Swami Sahajanand College of Computer Science

B.C.A. SEM-V[NEP]

Subject: DATABASE TECHNOLOGY IN INDIA
Major12 - 26516

UNIT-3

ADVANCE SQL*PLUS

1) Data Constrains
2) Types of Data Constrains.
3) In Built Functions: Aggregate, Numeric, String,
4) Data/Time, Conversion.
5) Grouping of Data

Q-1 What is Data Constraints? Also explain types of Data Constraints.

Data Constraints

- Oracle provides a special feature called data constraint/integrity.
- Constraint that is applied at the time of creation of data structure.
- Only the data which satisfies the constraints rules will be stored in database.
- If it is violating the data constraints, it must be rejected.
- This ensures that data stored within data structure are valid data.
- Constraints could be **column level** or **table level.**
- **Column level** constraints are applied only to one column per table.
- **Table level** constraints are applied to the whole table.

- Following are commonly used constraints available in SQL.

No.	Constraint Name	Description
1	NOT NULL	Ensures that a column cannot have NULL value.
2	DEFAULT	Provide a default value for a column.
3	UNIQUE	Ensures that all values in a column are different.
4	PRIMARY KEY	Uniquely identified each rows/records in a database table.
5	FOREIGN KEY	Uniquely identified a rows/records in any another database table.
6	CHECK	It ensures that all values in a column satisfy certain conditions.

(1) NULL or NOT NULL Constraint:

- NULL keyword indicates that a column can contain NULL values.
- The NOT NULL constraint specifies that a column cannot contain NULL value.
- To satisfy this constraint NULL, the column can contain NULLs by default.
- The NOT NULL constraint requires that the columns of the table always contain a value.
- Setting a NULL value is appropriate when the actual value is unknown.
- A NULL value is not same as a value zero.
- NULL value can be inserted into columns of any data type.

:: Syntax ::

```
create table <Table_Name>
(
<Column_Name 1><data type> (size) NULL,
<Column_Name 2><data type> (size) NOT NULL,
 …………………………………………………..
…………………………………………………..,
<Column_Name N><data type> (size)
);
```

:: Example ::

```
create table student
(
      IDnumber(5),
      NAME varchar2(30) NOT NULL,
      DOB date,
      MOBILE_NO number(10) NULL
);
```

Output: Table created.

- Here, NAME field cannot contain NULL value. Because we used NOT NULL constraint.
- MOBILE_NO can contain NULL value. Because we used NULL constraint.

(2) DEFAULT Constraint:

- The DEFAULT constraint provides a default value to a column.
- At the time of table creation a default value can be assign to it.
- When the user is loading a record with values and leaves this cell empty, the DBA will automatically load this cell with the default value specified.
- The data type of the default value should match the data type of the column.
- If INSERT INTO statement does not provide a specific value, then we used DEFAULT constraint.
- These constraint apply only column level.

:: Syntax ::

```
create table <Table_Name>
(
<Column_Name 1><data type> (size) default <value>,
<Column_Name 2><data type> (size),
 ……………………………………………………..
……………………………………………………..,
<Column_Name N><data type> (size)
);
```

:: Example ::

```
create table student
(
     IDnumber(5) default 111,
     NAME varchar2(30),
     DOB date,
     MOBILE_NO number(10)
);
Output: Table created.
```

- Here, ID field contain default value like 111.

(3) UNIQUE Constraint:

- The UNIQUE column constraint permits multiple entries of NULL into the column.
- Unique key will not allow duplicate value.
- Unique index is created automatically.
- A table can have more than one unique key which is not possible in PRIMARY KEY.
- Unique key can combine upto 16 columns in a Composite Unique Key.
- Unique key cannot be possible in LONG or LONG RAW data type.

