1. Introduction:

1.1 Data

Related information is called as DATA.

1.2 Database

Data stored as a collection in an organized manner can be called as database.

1.3 RDBMS

A Relational Database Management System is a program that lets you create, update and administrator a relational database. The primary rule for RDBMS is that the Data should be stored in the form of tables.

Most of the RDBMS's use the Structures Query Language to access the database.

When a database undergoes NORMALISATION it is called as a RDBMS.

Different products of RDBMS are -

ORACLE	Oracle Corporation.
SQL Server2000	Microsoft Corporation.
DB2 UDB	IBM
MySQL	MySQL
Sybase	Sybase
Teradata	NCR

2. SQL (Structured Query Language)

This is a common language through which we can interact with the database.

SQL is classified mainly into three categories.

1. **DDL** (Data Definition Language).
   1. Create **2.** Alter **3.** Truncate **4.** Drop
2. **DML** (Data Manipulation Language).
   1. Select **2.** Insert **3.** Update **4.** Delete
3. **TCL** (Transaction Control Language).
   1.Commit **2.** Rollback **3.** Save point
4. **DCL** (Data Control Language)

2.1 DATA DEFINITION LANGUAGE

These commands are used to create, modify and delete database objects.

DDL Commands are **Auto-commit.**

Create: This command is used to create database objects such as - Tables, Views, Indexes, Synonyms, Sequences, Stored Procedures, Triggers, Functions, Packages and user-defined data-types.

Alter: This command is used to modify the structure of the database objects.

Drop: This command is used to remove the objects from the database.

Truncate: Using this command we can permanently remove the rows from a table. Here we cannot append any clauses to the Truncate statement.

2.1.1 DATA TYPES

ORACLE	SQL Server	DB2
int	int	Int
char	char	Char
varchar2	-	-
varchar	varchar	Varchar
number	numeric	Numeric
float	float	Float
decimal	decimal	Decimal
Date	smalldatetime	Date
timestamp	datetime	Timestamp

2.1.2 Constraints

Constraint restricts the values that the table can store.

We can declare integrity constraints at the table level or column level.

In **column-level constraints** the constraint type is specified after specifying the column datatype i.e., before the delimiting comma.

Where as in the **table-level constraints** the constraint type is going to be specified as separate comma-delimited clauses after defining the columns.

There are five constraints

1. Not Null
2. Unique Key
3. Check
4. Primary Key
5. Foreign Key

2.1.2.1 Not Null

If a column in a table is specified as Not Null, then it's not possible to insert a null in such a column. It can be implemented with create and alter commands. When we implement the Not Null constraint with alter command there should not be any null values in the existing table.

2.1.2.2 Unique Key

The unique constraint doesn't allow duplicate values in a column. If the unique constraint encompasses two or more columns, no two equal combinations are allowed.

Note: There is a different behaviour in the following Relational Databases.

DB2: To enforce unique constraint, the column must be a Not Null column. That means, we can not insert a single row with a Null value against the Unique Key constraint column.

MS SQL Server: In this, we can insert one Row with a Null value against the Unique Key constraint column.

Oracle: In this, we can insert any number of Rows with a Null value against the Unique Key constraint column. Please keep it in mind that two Null values are not equal.

2.1.2.3 Check

Check constraint is used to restrict the values before inserting into a table.

2.1.2.4 Primary Key

The key column with which we can identify the entire Table is called as a primary key column. A primary key is a combination of a Unique and a Not Null constraint, it will not allow null and duplicate values. A table can have only one primary key.

A primary key can be declared on two or more columns as a Composite Primary Key.

2.1.2.5 Foreign Key

Columns defined as foreign keys refer the Primary Key of other tables. The Foreign Key "points" to a primary key of another table, guaranteeing that you can't enter data into a table unless the referenced table has the data already which enforces the REFERENTIAL INTEGRITY. This column will take Null values.

Except Not Null constraint all other constraints can be given at both the column level and table level.

