# 8. INTERACTIVE SQL PART – II

## DATA CONSTRAINTS

All businesses of the world run on business data being gathered, stored and analyzed. Business managers determine a set of business rules that must be applied to their data prior to it being stored in the database/table to ensure its integrity.

For instance, no employee in the sales department can have a salary of less than Rs.1000/-.

Such rules have to be enforced on data stored. Only data, which satisfies the conditions set, should be stored for future analysis. If the data gathered fails to satisfy the conditions set, it must be rejected. This ensures that the data stored in a table will be valid, and have integrity.

Business rules that are applied to data are completely **System dependent**. The rules applied to data gathered and processed by a **Savings bank system** will be very different, to the business rules applied to data gathered and processed by an **Inventory system**, which in turn will be very different, to the business rules applied to data gathered and processed by a **Personnel management system**.

Business rules, which are enforced on data being stored in a table, are called **Constraints**. Constraints, **super control** the data being entered into a table for permanent storage.

To understand the concept of data constraints, several tables will be created and different types of constraints will be applied to table columns **or** the table itself. The set of tables are described below. Appropriate examples of data constraints are bound to these tables.

### Applying Data Constraints

Oracle permits data constraints to be attached to table columns via SQL syntax that checks data for integrity prior storage. Once data constraints are part of a table column construct, the Oracle database engine checks the data being entered into a table column against the data constraints. If the data passes this check, it is stored in the table column, else the data is rejected. Even if a single column of the record being entered into the table fails a constraint, the **entire record is rejected and not stored in the table.**

Both the **Create Table** and **Alter Table** SQL verbs can be used to write SQL sentences that attach constraints (i.e. Business / System rules) to a table column.

**Caution**


Until now tables created in this material have **not** had any data constraints attached to their table columns. Hence the tables have **not** been given any instructions to filter what is being stored in the table. This situation **can** and **does**, result in erroneous data being stored in the table.

Once a constraint is attached to a table column, any SQL **INSERT** or **UPDATE** statement automatically causes these constraints to be applied to data prior it is being inserted into the table column for storage.

**Note**

Oracle also permits applying data constraints at **Table level**. More on table level constraints later in this material.

## TYPES OF DATA CONSTRAINTS

There are two types of data constraints that can be applied to data being inserted into a Oracle table. One type of constraint is called an **I/O** constraint (**input / output**). This data constraint determines the speed at which data can be inserted or extracted from a Oracle table. The other type of constraint is called a **business rule** constraint.

### I/O Constraints

The input/output data constraints are further divided into **two** distinctly different constraints.

### The PRIMARY KEY Constraint

A primary key is one or more column(s) in a table used to uniquely identify **each row** in the table. None of the fields that are part of the primary key can contain a null value. A table can have only one primary key. A **primary key column** in a table has special attributes:

- It defines the column, as a mandatory column (**i.e.** the column cannot be left blank). As the NOT NULL attribute is active
- The data held across the column MUST be UNIQUE

A single column primary key is called a **Simple** key. A multicolumn primary key is called a **Composite** primary key. The only function of a primary key in a table is to **uniquely identify a row**. When a record cannot be uniquely identified using a value in a simple key, a composite key must be defined. A primary key can be defined in either a **CREATE TABLE** statement or an **ALTER TABLE** statement.

For example, a **SALES_ORDER_DETAILS** table will hold multiple records that are sales orders. Each such sales order will have multiple products that have been ordered. Standard business rules do not allow multiple entries for the same product. However, multiple orders will definitely have multiple entries of the same product.

Under these circumstances, the only way to uniquely identify a row in the **SALES_ORDER_DETAILS** table is via a composite primary key, consisting of **ORDER_NO** and **PRODUCT_NO**. Thus the combination of order number and product number will uniquely identify a row.

**Features of Primary key**

1. Primary key is a column or a set of columns that uniquely identifies a row. Its main purpose is the **Record Uniqueness**
2. Primary key will not allow duplicate values
3. Primary key will also not allow null values
4. Primary key is not compulsory but it is recommended
5. Primary key helps to identify one record from another record and also helps in relating tables with one another
6. Primary key cannot be LONG or LONG RAW data type
7. Only one Primary key is allowed per table
8. Unique Index is created automatically if there is a Primary key
9. One table can combine upto 16 columns in a Composite Primary key