  - **<u>Column Level:</u>**

:: Syntax ::

```
create table <Table_Name>
(
<Column_Name 1><data type> (size) UNIQUE,
<Column_Name 2><data type> (size),
 ……………………………………………………..
<Column_Name N><data type> (size)
);
```

:: Example ::

```
create table student
(
     IDnumber(5) UNIQUE,
     NAME varchar2(30),
     DOB date,
     MOBILE_NO number(10)
);
```

- **Table Level:**

:: Syntax ::

```
create table <Table_Name>
(
<Column_Name 1><data type> (size),
<Column_Name 2><data type> (size),
 ……………………………………………………..
<Column_Name N><data type> (size),
UNIQUE (< Column_Name 1>,<Column_Name 2>)
);
```

:: Example ::

```
create table student
(
      ID number(5) UNIQUE,
      NAME varchar2(30),
      DOB date,
      MOBILE_NO number(10),
      UNIQUE (ID, NAME)
);
```

(4) PRIMARY KEY Constraint:

"A PRIMARY KEY is used to uniquely identify each row in a table."

- A primary key is one or more column in a table.
- A primary key values must not be NULL and must be unique across the column.
- When you define any column as primary key it becomes a mandatory column.
- The column cannot be left blank.
- If single column is not sufficient to uniquely identify the row, you can use combination of two or more columns to uniquely identify a row.
- This combination of primary key is known as composite primary key.
- A table can have only one primary key.

PRIMARY KEY= UNIQUE+ NOT NULL

- **Features of Primary key:**

- Primary key is a column or a set of columns that uniquely identifies a row.
- It main purpose is the record uniqueness.
- Primary key will not allow duplicate values.
- Primary key will also not allow NULL values.
- Primary key is not compulsory but it is recommended.
- Primary key helps to identify one record from another record also helps in relation of table.
- Primary key cannot be possible in LONG or LONG RAW datatype.
- Only one primary key is allowed per table.
- Unique index is created automatically if there is a primary key.
- One table can combine upto 16 columns in a composite primary key.

- **Column Level:**

:: Syntax ::

```
create table <Table_Name>
(
<Column_Name 1><data type> (size) PRIMARY KEY,
<Column_Name 2><data type> (size),
 ……………………………………………………..
<Column_Name N><data type> (size)
);
```

:: Example ::

```
create table student
(
      IDnumber(5) PRIMARY KEY,
      NAME varchar2(30),
      DOB date,
      MOBILE_NO number(10)
);
```

- Here, only one filed ID has a primary key.

- **Table Level:**

:: Syntax ::

```
create table <Table_Name>
(
<Column_Name 1><data type> (size),
<Column_Name 2><data type> (size),
 ……………………………………………….….
<Column_Name N><data type> (size),
PRIMARY KEY (< Column_Name 1>,<Column_Name 2>)
);
```

:: Example ::

```
create table student
(
      ID number(5) UNIQUE,
      NAME varchar2(30),
      DOB date,
      MOBILE_NO number(10),
      PRIMARY KEY (ID, NAME)
);
```

- Here, ID and NAME both field have primary key.

(5) FOREIGN KEY Constraint:

- Foreign keys represent relationship between tables.
- A foreign key is a column whose values are derived from the primary key of some other table.
- The table in which the foreign key is defined is called a **foreign table** or **detail table**.
- The table that define the primary key or unique key and is referenced by the foreign key is called the **primary table** or **master table**.
- A foreign key can be defined in either a create table statement or an alter table statement.
- REFERENCES keyword is used for referencing table.

Features of Foreign key:

- Data type for relevant column in master and detail table must be same.
- Parent that is being referenced has to be unique or primary key.
- Child may have duplicates and nulls but unless it is specified.
- Foreign key constraint can be specified on child but not on parent.
- Deleting record from master table is not allowed if corresponding records are available in detail table.
- If ON DELETE CASCADE option is set delete operations on master will delete all records from detail table.
- Relationship can be established with primary key or unique key columns in master table.

- **Column Level:**

:: Syntax ::

```
create table <Table_Name>
(
<Column_Name 1><data type> (size) REFERENCES <Table_Name> (Column_Name),
<Column_Name 2><data type> (size),….., <Column_Name N><data type> (size)
);
```

:: Example ::

```
create table marksheet
(
      ID number(5) REFERENCES student(ID), Total number(3), Percentage number(5,2),
);
```

- **Table Level:**

:: Syntax ::

```
create table <Table_Name>
(
<Column_Name 1><data type> (size), <Column_Name 2><data type> (size),……………………….,
<Column_Name N><data type> (size),
FOREIGN KEY (Column1) REFERENCES <Table_Name> (Column_Name),
FOREIGN KEY (Column2) REFERENCES <Table_Name> (Column_Name)
);
```