TABLE CREATION

Syntax

```
CREATE TABLE <table name> (
<column_name1> <data type>[(<width>)]
                    [constraint <constraint name> <constraint type>],
<column_name2> <data type>[(<width>)],
<column_name3> <data type>[(<width>)],
<column_name4> <data type>[(<width>)],
.
<column_nameN> <data type>[(<width>)]
);
```

Example

```
CREATE TABLE Employee(
empno number(4) constraint pk_empno primary key,
ename varchar2(50) constraint nn_ename not null,
salary number(10,2),
hire_date date,
gender char(1) constraint chk_gen check(gender in ('M', 'F', 'm', 'f')),
email varchar2(50) unique
);
```

ALTER TABLE

Syntax

ALTER TABLE <table name> ADD CONSTRAINT <constraint name> <constraint type> <column name>;

Syntax to add referential integrity

ALTER TABLE <table name> ADD CONSTRAINT <constraint name>
FOREIGN KEY(<foreign key column>)
REFERNCES <parent table name>(<primary key column>);

Example

ALTER TABLE Employee ADD CONSTRAINT nn_hdate NOT NULL hire_date;
ALTER TABLE Employee ADD CONSTRAINT Ref_dept
FOREIGN KEY (deptno)
REFERENCES Department(deptno);

DROP

Syntax

DROP <OBJECT> <object name>;

Example

DROP PROCEDURE ins_emp;
DROP TABLE Employee;

TRUNCATE

Syntax

TRUNCATE TABLE <table-name>;

Example

TRUNCATE TABLE Employee;

2.2 DATA MANIPULATION LANGUAGE

These commands are used to append, change or remove the data in a Table.
COMMIT/ROLLBACK statement should be given to make the changes permanent or to revert back.

2.2.1 INSERT

Using this command we can append data into tables.

Syntax

INSERT INTO <table name>(<column_name1>, <column_name2>, ...)
VALUES (column1_value, column2_value, ...);
INSERT INTO <table-name>(<column_name2>, <column_name1>, ...)
VALUES (column2-value, column1-value, ...);
INSERT INTO <table-name> VALUES (value1, value2, ...);

Example

INSERT INTO Employee(empno, ename, salary, hire_date, gender, email)
VALUES(1234, 'JOHN', 8000, '18-AUG-80, 'M', 'john@miraclesoft.com');
INSERT INTO Employee(email , ename, empno, hire_date, gender, salary)
VALUES('rhonda@miraclesoft.com', 'RHONDA', 1235, '24-JUL-81', 'F', 7500);
INSERT INTO Employee
VALUES(1236, 'JACK', 15000, '23-SEP-79', 'm', 'jack@miraclesoft.com');

2.2.2 UPDATE

This command is used to modify the data existing in the tables.

Syntax

UPDATE <table-name> SET <column-name> = <value>;
UPDATE <table-name> SET <column-name> = <value> WHERE <condition>;

Example

UPDATE Employee SET salary = 15000;
UPDATE employee SET salary = 15000 WHERE empno = 1235;

2.2.3 DELETE

This command is used to remove the data from the tables.

Syntax

DELETE FROM <table-name>;
DELETE FROM <table-name>WHERE <condition>;

Example

DELETE FROM Employee;
DELETE FROM Employee WHERE empno = 1236;

2.3 TRANSACTION CONTROL LANGUAGE

These command are used to maintain the consistency of the database.

Commit

This command is used to make the transaction permanent.

Rollback

This command is used to undo the recent transaction.

Save Point

This is used to rollback a transaction to a specified point.

3.5 SET OPERATORS

OPERATOR	DESCRIPTION
UNION	Returns all rows retrieved by both the queries excluding common rows.
UNION ALL	Returns all rows retrieved by both the queries including duplicate rows.
INTERSECT	Returns all common rows in both the queries.
MINUS	Returns all rows from the 1st query which are not existing in 2nd query.

4 BUILT-IN FUNCTIONS

To test functions databases following different mechanisms:

DUAL in Oracle
SYSIBM.SYSDUMMY1 in DB2
SELECT statement in SQL Server

DUAL

A table in Oracle having only ONE COLUMN with varchar2(1) data type and ONE ROW with a value of 'x'. This table is meant for getting only one row of data when we use a function, which returns a value.

Ex: Select 12345 * 33 from dual;

SYSIBM.SYSDUMMY1

A table in DB2 of SYSIBM's schema against which we can test any functions.

