C.P.PATEL & F.H.SHAH COMMERCE COLLEGE
(MANAGED BY SARDAR PATEL EDUCATION TRUST)
BCA, BBA (ITM) & PGDCA PROGRAMME:

**BBA (ITM) Semester V - Paper Code: UM05CBBI09**

**UNIT 2- Basic SQL Concepts**

| Sr. No. | Topics |
|---|---|
| 1 | Built-in Data Types – (Number, Char, Varchar2, Date); |
| 2 | Creating Table and Inserting Data, Retrieving Data Using Query, Manipulating |
| 3 | Data using DELETE and UPDATE; Modifying table structure, Removing table, |
| 4 | Pseudo Columns – ROWID, ROWNUM, USER, SYSDATE, Null values, |
| 5 | TAB table, DUAL table, |
| 6 | Operators – Arithmetic, Relational, Logical, Range Searching, Pattern Matching and Set operators |

**DATATYPE(S):**

**1. Character Data types**

Character data types are used to manipulate words and free-form text. These data types are used to store character (alphanumeric) data in the database or national character set. They are less restrictive than other data types and consequently have fewer properties.

These data types are used for character data:

- CHAR
- VARCHAR
- VARCHAR2

**CHAR Data type**

The CHAR data type specifies a fixed length character string. When you create a table with a CHAR column, you supply the column length in bytes. Oracle subsequently ensures that all values stored in that column have this length. If you insert a value that is shorter than the column length, Oracle blank-pads the value to column length. If you try to insert a value that is too long for the column, Oracle returns an error.

The default length for a CHAR column is 1 character and the maximum allowed is 2000 characters. A zero-length string can be inserted into a CHAR column, but the column is blank-padded to 1 character when used in comparisons.

**VARCHAR2 Datatype**

The VARCHAR2 datatype specifies a variable length character string. When you create a VARCHAR2 column, you can supply the maximum number of bytes of data that it can hold. Oracle subsequently stores each value in the column exactly as you specify it, provided it does not exceed the column's maximum length. This maximum must be at least 1 byte,

although the actual length of the string stored is permitted to be zero. If you try to insert a value that exceeds the specified length, Oracle returns an error.

You must specify a maximum length for a VARCHAR2 column. The maximum length of VARCHAR2 data is 4000 bytes.

**VARCHAR Datatype**

The VARCHAR datatype is currently synonymous with the VARCHAR2 datatype. It is recommended that you use VARCHAR2 rather than VARCHAR. In a future version of Oracle, VARCHAR might be a separate datatype used for variable length character strings compared with different comparison semantics.

**2. NUMBER Datatype**

The NUMBER datatype is used to store zero, positive and negative fixed and floating point numbers with magnitudes between $1.0 \times 10^{-130}$ and $9.9...9 \times 10^{125}$ (38 9s followed by 88 0s) with 38 digits of precision. If you specify an arithmetic expression whose value has a magnitude greater than or equal to $1.0 \times 10^{126}$, Oracle returns an error.

You can specify a fixed-point number using the following form:

NUMBER (p,s)

Where,

p is the *precision*, or the total number of digits. Oracle guarantees the portability of numbers with precision ranging from 1 to 38.

s is the scale, or the number of digits to the right of the decimal point. The scale can range from -84 to 127.

The following examples show how Oracle stores data using different precisions and scales.

**3. LONG Datatype**

LONG columns store variable length character strings containing up to 2 gigabytes, or $2^{31}$-1 bytes. LONG columns have many of the characteristics of VARCHAR2 columns. You can use LONG columns to store long text strings. Oracle uses LONG columns in the data dictionary to store the text of view definitions. The length of LONG values may also be limited by the memory available on your computer.

The use of LONG values is subject to some restrictions:

- A table cannot contain more than one LONG column.
- LONG columns cannot appear in integrity constraints (except for NULL and NOT NULL constraints).
- LONG columns cannot be indexed.
- A stored function cannot return a LONG value.
- Within a single SQL statement, all LONG columns, updated tables, and locked tables must be located on the same database.

**4. DATE Datatype**

The DATE datatype is used to store date and time information. Although date and time information can be represented in both CHAR and NUMBER datatypes, the DATE datatype has special associated properties.

For each DATE value the following information is stored:

- century
- year
- month
- day
- hour
- minute
- second

**5. RAW and LONG RAW Datatypes**

The RAW and LONG RAW datatypes are used for data that is not to be interpreted (not converted when moving data between different systems) by Oracle. These data types are intended for binary data or byte strings. For example, LONG RAW can be used to store graphics, sound, documents, or arrays of binary data; the interpretation is dependent on the use.