### *PRIMARY KEY Constraint Defined At Column Level*

**Syntax:**

```
<ColumnName> <Datatype>(<Size>) PRIMARY KEY
```

**Example 1:**
Drop the CUST_MSTR table, if it already exists. Create a table CUST_MSTR such that the contents of the column CUST_NO is unique and not null.

```
DROP TABLE CUST_MSTR;
CREATE TABLE CUST_MSTR (
    "CUST_NO" VARCHAR2(10) PRIMARY KEY,
    "FNAME" VARCHAR2(25), "MNAME" VARCHAR2(25),
    "LNAME" VARCHAR2(25), "DOB_INC" DATE,
    "OCCUP" VARCHAR2(25), "PHOTOGRAPH" VARCHAR2(25),
    "SIGNATURE"  VARCHAR2(25), "PANCOPY" VARCHAR2(1),
    "FORM60" VARCHAR2(1));
```

**Output:**
```
Table created.
```

For testing purpose, execute the following **INSERT INTO** statement:
```
INSERT INTO CUST_MSTR (CUST_NO, FNAME, MNAME, LNAME, DOB_INC, OCCUP, PHOTOGRAPH,
                       SIGNATURE, PANCOPY, FORM60)
    VALUES('C1', 'Ivan', 'Nelson', 'Bayross', '25-JUN-1952', 'Self Employed', 'D:/ClntPht/C1.gif',
           'D:/ClntSgnt/C1.gif', 'Y', 'Y');
```

**Output:**
```
1 row created.
```

To verify whether the Primary Key Constraint is functional, reissue the same **INSERT INTO** statement. The result is the following error:

**Output:**
```
INSERT INTO CUST_MSTR (CUST_NO, FNAME, MNAME, LNAME, DOB_INC, OCCUP,
PHOTOGRAPH, SIGNATURE, PANCOPY,
*
ERROR at line 1:
ORA-00001: unique constraint (DBA_BANKSYS.SYS_C003009) violated
```

## *PRIMARY KEY Constraint Defined At Table Level*

**Syntax:**
```
PRIMARY KEY (<ColumnName>, <ColumnName>)
```

**Example 2:**
Drop the FD_MSTR table, if it already exists. Create a table FD_MSTR where there is a composite primary key mapped to the columns FD_SER_NO and CORP_CUST_NO.
Since this constraint spans across columns, it must be described at table level.

```
DROP TABLE FD_MSTR;
CREATE TABLE "DBA_BANKSYS"."FD_MSTR"(
    "FD_SER_NO" VARCHAR2(10), "SF_NO" VARCHAR2(10),
    "BRANCH_NO" VARCHAR2(10), "INTRO_CUST_NO" VARCHAR2(10),
    "INTRO_ACCT_NO" VARCHAR2(10), "INTRO_SIGN" VARCHAR2(1),
    "ACCT_NO" VARCHAR2(10), "TITLE" VARCHAR2(30),
    "CORP_CUST_NO" VARCHAR2(10), "CORP_CNST_TYPE" VARCHAR(4),
    "VERI_EMP_NO" VARCHAR2(10), "VERI_SIGN" VARCHAR2(1),
    "MANAGER_SIGN" VARCHAR2(1), PRIMARY KEY(FD_SER_NO, CORP_CUST_NO));
```

**Output:**
```
Table created.
```

For testing purpose, execute the following **INSERT INTO** statement:
```
INSERT INTO FD_MSTR (FD_SER_NO, SF_NO, BRANCH_NO, ACCT_NO, TITLE, CORP_CUST_NO,
                     CORP_CNST_TYPE, INTRO_CUST_NO, INTRO_ACCT_NO, INTRO_SIGN,
                     VERI_EMP_NO, VERI_SIGN, MANAGER_SIGN)
  VALUES ('FS1', 'SF-0011', 'B1', 'CA2', 'Uttam Stores', 'O11', '1C', null, null, 'N', 'E1', 'Y', 'Y');
```

**Output:**
```
1 row created.
```

To verify whether the Composite Primary Key Constraint is functional, reissue the same **INSERT INTO** statement. The result is the following error:

**Output:**
```
INSERT INTO FD_MSTR (FD_SER_NO, SF_NO, BRANCH_NO, ACCT_NO, TITLE,
CORP_CUST_NO, CORP_CNST_TYPE, INTR
*
ERROR at line 1:
ORA-00001: unique constraint (DBA_BANKSYS.SYS_C003010) violated
```