Ex: Select 12345 * 33 from SYSIBM.SYSDUMMY1;

In **SQL Server** we use a SELECT statement directly to test a function

Ex: Select 12345 * 33

4.1 CHARACTER FUNCTIONS

ORACLE	SQL SERVER	DB2	
ASCII	ASCII		ASCII returns the decimal representation in the database character set of the first character of char.
CHR	CHAR	CHAR	CHR returns the character having the binary equivalent to n in either the database character set or the national character set.
CONCAT	+(String concatenation)	-	CONCAT returns char1 concatenated with char2. Both char1 and char2 can be any of the datatypes CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB. The string returned is of VARCHAR2 datatype and is in the same character set as char1. This function is equivalent to the concatenation operator (||).
INITCAP	-		This function is used to covert the beginning letter of every word in a phrase to uppercase.
INSTR	CHARINDEX	LOCATE	This function search string for substring and returns an integer indicating the position of the character in-string that is the first character of this occurrence.
LENGTH	LEN	LENGTH	This function returns the length of the given string.
LOWER	LOWER	LOWER/LCASE	Convert the whole string into lower case.
LPAD	-		It adds the specified character to the given string on LEFT up to the specified length.
LTRIM	LTRIM	LTRIM	It trims the given string on LEFT side by removing the specified characters.
RPAD	-		It adds the specified character to the given string on RIGHT up to the specified length.
RTRIM	RTRIM	RTRIM	It trims the given string on RIGHT side by removing the specified characters.
REPLACE	REPLACE	REPLACE	It returns a string with every occurrence of search string replaced with the replacement string.
SUBSTR	SUBSTRING	SUBSTR	This function returns a portion of string beginning at the character portion and character length specified.
TRANSLATE	-	-	TRANSLATE returns char with all occurrences of each character in 'from string'replaced by its corresponding character in 'to string'.

TRIM	LTRIM/RTRIM	-	It trims both the leading and trailing characters from a character string.
UPPER	UPPER	UPPER/UCASE	Converts the given string into uppercase.

4.2 NUMERIC FUNCTIONS

ORACLE	SQL SERVER	DB2	
ABS	ABS	ABS	It returns the absolute value of the given integer.
CEIL	CEILING	CEIL	It returns the smallest integer greater than or equal to the given integer.
FLOOR	FLOOR	FLOOR	It returns the smallest integer greater than or equal to the given integer.
MOD	%	MOD	It returns the remainder of the m divided by m, returns 0 if n is 0.
POWER	POWER	POWER	Returns the first value raised the second value.
ROUND	ROUND	ROUND	
SQRT	-	SQRT	SQRT returns square root of n. The value cannot be negative. SQRT returns a "real"result.
TRUNC	CONVERT		

4.3 DATE FUNCTIONS

ORACLE	SQL SERVER	DB2
ADD_MONTHS	DATEADD	
CURRENT_TIMESTAMP	DATETIME	CURRENT_TIMESTAMP
EXTRACT	DATEPART	
LAST_DAY	-	
MONTHS_BETWEEN	DATEDIFF	
NEXT_DAY	-	
ROUND	-	
TRUNC	-	

ADD_MONTHS
Returns the date added with the respective number of months.

CURRENT_TIMESTAMP
Returns current time and date in the session time zone.

EXTRACT(datetime)
Returns a value of the specified datetime field from a datetime or interval value expression.

LAST_DAY
Returns the last day of the month in the specified date.

MONTHS_BETWEEN
Returns the number of months between the two specified dates.

NEXT_DAY
Returns the date of the first weekday of the given character(weekday name)

Here are a couple of examples

Description	Date Expression
Now	SYSDATE
Tomorrow/ next day	SYSDATE + 1
Seven days from now	SYSDATE + 7
One hour from now	SYSDATE + 1/24
Three hours from now	SYSDATE + 3/24
An half hour from now	SYSDATE + 1/48
10 minutes from now	SYSDATE + 10/1440
30 seconds from now	SYSDATE + 30/86400
Tomorrow at 12 midnight	TRUNC(SYSDATE + 1)
Tomorrow at 8 AM	TRUNC(SYSDATE + 1) + 8/24
Next Monday at 12:00 noon	NEXT_DAY(TRUNC(SYSDATE), 'MONDAY') + 12/24
First day of the month at 12 midnight	TRUNC(LAST_DAY(SYSDATE) + 1)
The next Monday, Wednesday or Friday at 9 a.m	TRUNC(LEAST(NEXT_DAY(sysdate,''MONDAY''),NEXT_DAY(sysdate,''WEDNESDAY''), NEXT_DAY(sysdate,''FRIDAY''))) + (9/24)

