to the STUDENT table?

- A ALTER TABLE Student add column (F_name varchar(20));
- B ALTER TABLE Student add (F_name varchar(20));
- C ALTER TABLE Student add column (F_name);
- [✓ D ALTER TABLE Student add F_name varchar(20);]

Quiz

User defined function in SQL Server can return:

A	Scalar value
B	Set of values
C	Result set
D	All of the mentioned

Solution

User defined function in SQL Server can return:

A	Scalar value
B	Set of values
C	Result set
[✓ D]	[✓ All of the mentioned]

Quiz

Which of the following functions will give the average value of a numeric column:

A	mean()
B	avg()
C	average()
D	trim()

Solution

Which of the following functions will give the average value of a numeric column:

- A mean()
- [✓ B avg()]
- C average()
- D trim()

India: +91-7847955955

US: 1-800-216-8930 (TOLL FREE)

sales@intellipaat.com

24/7 Chat with Our Course Advisor