Now, simply modify the INSERT INTO statement as show below, to allow the record to pass the composite primary key constraint:
```
INSERT INTO FD_MSTR (FD_SER_NO, SF_NO, BRANCH_NO, ACCT_NO, TITLE, CORP_CUST_NO,
                     CORP_CNST_TYPE, INTRO_CUST_NO, INTRO_ACCT_NO, INTRO_SIGN,
                     VERI_EMP_NO, VERI_SIGN, MANAGER_SIGN)
  VALUES ('FS2', 'SF-0012', 'B1', 'CA4', 'Sun"s Pvt. Ltd.', 'C12', '4C', null, null, 'N', 'E1', 'Y', 'Y');
```

**Output:**
```
1 row created.
```

## The Foreign Key (Self Reference) Constraint

Foreign keys represent relationships between tables. A foreign key is a column (or a group of columns) whose values are derived from the **primary key** or **unique 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 defines the **primary 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

The master table can be referenced in the foreign key definition by using the clause **REFERENCES TableName.ColumnName** when defining the foreign key, column attributes, in the detail table.

**Features of Foreign Keys**

1. Foreign key is a column(s) that references a column(s) of a table and it can be the same table also
2. Parent that is being referenced has to be unique or Primary key
3. Child may have duplicates and nulls but unless it is specified
4. Foreign key constraint can be specified on child but not on parent
5. Parent record can be delete provided no child record exist
6. Master table cannot be updated if child record exist

This constraint establishes a relationship between records (i.e. column data) across a Master and a Detail table. This relationship ensures:

- Records cannot be **inserted** into a **detail table** if corresponding records in the master table do not exist
- Records of the master **table** cannot be **deleted** if corresponding records in the detail table actually exist

### Insert Or Update Operation In The Foreign Key Table

The existence of a foreign key implies that the table with the foreign key is **related** to the master table from which the foreign key is derived. A foreign key must have a corresponding primary key or unique key value in the master table.

For example a personnel information system includes two tables (i.e. department and employee). An employee cannot belong to a department that does not exist. Thus the department number specified in the employee table must be present in the department table.

### Delete Operation On The Primary Key Table

Oracle **displays an error message** when a record in the master table is deleted and corresponding records exists in a detail table and **prevents the delete operation** from going through.

**Note**

The default behavior of the foreign key can be changed, by using the **ON DELETE CASCADE** option. When the **ON DELETE CASCADE** option is specified in the foreign key definition, if a record is deleted in the master table, all corresponding records in the detail table along with the record in the master table will be deleted.

Principles of **Foreign Key/References** constraint:

- Rejects an **INSERT** or **UPDATE** of a value, if a corresponding value does not currently exist in the master key table
- If the **ON DELETE CASCADE** option is set, a **DELETE** operation in the master table will trigger a **DELETE** operation for corresponding records **in all** detail tables
- If the **ON DELETE SET NULL** option is set, a **DELETE** operation in the master table will set the value held by the foreign key of the detail tables to null
- **Rejects** a **DELETE** from the Master table if **corresponding records** in the DETAIL table **exist**
- Must reference a **PRIMARY KEY** or **UNIQUE** column(s) in primary table
- Requires that the **FOREIGN KEY** column(s) and the **CONSTRAINT** column(s) have **matching** data types
- Can reference the same table named in the **CREATE TABLE** statement

### FOREIGN KEY Constraint Defined At The Column Level

**Syntax:**

```
<ColumnName> <DataType>(<Size>)
    REFERENCES <TableName> [(<ColumnName>)]
        [ON DELETE CASCADE]
```

**Example 3:**
Drop the table EMP_MSTR, if it already exists. Create a table EMP_MSTR with its primary as EMP_NO referencing the foreign key BRANCH_NO in the BRANCH_MSTR table.

```
DROP TABLE EMP_MSTR;
CREATE TABLE "DBA_BANKSYS"."EMP_MSTR"(
    "EMP_NO" VARCHAR2(10) PRIMARY KEY,
    "BRANCH_NO" VARCHAR2(10) REFERENCES BRANCH_MSTR,
    "FNAME" VARCHAR2(25), "MNAME" VARCHAR2(25),
    "LNAME" VARCHAR2(25), "DEPT" VARCHAR2(30),
    "DESIG" VARCHAR2(30));
```

**Output:**
```
Table created.
```

The **REFERENCES** key word points to the table **BRANCH_MSTR**. The table **BRANCH_MSTR** has the column **BRANCH_NO** as its primary key column. Since no column is specified in the foreign key definition, Oracle applies an automatic (default) link to the primary key column i.e. **BRANCH_NO** of the table **BRANCH_MSTR**.