:: Example ::

```
create table marksheet
(
      ID number(5), EID number(5), Total number(3), Percentage number(5,2),
      FOREIGN KEY (ID) REFERENCES student(ID),
      FOREIGN KEY (EID) REFERENCES employee(EID)
);
```

(6) CHECK Constraint:

- Business rule validation can be applied to a table column by using CHECK constraints.
- If you want to keep values of a column within a certain range then CHECK constraint is used.
- CHECK constraints must be specified as a logical expression that evaluate either True or False.
- For example: In a bank table balance column contains value >=500.
- So the condition defined in the constraint and permits the INSERT or UPDATE of the row in the table if the condition is satisfied

- **Column Level:**

:: Syntax ::

```
create table <Table_Name>
(
<Column_Name 1><data type> (size) CHECK (Logical Expression),
<Column_Name 2><data type> (size),….., <Column_Name N><data type> (size)
);
```

:: Example ::

```
create table marksheet
(
      ID number(5) CHECK(ID>0), Total number(3), Percentage number(5,2),
);
```

- Here, ID field contain value greater than zero.

- **Table Level:**

:: Syntax ::

```
create table <Table_Name>
(
<Column_Name 1><data type> (size), <Column_Name 2><data type> (size),……………………..,
<Column_Name N><data type> (size),
CHECK (Column1 [Logical Expression]), CHECK (Column2 [Logical Expression]),……………..
);
```

:: Example ::

```
create table student
(
      ID number(5), NAME varchar2(30), EID number(3), BALANCEnumber(10,2),
      CHECK(ID>0), CHECK(NAME=UPPER(NAME)), CHECK(BALANCE>=500)
);
```

- Here, ID field contain value greater than zero.
- NAME must be entered in Capital Letters only.
- BALANCE field contain value >=500.

Q-2 Explain InBuilt Functions: Aggregate, Numeric, String, Date/Time, Conversion in detail.

❖ **DUAL:**

- Dual is a small oracle inbuilt table. The table is owned by SYS and it is available for the all users.
- The dual table provides only one row and one column in it.
- It supports arithmetic calculation and data formatting.
- It is also known as DUMMY table.
- **Example**: - select 4*5 “multiply” from dual;
- **Output**: - multiply 20
- **Example**: - select sysdate from dual; **Output**: - 23-Jun-2018

❖ **FUNCTION:**

- Function are pre-defined set of subroutines that may operate on one or more rows.
- Basically a function takes some data input as an argument.
- Processes that data and returns some values as a result.
- Function can divided into following categories:
  1) Aggregate Functions.
  2) Numeric Functions.
  3) String Functions.
  4) Date/Time Functions.
  5) Conversion Functions.

(1) Aggregate Functions:

1) sum():

Syntax	sum(value)
Purpose	It returns the total of given numeric expression.
Example	select sum(salary) “Total_salary” from emp;
Output	Total_salary 29000

2) avg():

Syntax	avg(value)
Purpose	It returns the average value of N and it ignore the NULL value.
Example	select avg(salary) from emp;
Output	5000.589

3) min():

Syntax	min(value)
Purpose	It returns the minimum values from given list of values.
Example	select min(salary) from emp;
Output	500

4) max():

Syntax	max(value)
Purpose	It returns the maximum values from given list of values.
Example	select max(salary) from emp;
Output	50000

5) count(*):

Syntax	count(*)
Purpose	It returns total number of records in the table, including duplicate and null value.
Example	select count(*) “total no of rows” from emp;
Output	total no of rows 10

6) count():

Syntax	count(value)
Purpose	It returns total number of records based on value, if value is null it will not be included in counting.
Example	select count(salary) “total no of salary” from emp;
Output	total no of salary 8

(2) Numeric Functions:

1) abs():

Syntax	abs(value)
Purpose	Returns absolute value of the given number
Example	select abs(-15)from dual;
Output	15

2) ceil():

Syntax	ceil(value)
Purpose	Returns smallest integer value that is greater than or equal to given value.
Example	select ceil(15.20)from dual;
Output	16

3) floor():