4.4 CONVERSION FUNCTIONS

ORACLE	SQL SERVER	DB2
CAST	CAST	CAST
TO_CHAR	CAST/CONVERT	TO_CHAR
TO_DATE	CAST	DATE
TO_NUMBER		INT
GREATEST		
LEAST		

CAST
To convert one built- in datatype into another built-in datatype.

TO_CHAR
To convert a date or timestamp value to a varchar2 value.

TO_DATE
To convert a character date to a date value.

TO_NUMBER
To convert a character number into a number value.

GREATEST
Returns the grater values in the given values.

LEAST
Returns the lower value in the supplied values.

4.5 GROUP FUNCTIONS

Group function work on a group of data to obtain aggregated values.

ORACLE	SQL SERVER	DB2UDB
AVG	AVG	
COUNT	COUNT	COUNT
MIN	MIN	MIN
MAX	MAX	MAX
SUM	SUM	SUM

4.6 NULL FUNCTIONS
COALESCE

The COALESCE function returns the first non-null expr in the expression list. At least one expr must not be the literal NULL. If all occurrences of expr evaluate to null, the function returns null. This function is a generalization of the NVL function. You can also use COALESCE as a variety of the CASE expression.

COALESCE (expr1, expr2)
is equivalent to:
CASE WHEN expr1 IS NOT NULL THEN expr1 ELSE expr2 END

NVL

If expr1 is null, NVL returns expr2. If expr1 is not null, NVL returns expr1. The arguments expr1 and expr2 can have any datatype. If their datatypes are different, Oracle converts expr2 to the datatype of expr1 before comparing them. The datatype of the return value is always the same as the datatype of expr1, unless expr1 is character data, in which case the return value's datatype is VARCHAR2 and is in the character set of expr1.

The following example returns a list of employee names and commissions, substituting "Not Applicable" if the employee receives no commission:

```
SELECT last_name, NVL(TO_CHAR(commission_pct), 'Not Applicable')
"COMMISSION" FROM employees WHERE last_name LIKE 'B%';

LAST_NAME                 COMMISSION
------------------------- ----------------------------------------
Baer                      Not Applicable
Baida                     Not Applicable
Banda                     .11
Bates                     .16
```

4.7 CASE Expression

This expression lets us use IF…THEN…ELSE logic in SQL Statements.

```
SELECT ename,
CASE  WHEN sal< 3000 THEN 'Low'
WHEN sal>10000 THEN 'High'
ELSE 'Medium' END
FROM Emp
```

How does one escape special characters when building SQL queries?

The LIKE keyword allows for string searches. The '_' wild card character is used to match exactly one character, '%' is used to match zero or more occurrences of any characters. These characters can be escaped in SQL.

Example

```
SELECT  ename FROM emp WHERE ename LIKE '_LA%' ;
```

- **How does one generate primary key values for a table?**

SEQUENCE

It is a auto-generated number which will be unique.

```
CREATE SEQUENCE EMPNO_SEQ START WITH 1111 INCREMENT BY 1;
INSERT INTO EMPLOYEE VALUES(EMPNO_SEQ.NEXTVAL,'PRAMOD');
```

To get Nth row from a table?

```
SELECT * FROM EMP E1 WHERE  n = (SELECT COUNT(rowid) FROM EMP E2 WHERE E1.rowid >=
E2.rowid);
```

7 VIEWS

A view is a logical or virtual table, which does not, exists physically in the database. Views are also called as Dictionary Objects.

Advantages

Security – The confidential columns can be suppressed from viewing and manipulation.
Complexity can be avoided.
Logical columns can be created.
Application design becomes easier.

Views can be classified into two categories on the way they are created.

1. Simple View
2. Complex View

Simple View

If the view is created from a single table, it is called as a simple view. We can do DML Operation, if it is Simple View.