The foreign key definition is specified as

```
"BRANCH_NO" VARCHAR2(10) REFERENCES BRANCH_MSTR
```

## *FOREIGN KEY Constraint Defined At The Table Level*

**Syntax:**

```
FOREIGN KEY (<ColumnName> [,<ColumnName>] )
    REFERENCES <TableName> [(<ColumnName>,<ColumnName>)
```

**Example 4:**
Drop the table ACCT_FD_CUST_DTLS, if it already exists. Create a table ACCT_FD_CUST_DTLS with CUST_NO as foreign key referencing column CUST_NO in the CUST_MSTR table

```
DROP TABLE ACCT_FD_CUST_DTLS;
CREATE TABLE "DBA_BANKSYS"."ACCT_FD_CUST_DTLS"(
    "ACCT_FD_NO" VARCHAR2(10), "CUST_NO" VARCHAR2(10),
    FOREIGN KEY (CUST_NO) REFERENCES CUST_MSTR(CUST_NO));
```

**Output:**
```
Table created.
```

## *FOREIGN KEY Constraint Defined With ON DELETE CASCADE*

**Example 5:**
Drop the table FD_MSTR, if it already exists. Create a table FD_MSTR with its primary key as FD_SER_NO.

Drop the table FD_DTLS, if it already exists. Create a table FD_DTLS with its foreign key as FD_SER_NO with the ON DELETE CASCADE option. The foreign key is FD_SER_NO and is available as a primary key column named FD_SER_NO in the FD_MSTR table.

Insert some records into both the tables.

```
DROP TABLE FD_MSTR;
CREATE TABLE "DBA_BANKSYS"."FD_MSTR"(
        "FD_SER_NO" VARCHAR2(10) PRIMARY KEY,
        "SF_NO" VARCHAR2(10), "BRANCH_NO" VARCHAR2(10),
        "INTRO_CUST_NO" VARCHAR2(10), "INTRO_ACCT_NO" VARCHAR2(10),
        "INTRO_SIGN" VARCHAR2(1), "ACCT_NO" VARCHAR2(10),
        "TITLE" VARCHAR2(30), "CORP_CUST_NO" VARCHAR2(10),
        "CORP_CNST_TYPE" VARCHAR(4), "VERI_EMP_NO" VARCHAR2(10),
        "VERI_SIGN" VARCHAR2(1), "MANAGER_SIGN" VARCHAR2(1));
```

**Output:**
```
Table created.
```

```
DROP TABLE FD_DTLS;
CREATE TABLE "DBA_BANKSYS"."FD_DTLS"(
    "FD_SER_NO" VARCHAR2(10), "FD_NO" VARCHAR2(10),
    "TYPE" VARCHAR2(1), "PAYTO_ACCTNO" VARCHAR2(10),
    "PERIOD" NUMBER(5), "OPNDT" DATE,
    "DUEDT" DATE, "AMT" NUMBER(8,2),
    "DUEAMT" NUMBER(8,2), "INTRATE" NUMBER(3),
    "STATUS" VARCHAR2(1) DEFAULT 'A', "AUTO_RENEWAL" VARCHAR2(1) ,
    CONSTRAINT f_FDSerNoKey
        FOREIGN KEY (FD_SER_NO) REFERENCES FD_MSTR(FD_SER_NO)
        ON DELETE CASCADE);
```

**Output:**

```
Table created.
```

Now delete a record from the FD_MSTR table as:

```
DELETE FROM FD_MSTR WHERE FD_SER_NO = 'FS1';
```

**Output:**

```
1 row deleted.
```

Query the table FD_DTLS for records:

```
SELECT * FROM FD_DTLS;
```

Notice the deletion of the records belonging to **FS1**.

**Explanation:**
In this example, a primary key is created in the FD_MSTR table. It consists of only **one field** i.e. FD_SER_NO field. Then a foreign key is created in the FD_DTLS table that references the FD_MSTR table based on the contents of the FD_SER_NO field.

Because of the **cascade delete**, when a record in the FD_MSTR table is deleted, all records in the FD_DTLS table will also be deleted that have the same FD_SER_NO value.

## ***FOREIGN KEY Constraint Defined With ON DELETE SET NULL:***

A FOREIGN key with a **SET NULL ON DELETE** means that if a record in the parent table is deleted, then the corresponding records in the child table will have the foreign key fields set to **null**. The records in the child table **will not** be deleted.

A FOREIGN key with a **SET NULL ON DELETE** can be defined in either a CREATE TABLE statement or an ALTER TABLE statement.

**Example 6:**
Drop the table FD_MSTR, if it already exists. Create a table FD_MSTR with its primary key as FD_SER_NO.

Drop the table FD_DTLS, if it already exists. Create a table FD_DTLS with its foreign key as FD_SER_NO with the ON DELETE SET NULL option. The foreign key is FD_SER_NO and is available as a primary key column named FD_SER_NO in the FD_MSTR table.