Syntax	floor(value)
Purpose	Returns smallest integer value that is less than or equal to given value.
Example	select floor(15.20)from dual;
Output	15

4) mod():

Syntax	mod(value, divisor)
Purpose	It returns remainder value.
Example	select mod(5,2)from dual;
Output	1

5) power():

Syntax	power(x, n)
Purpose	It returns x raise to n value. (x^n)
Example	Select power(5,2)from dual;
Output	25

6) sqrt():

Syntax	sqrt(value)
Purpose	It returns square root of given value.
Example	select sqrt(25) from dual;
Output	5

7) round():

Syntax	round(value, precision)
Purpose	It rounds the value up to given precision value.
Example	select round(15.456,2)from dual;
Output	15.46

8) trunc():

Syntax	trunc(value, precision)
Purpose	It returns a number truncated to a certain number of decimal places.
Example	select trunc(15.456,1)from dual;
Output	15.4

9) sign():

Syntax	sign(value)
Purpose	This function tells you the sign of value. 1 for positive value. -1 for negative value and 0 for 0 value.
Example	select sign(-5)from dual;
Output	-1 [sign(-5)=-1, sign(5)=1, sign(0)=0]

10) greatest():

Syntax	greatest (expr1,expr2, …)
Purpose	Returns the greatest [Highest] value from list of expression.
Example	select greatest(10,30,20) from dual;
Output	30

11) least():

Syntax	least (expr1,expr2, …)
Purpose	Returns the least [Lowest] value from list of expression.
Example	select least(10,30,20) from dual;
Output	10

12) exp():

Syntax	exp(value)
Purpose	Returns e raised to the nth power, where e=2.71828183.
Example	select exp(5)from dual;
Output	148.413159 [2.7182^5]

(3) String Functions:

1) upper():

Syntax	upper(string)
Purpose	It returns set of character with all the letters in the upper case.
Example	select upper('oracle') from dual;
Output	ORACLE

2) lower():

Syntax	lower(string)
Purpose	It returns set of character with all the letters in the lower case.
Example	select upper('ORACLE') from dual;
Output	Oracle

3) initcap():

Syntax	initcap(string)
Purpose	Returns a string with the first letter of each word in upper case.
Example	select initcap('hello how are you') from dual;
Output	Hello How Are You

4) substr():

Syntax	substr(string, m,n)
Purpose	It returns the specific part of a given string. M indicates the starting position and N indicates total number of character.
Example	Select substr(‘ORACLE’,3,4) from dual;
Output	ACLE

5) length():

Syntax	length(string)
Purpose	It returns the number of character in a given string.
Example	Select length(‘BCA’) from dual;
Output	3

6) ltrim():

Syntax	ltrim(string, [character_set])
Purpose	Removes characters from the left of char with initial characters removed upto the first character not in set.
Example	Select ltrim(‘BCA’,’B’) from dual;
Output	CA

7) rtrim():

Syntax	rtrim(string, [character_set])
Purpose	Returns char, with final characters removed after the last character not in the set. Set is optional, it defaults to spaces.
Example	Select rtrim(‘RAMA’,’A’) from dual;
Output	RAM

8) trim():

Syntax	ltrim(leading\trailing\both[<trim_character>] FROM string)
Purpose	Removes characters from leading, trailing and both the side.
Example	Select trim(‘ BCA ‘) from dual;
Output	BCA

9) lpad():

Syntax	lpad(string, length, [character_set])
Purpose	It returns string after adding character set to the left of the string by the number specified in length.
Example	Select lpad(‘BCA’, 10,’*’) from dual;
Output	*******BCA

10) rpad():

Syntax	rpad(string, length, [character_set])
Purpose	It returns string after adding character set to the right of the string by the number specified in length.
Example	Select rpad(‘BCA’, 10,’*’) “rpad” from dual;
Output	BCA*******

11) instr():

Syntax	instr(string1,string2, [start position], [occurrence])
Purpose	It returns the occurrence of character in a given string.
Example	Select instr(‘information’, ‘n’,1,1) “occurrence-1”, instr(‘information’, ‘n’,1,2) “occurrence-2” from dual;
Output	occurrence-1 occurrence-2 2 11

12) translate():