LONG RAW data cannot be indexed, but RAW data can be indexed.

**6. Large Object (LOB) Data types**

Internal LOB data types, BLOB, CLOB, NCLOB, and external data type BFILE, can store large and unstructured data such as text, image, video, and spatial data, up to four gigabytes in size.

You can define one or more LOB data type columns in a table.

**7. ROWID**

For each row in the database, the ROWID pseudo column returns a row's address.

Usually, a ROWID value uniquely identifies a row in the database. However, rows in different tables that are stored together in the same cluster can have the same ROWID.

Values of the ROWID pseudo column have the data type ROWID.

ROWID values have several important uses:

- They are the fastest way to access a single row.
- They can show you how a table's rows are stored.
- They are unique identifiers for rows in a table.

You should not use ROWID as a table's primary key. If you delete and reinsert a row with the Import and Export utilities, for example, its ROWID may change. If you delete a row, Oracle may reassign its ROWID to a new row inserted later.

Although you can use the ROWID pseudo column in the SELECT and WHERE clauses of a query, these pseudo column values are not actually stored in the database. You cannot insert, update, or delete a value of the ROWID pseudo column.

Example: Select ROWID from EPM;

- **CREATE TABLE**

TYPE: DDL

SYNTAX:

CREATE TABLE tablename

(columnname datatype(size), columnname datatype(size)….);

Whenever user creates table, the table name must be a legal SQL name and it must not conflict with the name of any of your existing tables.

e.g.

CREATE TABLE stud

(s_no NUMBER(3), s_name VARCHAR2(15), address VARCHAR2(20) );

Creating a table from previously created table.

SYNTAX:

CREATE TABLE tablename1 (columnanme, columnname…..) AS SELECT columnname,columnname….. FROM tablename2

[WHERE <condition>]

Note: If the tablename2 is having records in it then the target table – tablename1 is populated with records well.

e.g CREATE TABLE stud1 AS SELECT * FROM STUD;

Create duplicate table with empty:

CREATE TABLE <new tablename> AS SELECT * FROM <old tablename> WHERE 1=2

- **DROP TABLE**

TYPE: DDL

SYNTAX: DROP TABLE tablename;

Removing the table and all its data and constraints from the database.

Example: DROP TABLE stud;

- **INSERTION OF DATA INTO TABLES**

TYPE: DML

Purpose: To add rows/records to a table or a view's base table.

SYNTAX: INSERT INTO <tablename> [columnname, ……]

VALUES (<expr1>,……);

## Bind variables:

1. &variable_name
   It will not preserve value to a variable. Once the value is transferred into a specific column each time it will ask the user to input users to input value.

INSERT INTO <tablename> [ <columnname>,…..] VALUES (&S_NO,….);

2. &&variable_name

It will preserve value to a variable. It is used for define variable first time. It will not ask user second time and previous values will automatically been assigned to the specific field.

INSERT INTO <tablename> [ <columnname>,…..] VALUES (&&S_NO,&&SNAME,….);

Inserting data into a table from another table:

INSERT INTO tablename SELECT columnname,columnname FROM tablename [ WHERE <condition> ];

- **UPDATING A CONTENTS OF A TABLE:**

  TYPE: DML

  SYNTAX:

  UPDATE <tablename>

  SET columnname = <expr>, columnname =<expr>

  [ WHERE <condition> ];

  Changing the existing values in a table or in a view's base table.

- **DELETION OPERATIONS:**

  TYPE: DML

  SYNTAX:

  DELETE [FROM] <tablename> [WHERE <condition>];

  To remove row(s) from table:

  - DELETE FROM <tablename>;
    Will delete all the records from a table name specified
  - DELETE FROM <tablename> WHERE <condition>
    Will delete all the records, which satisfies the condition specified after where.

- **TRUNCATE**

  TYPE: DML

  SYNTAX: TRUNCATE TABLE <tablename>;

  Will delete all the records from a table and records cannot be ROLLBACK.

- **RENAME**

  TYPE: DDL

  SYNTAX: RENAME <oldname> TO <newname>;

  User can rename a table.

- **SELECT COMMAND:**

  TYPE: DML

  SYNTAX:

  - SELECT * FROM **tab***;*
    Will display the information of all the tables, views in the current database.
  - SELECT ***** FROM tablename*;*
    Will display all the records from a table with all the fields.
  - SELECT **columnname, columnname** FROM tablename;
    Will display all the records with specific fields that user has specified with SELECT command.