Insert some records into both the tables.

```
DROP TABLE FD_MSTR;
```

```
CREATE TABLE "DBA_BANKSYS"."FD_MSTR"(
        "FD_SER_NO" VARCHAR2(10) PRIMARY KEY, "SF_NO" VARCHAR2(10),
        "BRANCH_NO" VARCHAR2(10), "INTRO_CUST_NO" VARCHAR2(10),
        "INTRO_ACCT_NO" VARCHAR2(10), "INTRO_SIGN" VARCHAR2(1),
        "ACCT_NO" VARCHAR2(10), "TITLE" VARCHAR2(30), "CORP_CUST_NO" VARCHAR2(10),
        "CORP_CNST_TYPE" VARCHAR(4), "VERI_EMP_NO" VARCHAR2(10),
        "VERI_SIGN" VARCHAR2(1), "MANAGER_SIGN" VARCHAR2(1));
```

**Output:**

```
Table created.
```

```
DROP TABLE FD_DTLS;
CREATE TABLE "DBA_BANKSYS"."FD_DTLS"(
    "FD_SER_NO" VARCHAR2(10), "FD_NO" VARCHAR2(10), "TYPE" VARCHAR2(1),
    "PAYTO_ACCTNO" VARCHAR2(10), "PERIOD" NUMBER(5), "OPNDT" DATE,
    "DUEDT" DATE, "AMT" NUMBER(8,2), "DUEAMT" NUMBER(8,2), "INTRATE" NUMBER(3),
    "STATUS" VARCHAR2(1) DEFAULT 'A', "AUTO_RENEWAL" VARCHAR2(1) ,
    CONSTRAINT f_FDSerNoKey
        FOREIGN KEY (FD_SER_NO) REFERENCES FD_MSTR(FD_SER_NO)
        ON DELETE SET NULL);
```

**Output:**

```
Table created.
```

Now delete a record from the FD_MSTR table as:

```
DELETE FROM FD_MSTR WHERE FD_SER_NO = 'FS1';
```

**Output:**

```
1 row deleted.
```

Query the table FD_DTLS for records:

```
SELECT * FROM FD_DTLS;
```

Notice the value held by the field FD_SER_NO.

**Explanation:**
In this example, a primary key is created in the FD_MSTR table. It consists of only one field i.e. FD_SER_NO field. Then a foreign key is created in the FD_DTLS table that references the FD_MSTR table based on the FD_SER_NO field.

Because of the **cascade set null**, when a record in the FD_MSTR table is deleted, all corresponding records in the FD_DTLS table will have the FD_SER_NO values set to **null**.

## Assigning User Defined Names To Constraints

When constraints are defined, Oracle assigns a **unique name** to each constraint. The convention used by Oracle is

SYS_C**n**

where **n** is a **numeric value** that makes the constraint name **unique**.

Constraints can be given a unique user-defined name along with the constraint definition. A constraint can then, be dropped by referring to the constraint by its name. Under these circumstances a user-defined constraint name becomes very convenient.

User named constraints simplifies the task of dropping constraints. A constraint can be given a user-defined name by preceding the constraint definition with the reserved word **CONSTRAINT** and a **user-defined name**.

**Syntax:**

```
CONSTRAINT <Constraint Name> <Constraint Definition>
```

**Example 7:**
Drop the CUST_MSTR table, if it already exists. Create a table CUST_MSTR with a primary key constraint on the column CUST_NO and also define its constraint name.

```
DROP TABLE CUST_MSTR;
CREATE TABLE CUST_MSTR (
    "CUST_NO" VARCHAR2(10) CONSTRAINT p_CUSTKey PRIMARY KEY,
    "FNAME" VARCHAR2(25), "MNAME" VARCHAR2(25),
    "LNAME" VARCHAR2(25), "DOB_INC" DATE,
    "OCCUP" VARCHAR2(25), "PHOTOGRAPH" VARCHAR2(25),
    "SIGNATURE"  VARCHAR2(25), "PANCOPY" VARCHAR2(1),
    "FORM60" VARCHAR2(1));
```