Complex View

If the view is created on multiple tables, it is called as a complex view.

Syntax

CREATE [OR REPLACE] VIEW <view name>
AS <query>

Example 1

```
CREATE OR REPLACE VIEW data_emp
AS
SELECT ename, salary FROM Employee;
```

Example 2

```
CREATE OR REPLACE VIEW dup_emp
AS
SELECT ename, salary, salary*0.2 comm FROM Employee;
```

DROPPING A VIEW

Only the owner of the view can drop it.

Syntax

DROP VIEW <view name>;

Example

```
DROP VIEW data_emp;
```

8 PL/SQL

In SQL there is no control on the flow of execution of the statements.
What to do is can determined in SQL but when to do is not possible.
There is interactive programming only in SQL.
To obtain the control ORACLE introduced PL/SQL programming.

PL/SQL stands for Procedural Language/Structured Query Language.
It is a III Generation Language.
There is a Batch programming in PL/SQL.
The basic unit of PL/SQL is a **BLOCK.**

Basic Structure of a PL/SQL Block

```
[DECLARE
        <declarations>;]            //declarations are done here.
BEGIN
        <action>;                   //main objective of the block.
[EXCEPTION
        <action>;]                  //exception handling.
END;
/

OR

BEGIN
        <action>;
END;
/
```

There are two types of blocks

ANONYMOUS Blocks
NAMED Blocks

8.1 Anonymous blocks

These are meant for batch processing but they don't have any names.
Simply these are called as no-named blocks.

Example

```
-- Program to calculate average of two numbers.

DECLARE
        a number;
        b number;
        c number;
BEGIN
        a:=&first_number;
        b:=&second_number;
        c:=(a*b)/2
        dbms_output.put_line('The average of the given two numbers is: '||c)
END;
/               -- PL/SQL block terminator.
```

8.2 Named Blocks

These are also meant for batch processing but posses a name and are stored as an object in the database.
Generally these are

- Stored Procedures.
- Stored Functions.

8.2.1 STORED PROCEDURE

Definition

It is sub-program, which is stored in the database and whenever it is called, it does something but doesn't return any value.
But they could return indirectly through OUT parameters.
Using the stored procedures you can reduce the number of calls to the database and decrease network traffic.

Syntax of a Stored Procedure (ORACLE)

```
CREATE [OR REPLACE] PROCEDURE <procedure name>[(
        <parameter1> <data type> [<width>] [<parameter mode>],
        <parameter2> <data type> [<width>] [<parameter mode>],
        <parameter3> <data type> [<width>] [<parameter mode>],
        <parameter4> <data type> [<width>] [<parameter mode>],
        .
        .
        .
        <parameterN> <data type> [<width>] [<parameter mode>]
        )]
AS
        <DECLERATIONS>;
BEGIN
        <action-code>;
[EXCEPTION
        <action-code>;]
END;
```

Example (ORACLE)

```
-- Procedure to insert a new row into the Employee table.

CREATE OR REPLACE PROCEDURE INS_EMP(
        p_empno      number,                    // by default all are IN mode parameters.
        p_ename      varchar2(50),
        p_salary     number(10,2),
        p_hire_date  date,
        p_gender     char(1),
        p_email      varchar2(50),
        p_out_param  number OUT          // OUT parameter declaration.
        )
AS

BEGIN
        INSERT INTO Employee(empno, ename, salary, hire_date, gender, email)
                VALUES(p_empno, p_ename, p_salary, p_hire_date, p_gender, p_email);

        p_out_param:=p_empno     //Assigning the employee number to out parameter.
        dbms_output.put_line('A row has been successfully inserted into Employee with
                                employee   number: '||p_out_param);
        // Output statement to get the feed back that insertion is done.
END;
```

Syntax of a Stored Procedure (SQL Server2000)

```
CREATE PROCE[DURE] <procedure name>[(
```

```
            <parameter1> <data type> [<width>] [<parameter mode>],
            <parameter2> <data type> [<width>] [<parameter mode>],
            <parameter3> <data type> [<width>] [<parameter mode>],
            <parameter4> <data type> [<width>] [<parameter mode>],
            .
            .
            .
            <parameterN> <data type> [<width>] [<parameter mode>]
            )]
      AS

            <sql statements>;
```