- SELECT * FROM <tablename> **WHERE** <condition>*;*
  Will display all the records with given condition specified after WHERE will become true.
- SELECT **DISTINCT** columname FROM tablename*;*
  Will eliminate the duplicate records for column specified after **DISTINCT**
- *SELECT columnname, columnname FROM <tablename>*
  **ORDER BY** columnname , columnname *;*

**Sorting** of data in a table.

**SELECT columnname, columnname FROM <tablename>**
**ORDER BY** <expN> [ASCE | DESC];

Here <expN> specifies that the field number that means user can specify field number instead of specifying the columnname.

ASCE means sort the records in an ascending order.

DESC means sort the records in a descending order.

- **VIEWING THE TABLE STRUCTURE:**

SYNTAX:

DESCRIBE <table name> OR DESC <table name>

- **MODIFYING THE STRUCTURE OF TABLE:**

TYPE: DDL
ADDING NEW COLUMNS:
SYNTAX:

ALTER TABLE <tablename> ADD
(newcolumnname datatype(size), newcolumnname datatype(size));

MODIFYING EXISTING COLUMNS:

ALTER TABLE <tablename> MODIFY
(Columnname newdatatype (size));

DELETING COLUMNS

ALTER TABLE <tablename> DROP COLUMN <columnname>;

Restrictions on the ALTER table:

- User cannot change the column name
- User cannot decrease the size of column if record(s) there in the table for that column.
- Earlier versions do not support the deletion of column(s).

- **NULL VALUE CONCEPTS:**

While creating tables, if a row lacks a data value for a particular column, that value is said to be *null*. Columns of any data types may contain null values unless the column was defined *not null* when table was created.

Principles of NULL values:

- Setting a null value is appropriate when the actual value is unknown, or when a value would not be meaningful.
- A null value is not equivalent to a value of zero.

- A null value will evaluate to null in any expression. E.g. null multiply by 10 is null.
- When a column name is defined as not null, then that column becomes a mandatory column. It implies that the user is forced to enter data into the column.

Example:

```
CREATE TABLE client_mst
(client_no VARCHAR2(6) NOT NULL,
name VARCHAR2(15) NOT NULL,
address1 VARCHAR2(20));
```

❖ **Arithmetic operators**

| | | | |
|---|---|---|---|
| + | Addition | / | Division |
| - | Subtraction | ( ) | Enclosed operation |
| * | Multiplication | ** | Exponential |

Example:

Select employee name, salary and compute salary * 0.05 for each row retrieved:

```
SELECT ename, sal, sal*0.05 "Salary" from emp;
```

OR

```
SELECT ename, sal, sal*0.05 AS "Salary" from emp;
```

Here the default output **column sal*0.05 is renamed with *Salary*.**

❖ **Relational operators**

| | | | |
|---|---|---|---|
| = | Equal to | | |
| < | Less than | <= | Less than or equal to |
| > | Greater than | >= | Greater than or equal to |
| != | Not equal to | <> | Not equal to |

#Not equal to

❖ **Logical operators**

The logical operators that can be used in SQL sentences are:

| | |
|---|---|
| and | *all of* must be included |
| or | *any of* may be included |
| not | *none of* would be included |

❖ **Range Searching (IN, BETWEEN, IS)**

- Select employee number, employee name, department number from emp table where the department number is 10 or 20

```
SELECT empno, ename, deptno FROM EMP

WHERE deptno 10 or deptno=20;

OR

SELECT empno, ename, deptno FROM EMP

WHERE deptno IN (20,30);
```

- Select employee number, employee name, department number from emp table where the department number lies in 15 and 25 (both inclusive)

```
SELECT empno, ename, deptno FROM EMP
WHERE deptno >=15 and deptno<=25;
OR
SELECT empno, ename, deptno FROM EMP WHERE
deptno BETWEEN 15 AND 25;
```

- Select all the records from emp table who receives commission

```
SELECT * FROM emp WHERE comm IS NOT null;
```

❖ **Pattern matching**

Select the records of that employee whose name starts with first character 'B' and third character 'A'

```
SELECT * FROM emp where

ename like 'B_A%';
```

Here '_' indicated that second character from ename may be any thing but the first character must be 'B' and the third character must be 'A' also '%' indicates that the rest of the name after third character may be anything and can be of any length.