**Output:**

```
Table created.
```

**Example 8:**
Drop the table EMP_MSTR, if it already exists. Create a table EMP_MSTR with its foreign key as BRANCH_NO. The foreign key is BRANCH_NO available and as a primary key in the BRANCH_MSTR table. Also define the name of the foreign key.

```
DROP TABLE EMP_MSTR;
CREATE TABLE "DBA_BANKSYS"."EMP_MSTR"(
    "EMP_NO" VARCHAR2(10), "BRANCH_NO" VARCHAR2(10),
    "FNAME" VARCHAR2(25), "MNAME" VARCHAR2(25),
    "LNAME" VARCHAR2(25), "DEPT" VARCHAR2(30),
    "DESIG" VARCHAR2(30),
    CONSTRAINT f_BranchKey
        FOREIGN KEY (BRANCH_NO) REFERENCES BRANCH_MSTR);
```

**Output:**

```
Table created.
```

## The Unique Key Constraint

The **Unique** column constraint **permits multiple entries** of NULL into the column. These NULL values are clubbed at the top of the column in the order in which they were entered into the table. This is **the essential difference** between the Primary Key and the Unique constraints when applied to table column(s).

Key point about Unique Constraint:

1. Unique key will not allow duplicate values
2. Unique index is created automatically
3. A table can have more than one Unique key which is not possible in Primary key
4. Unique key can combine upto 16 columns in a Composite Unique key
5. Unique key can not be LONG or LONG RAW data type

## *UNIQUE Constraint Defined At The Column Level*

**Syntax:**

**<ColumnName> <Datatype>(<Size>) UNIQUE**

**Example 9:**
Drop the CUST_MSTR table, if it already exists. Create a table CUST_MSTR such that the contents of the column CUST_NO are unique across the entire column.

```
DROP TABLE CUST_MSTR;
CREATE TABLE CUST_MSTR (
    "CUST_NO" VARCHAR2(10) UNIQUE, "FNAME" VARCHAR2(25), "MNAME" VARCHAR2(25),
    "LNAME" VARCHAR2(25), "DOB_INC" DATE, "OCCUP" VARCHAR2(25),
    "PHOTOGRAPH" VARCHAR2(25), "SIGNATURE" VARCHAR2(25), "PANCOPY" VARCHAR2(1),
    "FORM60" VARCHAR2(1));
```

**Output:**

```
Table created.
```

For testing the Unique constraint execute the following INSERT INTO statements:

```
INSERT INTO CUST_MSTR (CUST_NO, FNAME, MNAME, LNAME, DOB_INC, OCCUP, PHOTOGRAPH,
                       SIGNATURE, PANCOPY, FORM60)
    VALUES('C1', 'Ivan', 'Nelson', 'Bayross', '25-JUN-1952', 'Self Employed', 'D:/ClntPht/C1.gif',
           'D:/ClntSgnt/C1.gif', 'Y', 'Y');
INSERT INTO CUST_MSTR (CUST_NO, FNAME, MNAME, LNAME, DOB_INC, OCCUP, PHOTOGRAPH,
                       SIGNATURE, PANCOPY, FORM60)
    VALUES('C1', 'Chriselle', 'Ivan', 'Bayross', '29-OCT-1982', 'Service', 'D:/ClntPht/C2.gif', 'D:/ClntSgnt/C2.gif', 'N',
           'Y');
INSERT INTO CUST_MSTR (CUST_NO, FNAME, MNAME, LNAME, DOB_INC, OCCUP, PHOTOGRAPH,
                       SIGNATURE, PANCOPY, FORM60)
    VALUES('C2', 'Mamta', 'Arvind', 'Muzumdar', '28-AUG-1975', 'Service', 'D:/ClntPht/C3.gif', 'D:/ClntSgnt/C3.gif',
           'Y', 'Y');
```

**Output:**
The first INSERT INTO statement will execute without any errors as show below:

```
1 row created.
```

When the second INSERT INTO statement is executed an errors occurs as show below:

```
INSERT INTO CUST_MSTR (CUST_NO, FNAME, MNAME, LNAME, DOB_INC, OCCUP,
PHOTOGRAPH, SIGNATURE, PANCOPY,
*
ERROR at line 1:
ORA-00001: unique constraint (DBA_BANKSYS.SYS_C003007) violated
```

The third INSERT INTO statement rectifies this and the result is as show below:

```
1 row created.
```

When a SELECT statement is executed on the Client_Master table the records reterived are:

**SELECT** CUST_NO, FNAME, MNAME, LNAME **FROM** CUST_MSTR;

**Output:**

```
CUST_NO    FNAME    MNAME     LNAME
-----------------------------------
C1         Ivan     Nelson    Bayross
C2         Mamta    Arvind    Muzumdar
```