Example (SQL Server)

```
-- Procedure to insert a new row into the Employee table.

DROP PROCEDURE INS_EMP                  -- to drop the procedure if already exist.
go
CREATE PROCEDURE INS_EMP(
      @p_empno    numeric,                    -- by default all are IN mode parameters.
      @p_ename    varchar(50),
      @p_salary     numeric(10,2),
      @p_hire_date        datetime,
      @p_gender   char(1),
      @p_email    varchar(50),
      @p_out_param        numeric OUTPUT      -- OUT parameter declaration.
      )
AS

BEGIN
      INSERT INTO Employee(empno, ename, salary, hire_date, gender, email)
            VALUES(@p_empno, @p_ename, @p_salary, @p_hire_date, @p_gender,
@p_email);
      SET @p_out_param = empno;

      PRINT 'A row has been successfully inserted into Employee table with EmpNo: ' +
p_out_param;
                  -- Output statement to get the feed back that insertion is done.
END
go
```

Syntax of a Stored Procedure (DB2 UDB)

```
      CREATE PROCEDURE <procedure name>(
            [<parameter mode>]<parameter1> <data type> [<width>],
            [<parameter mode>]<parameter2> <data type> [<width>],
            [<parameter mode>]<parameter3> <data type> [<width>],
            [<parameter mode>]<parameter4> <data type> [<width>],
            .
            .
            .
            [<parameter mode>]<parameterN> <data type> [<width>]

      [DYNAMIC RESULT SETS
      LANGUAGE <language specifier>]
      BEGIN
```

```
        <sql statements>;
    END
    <special symbol other than ';'>      -- block terminator
```

Example (DB2)

```
-- Procedure to insert a new row into the Employee table.

DROP PROCEDURE INS_EMP                  -- to drop the procedure if already exist.
@
CREATE PROCEDURE INS_EMP(
        p_empno     numeric,                    -- by default all are IN mode parameters.
        p_ename     varchar(50),
        p_salary    numeric(10,2),
        p_hire_date datetime,
        p_gender    char(1),
        p_email     varchar(50),
OUT  p_out_param    numeric             -- OUT parameter declaration.
        )
AS
LANGUAGE SQL
BEGIN
        INSERT INTO Employee(empno, ename, salary, hire_date, gender, email)
                VALUES(p_empno, p_ename, p_salary, p_hire_date, p_gender, p_email);

        SET p_out_param=p_empno;        -- Assigning the employee number to out parameter.
END
@
```

8.2.2 STORED FUNCTION

Definition

It is sub-program, which is stored in the database and whenever it is called, it does something and return some value.

Syntax of a function (ORACLE).

```
    CREATE OR REPLACE FUNCTION <function name>[(
            <parameter1> <data type> [<width>],
            <parameter2> <data type> [<width>],
            <parameter3> <data type> [<width>],
            <parameter4> <data type> [<width>],
            .
            .
            .
            <parameterN> <data type> [<width>]
            )]

    )
    return <type>
    as
```

```
BEGIN
        <function body>;.
        return <value>;
END;
/
```

Example (ORACLE)

```
-- Function to multiply two numbers.


CREATE OR REPLACE Function mutiply (
                                a number,
                                b number
                                )
RETURN number
AS
                        c number(3);
BEGIN
       c:=a*b;
       RETURN c;
END;
/
```

Syntax of a function (SQL Server).

```
        CREATE FUNCTION <function name>[(
                <parameter1> <data type> [<width>],
                <parameter2> <data type> [<width>],
                <parameter3> <data type> [<width>],
                <parameter4> <data type> [<width>],
                .
                .
                .
                <parameterN> <data type> [<width>]
                )]

        )
        return <type>
        as
        BEGIN
                <function body>
                return <value>
        END
```

Example (SQL Server2000)

```
-- Function to multiply two numbers.

DROP FUNCTION multiply
go

CREATE FUNCTION mutiply(
                                @a numeric,
                                @b numeric
                                )
RETURNS numeric
AS
```

```
BEGIN
        DECLARE @c numeric(3)
        SET @c=@a*@b;
        RETURN @c;
END
go
```