Syntax	translate(string1,<string_to_replace>, <replacement_string>)
Purpose	Replaces a sequence of characters in a string with another set of characters.
Example	Select translate(‘ORACLE’,’OR’,’12’) from dual;
Output	12ACLE

(4) Date Functions:

- Oracle provides various functions to perform operations on date data type.
- You should be aware that there is a special keyword called sysdate to know the current data.
- **Ex**: select sysdate from dual;
- **Output**: 15-jun-18

1) add_months():

Syntax	add_months(date, n)
Purpose	Returns date after adding n month in date.
Example	Select add_months(’10-apr-17’,5) from dual;
Output	10-sep-17

2) last_day():

Syntax	last_day(date)
Purpose	Returns the last date of the month specified in function argument.
Example	Select last_day(’10-apr-17’) from dual;
Output	30-apr-17

3) months_between():

Syntax	months_between(date1,date2)
Purpose	Returns numbers of months between two dates date1 and date2.
Example	Select months_between(’10-apr-17’,’10-feb-17) from dual;
Output	2

4) next_day():

Syntax	next_day(date, ’day’)
Purpose	Returns the date for specified day coming after the date specified in the function argument.
Example	Select next_day(’23-dec-12’,’sunday’) from dual;
Output	30-dec-12

(5) Conversion Functions:

- Conversion functions are used to change one data type to other data type.

1) to_number():

Syntax	to_number(string)
Purpose	Converts character data type to number data type.
Example	Select to_number(substr(’$100’,2,3)) from dual;
Output	100

2) to_char():

Syntax	to_char(number, ’format’)
Purpose	Converts number data type to character data type.
Example	Select to_char(144530,’$099,999’) from dual;
Output	$144,530

3) to_char():

Syntax	to_char(date, ’format’)
Purpose	Returns date in given format.
Example	Select to_char(sysdate,’dd-mm-yyyy’) from dual;
Output	15-05-2018

4) to_date():

Syntax	to_date(string, ’format’)
Purpose	Converts string to date value.
Example	Select to_date(’14/12/2017’,’dd-mm-yy’) from dual;
Output	14-dec-17

♦ Q-3 Explain Grouping Data in detail.

- **Group By:**
  - To create summary information or to group a data based on columns value you can use GROUP BY.
  - While grouping data you must use grouping function expression.
  - The GROUP BY clause creates a data set, containing several sets of records grouped together based on a condition.
  - The SQL GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data into groups.
- **Features of group by clause:**
  - Group by clause comes after where clause if there is no ‘where’ clause it comes after from clause.
  - Group by partition the table row and create sub group within the same value in group by column.
  - Group by treats null value as separate column.
  - Group by will first sort the table on the group by column.
  - Only aggregate function can be used with group by clause. Such as sum, min, max, count, etc.
  - Column listed in group by clause need not be listed in select statement.

- **Syntax**

```
Select <column_name1>, <column_name2>,…., <Column_NameN>
Aggregate_Function (<Expression>) From <Table_Name> Where <Condition>
Group By <column_name1>, <column_name2>,…., <Column_NameN>;
```

- **Example-1**

```
Select city, count (city) from student group by city;
```

Here, above command will display city name and no. of student living in particular city.

City	Count (City)
Bhavnagar	5
Baroda	4
Surat	3

- **Example-2**

```
Select Branch_No, count (No_of_Employee) from Emp_master group by Branch_No;
```

Here, above command will display the total no of employees in each branch.

Branch_No	No_of_Employee
B1	2
B2	5
B3	4
B4	3

- **Having Clause:**
  - ➢ Having clause is used to filter a group data.
  - ➢ It is like a WHERE clause.
  - ➢ Only difference that where clause is used to filter individual row data while having clause is used to filter group data.
  - ➢ Having clause is use if and only if there is grouping clause.
- **Syntax**

```
Select <column_name1>, <column_name2>,…., <Column_NameN>
Aggregate_Function (<Expression>) From <Table_Name> Where <Condition>
Group By <column_name1>, <column_name2>,…., <Column_NameN>
Having (Condition);
```

- **Example**

```
Select city, count (city) from student group by city having count (city)>3;
```

Here, above command will display city name and no. of student living in particular city but more than 3.

City	Count (City)
Bhavnagar	5
Baroda	4