### *UNIQUE Constraint Defined At The Table Level*

**Syntax:**

```
CREATE TABLE TableName
    (<ColumnName1> <Datatype>(<Size>), <ColumnName2> <Datatype>(<Size>),
        UNIQUE (<ColumnName1>, <ColumnName2>));
```

**Example 10:**
Drop the CUST_MSTR table, if it already exists. Create a table CUST_MSTR such that the contents of the column CUST_NO are unique across the entire column.

```
DROP TABLE CUST_MSTR;
CREATE TABLE CUST_MSTR (
    "CUST_NO" VARCHAR2(10), "FNAME" VARCHAR2(25),
    "MNAME" VARCHAR2(25), "LNAME" VARCHAR2(25),
    "DOB_INC" DATE, "OCCUP" VARCHAR2(25),
    "PHOTOGRAPH" VARCHAR2(25), "SIGNATURE"  VARCHAR2(25),
    "PANCOPY" VARCHAR2(1), "FORM60" VARCHAR2(1),
    UNIQUE(CUST_NO));
```

**Output:**

```
Table created.
```

In the case of the table level unique constraints, the result for the following INSERT INTO statement will remain the same as explained earlier.

```
INSERT INTO CUST_MSTR (CUST_NO, FNAME, MNAME, LNAME, DOB_INC, OCCUP, PHOTOGRAPH,
                       SIGNATURE, PANCOPY, FORM60)
    VALUES('C1', 'Ivan', 'Nelson', 'Bayross', '25-JUN-1952', 'Self Employed', 'D:/ClntPht/C1.gif',
           D:/ClntSgnt/C1.gif', 'Y', 'Y');
INSERT INTO CUST_MSTR (CUST_NO, FNAME, MNAME, LNAME, DOB_INC, OCCUP, PHOTOGRAPH,
                       SIGNATURE, PANCOPY, FORM60)
    VALUES('C1', 'Chriselle', 'Ivan', 'Bayross', '29-OCT-1982', 'Service', 'D:/ClntPht/C2.gif', 'D:/ClntSgnt/C2.gif', 'N',
           'Y');
INSERT INTO CUST_MSTR (CUST_NO, FNAME, MNAME, LNAME, DOB_INC, OCCUP, PHOTOGRAPH,
                       SIGNATURE, PANCOPY, FORM60)
    VALUES('C2', 'Mamta', 'Arvind', 'Muzumdar', '28-AUG-1975', 'Service', 'D:/ClntPht/C3.gif', 'D:/ClntSgnt/C3.gif',
           'Y', 'Y');
```

## Business Rule Constraints

Oracle allows the application of **business rules** to table columns. Business managers determine business rules, they vary from system to system as mentioned earlier. These rules are applied to data, **prior** the data is being inserted into table columns. This ensures that the data (**records**) in the table have integrity.

For example, the rule that no employee in the company shall get a salary less than Rs.1000/- is a business rule. This means that no cell in the **salary** column of the employee table should hold a **value** less than 1000. If an attempt is made, to insert a value less than 1000 into the salary column, the database engine rejects the entire record automatically.

Business rules can be implemented in Oracle by using **CHECK** constraints. Check Constraints can be bound to a **column** or a **table** using the **CREATE TABLE** or **ALTER TABLE** command.

Business rule validation checks are performed when any table **write** operation is carried out. Any insert or update statement causes the relevant Check constraint to be evaluated. The Check constraint must be satisfied for the write operation to succeed. Thus **Check constraints** ensure the integrity of the data in tables.

Conceptually, data constraints are connected to a column, by the Oracle engine, as **flags**. Whenever, an attempt is made to load the column with data, the Oracle engine observes the flag and recognizes the presence of a constraint. The Oracle engine then retrieves the Check constraint definition and then applies the Check constraint definition, to the data being loaded into the table column. If the data being entered into a column fails any of the data constraint checks, the **entire** record is rejected. The Oracle engine will then flash an appropriate **error message**.

Oracle allows programmers to define constraints at:

- Column Level
- Table Level

## Column Level Constraints

If data constraints are defined as an attribute of a column definition when creating or altering a table structure, they are **column level constraints**.

**Caution**


Column level constraints are applied to the **current column**. The current column is the column that immediately **precedes** the constraint (**i.e.** they are local to a specific column). A column level constraint **cannot** be applied if the data constraint spans **across multiple columns** in a table.

## Table Level Constraints

If data constraints are defined **after defining all table column attributes** when creating or altering a table structure, it is a **table level constraint**.

**Note**

A table level constraint **must** be applied if the data constraint **spans across multiple columns** in a table.

Constraints are stored as a part of the global table definition by the Oracle engine in its **system tables**. The SQL syntax used to attach the constraint will change depending upon whether it is a column level or table level constraint.