Syntax of a function (DB2 UDB).

```
CREATE FUNCTION <function name>>[(
        <parameter1> <data type> [<width>],
        <parameter2> <data type> [<width>],
        <parameter3> <data type> [<width>],
        <parameter4> <data type> [<width>],
        .
        .
        .
        <parameterN> <data type> [<width>]
        )]
RETURNS INTEGER
LANGUAGE <language specifier>
BEGIN ATOMIC
[DECLARE
        <declarations>;]
        <function body>;
END
<special symbol other than ';'>      -- block terminator
```

Example (DB2)

```
-- Function to multiply two numbers.

create function product (a integer, b integer)
returns integer
language sql
no external action
not deterministic
begin atomic
declare  v_pro integer;
 set v_pro = a*b;
return v_pro ;
end
@
```

8.2.3 TRIGGER

A trigger is a block of statements, which are fired automatically when a DML operation takes place.
These are always associated with the database tables.

Triggers are classified as two types:

1. Row Level Triggers.
2. Statement Level Triggers.

1.Row Level Triggers

These triggers will be fired for each row with proportionate to the number of rows those are begin effected the DML statement.

2. Statement Level Triggers

These triggers are fired once for each statement regardless with the number of rows those are being affected by the DML statement.

In oracle triggers can be classified into two more categories such as

a) Before Triggers. b) After Triggers.

a) Before Triggers

These triggers are fired **before** the execution of the DML statements.

b) After Triggers

These triggers are fired **after** the execution of the DML statements.

Syntax of a Trigger(ORACLE)

```
CREATE OR REPLACE TRIGGER <trigger name>
        before/after <event1 or event2 or event3>
        on <table>
        [for each row[when <condition>]]
[DECLARE
        <declarations>;]
BEGIN
        <action block>;
END;
/
```

Example (ORACLE)

```
CREATE OR REPLACE TRIGGER Trig_Insert
        AFTER insert OR update OR delete ON Employee for each row
BEGIN
        IF INSERTING THEN
                dbms_output.put_line('A new row is inserted.' ||:new.empno);
        ELSIF UPDATING THEN
                dbms_output.put_line('A row is updated.'||:new.empno);
        ELSE
                dbms_output.put_line('A new row is deleted.'||:old.empno);
        END IF;
END;
/
```

Syntax of a Trigger(SQL Server2000)

```
CREATE TRIGGER <trigger name>
        on <table>
        for <event1, event2, event3>
AS
[DECLARE
        <declarations>]

        <action block>
/
```

Example (SQL Server2000)

```
DROP TRIGGER Trig_Insert
go

CREATE TRIGGER Trig_Insert ON Employee FOR insert, update, delete
AS
        DECLARE @empno numeric
        SELECT @empno = empno FROM INSERTED
        SELECT @empno 'inserted row id'

go
```

INSERTED – This is a temporary table which holds the current data that is inserted into the table.

DELETED – This is a temporary table which holds the current data that is removed from the table.

Syntax of a Trigger(DB2UDB)

```
CREATE TRIGGER <trigger name>
  before/after <event1 or event2 event3>
  ON <table>
  [for each row]
BEGIN ATOMIC
        <action block>;
```

Example (DB2UDB)

```
DROP TRIGGER Trig_Insert
go

CREATE TRIGGER Trig_Insert after insert
ON Employee
REFERENCING new AS nnn old AS ooo
BEGIN ATOMIC
begin
        declare p_empno numeric;
        set p_empno = nnn;
        select p_empno from sysibm.sysdummy1;

end;
@
```

nnn/ooo – These are the alias names given to the inserted/updated values.

9 CURSORS

A cursor is a memory context area.
For every SQL Statement execution certain area in memory is allocated.
PL/SQL allows us to name that private area which is called as context area or cursor.
A cursor acts as a handle or pointer into the context area.
A PL/SQL program controls the context area using the cursor.

Cursors are classified in two ways

1. Implicit Cursors 2. Explicit Cursors

9.1 Implicit Cursors

For SQL Queries returning single rows PL/SQL declares implicit cursors.
These are the simple SQL statements and are written in the BEGIN block of the PL/SQL.
PL/SQL implicitly declares cursors for all DML statements.
The most commonly raised exceptions here are NO_DATA_FOUND or TOO_MANY_ROWS.

9.2 Explicit Cursors

Explicit cursors are used in queries that return multiple rows.
The set of rows fetched by a query is called active set.
The size of the active set meets the search criteria in the select statement.
Explicit cursor is declared in the DECLARE section of PL/SQL program.

Declaring Cursor

CURSOR <cursor-name> IS <select statement>

CURSOR emp_cur IS SELECT ename FROM EMP;

Opening Cursor

Syntax

OPEN <cursor-name>;

Example

OPEN emp_cur;

Fetching data from Cursor

Syntax

FETCH <cursor-name> INTO <variables>;

Example

FETCH emp_cur INTO ena;

Closing a Cursor

Syntax

CLOSE <cursor-name>;

Example

CLOSE emp_cur;

Explicit Cursor Attributes

1. %NOTFOUND - true – the last fetch failed.
2. %FOUND - true – the last fetch succeeded.
3. %ROWCOUNT – returns the number of rows fetched by the cursor so far.
4. %ISOPEN - true – if cursor is opened.
   - false– if cursor is closed.

Example for a EXPLICIT CURSOR:

```
DECLARE
        ena EMP.ENAME%TYPE
        esa EMP.SAL%TYPE
        CURSOR c1 IS SELECT ename, sal FROM EMP;
BEGIN
        OPEN c1;
        FETCH c1 INTO ena, esa;
        DBMS_OUTPUT.PUT_LINE(ena || ` Salary  is Rs ` || esa);
        FETCH c1 INTO ena, esa;
        DBMS_OUTPUT.PUT_LINE(ena || ` Salary  is Rs ` || esa);
        FETCH c1 INTO ena, esa;
        DBMS_OUTPUT.PUT_LINE(ena || ` Salary  is Rs ` || esa);
        CLOSE c1;
END;
/
```

Note : In SQL Server 2000 you have explicitly deallocate the cursor memory using the DEALLOCATE statement after closing the cursor.

PART - II

UTILITIES

BCP Utility in SQL Server2000

Using this utility you can import and export large amounts of data in and out of SQL Server 2000.
This utility is accessed from the command prompt.

BCP utility is explained below by taking an example.
Steps

- ✓ Go to Run-> cmd
- ✓ C:\>bcp pubs.dbo.authors OUT c:\bcp\authors.bcp -n –P
  This command exports the data from a table to a flat file with extension **.bcp**
  pubs – database name
  dbo – database owner name
  Authors – table name

  After you issue the command you will find a message in this manner –

  Starting copy...

  23 rows copied.
  Network packet size (bytes): 4096
  Clock Time (ms.): total 1

- ✓ C:\>bcp sample_dup.dbo.test1 in c:\bcp\test.bcp –n
  This command prompts you for the password of your account
  Password:

  This command is used to load the data into a table from a **.bcp** file
  After you issue the command you will find a message in this manner -

  Starting copy...

  5 rows copied.
  Network packet size (bytes): 4096
  Clock Time (ms.): total 1

  The file need not be .bcp file it can be a .txt file also

Export and Import

What is import/export and why does one need it?
The Oracle export (EXP) and import (IMP) utilities are used to perform logical database backup and recovery. They are also used to move Oracle data from one machine, database or schema to another.

How does one use the import/export utilities?
Look for the "imp" and "exp" executables in your $ORACLE_HOME/bin directory. One can run them interactively, using command line parameters, or using parameter files. Look at the imp/exp parameters before starting. These parameters can be listed by executing the following commands: "*exp help=yes*" or "*imp help=yes*".

```
exp scott/tiger file=emp.dmp log=emp.log tables=emp rows=yes indexes=no

exp scott/tiger file=emp.dmp tables=(emp,dept)

imp scott/tiger file=emp.dmp full=yes
imp scott/tiger file=emp.dmp fromuser=scott touser=scott tables=dept

exp userid=scott/tiger@orcl parfile=export.txt

... where export.txt contains:
                BUFFER=100000
                FILE=account.dmp
                FULL=n
                OWNER=scott
                GRANTS=y
                COMPRESS=y
```