## NULL Value Concepts

Often there may be records in a table that do not have values for every field. This could be because the information is not available at the time of data entry or because the field is not applicable in every case. If the column was created as **NULLABLE**, Oracle will place a NULL value in the column in the **absence** of a user-defined value.

A NULL value is **different from** a blank or a zero. A **NULL value** can be inserted into **columns** of **any data type**.

## 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** if the data type is **number** and is not equivalent to **spaces** if the data type is **character**
- A NULL value will evaluate to NULL in any expression (**e.g.** NULL multiplied by 10 is NULL)
- NULL value can be inserted into columns of **any data type**
- If the column has a NULL value, Oracle ignores any UNIQUE, FOREIGN KEY, CHECK constraints that may be attached to the column

## Difference Between An Empty String And A NULL Value

Oracle has changed its rules about empty strings and null values in newer versions of Oracle. Now, an empty string is treated as a null value in Oracle.

To understand this go through the following example:

**Example 11:**
Drop the BRANCH_MSTR table, if it already exists. Create a table BRANCH_MSTR such that the contents of the column CUST_NO are unique across the entire column.

```
DROP TABLE BRANCH_MSTR;
CREATE TABLE BRANCH_MSTR (
    "BRANCH_NO" VARCHAR2(10), "NAME" VARCHAR2(25));
```

**Output:**
```
Table created.
```

Next, insert two records into this table.

```
INSERT INTO BRANCH_MSTR (BRANCH_NO, NAME) VALUES('B1', null);
```

**Output:**
```
1 row created.
```

```
INSERT INTO BRANCH_MSTR (BRANCH_NO, NAME) VALUES('B2', '');
```

**Output:**
```
1 row created.
```

The first statement inserts a record with a branch name that is null, while the second statement inserts a record with an empty string as a branch name.

Now, retrieve all rows with a branch name that is an empty string value as follows:

```
SELECT * FROM BRANCH_MSTR WHERE NAME = '';
```

When this statement is executed, it is expected to retrieve the row that was inserted above. But instead, this statement will not retrieve any records at all.

Now, try retrieving all rows where the branch name contains a null value:

```
SELECT * FROM BRANCH_MSTR WHERE NAME IS NULL;
```

When this statement is executed, both rows are retrieved. This is because Oracle has now changed its rules so that empty strings behave as null values.

It is also important to note that the null value is unique. Usual operands such as =, <, > and so on cannot be used on a null value. Instead, the IS NULL and IS NOT NULL conditions have to be used.

## NOT NULL Constraint Defined At The Column Level

In addition to Primary key and Foreign Key, Oracle has **NOT NULL** as column constraint. The **NOT NULL** column constraint ensures that a table column cannot be left empty.

When a column is defined as **not null**, then that column becomes a **mandatory** column. It implies that a value must be entered into the column if the record is to be accepted for storage in the table.

**Syntax:**

**<ColumnName> <Datatype>(<Size>) NOT NULL**

**Example 12:**
Drop the table CUST_MSTR, if already exists and then create it again making Date of Birth field not null. Refer to the details of table in chapter 6

```
CREATE TABLE "DBA_BANKSYS"."CUST_MSTR"(
    "CUST_NO" VARCHAR2(10), "FNAME" VARCHAR2(25),
    "MNAME" VARCHAR2(25), "LNAME" VARCHAR2(25),
    "DOB_INC" DATE NOT NULL, "OCCUP" VARCHAR2(25),
    "PHOTOGRAPH" VARCHAR2(25), "SIGNATURE"  VARCHAR2(25),
    "PANCOPY" VARCHAR2(1), "FORM60" VARCHAR2(1));
```

**Output:**

```
Table created.
```

**Note**

The **NOT NULL** constraint can only be applied at column level.

Execute the following INSERT INTO statements to verify whether mandatory field constraints are applied:

```
INSERT INTO CUST_MSTR (CUST_NO, FNAME, MNAME, LNAME, DOB_INC, OCCUP, PHOTOGRAPH,
                       SIGNATURE, PANCOPY, FORM60)
    VALUES('O14', null, null, null, null, 'Retail Business', null, null, 'N', 'Y');
```

**Output:**

```
INSERT INTO CUST_MSTR (CUST_NO, FNAME, MNAME, LNAME, DOB_INC, OCCUP,
PHOTOGRAPH, SIGNATURE, PANCOPY,
*
ERROR at line 1:
ORA-01400: cannot insert NULL into ("DBA_BANKSYS"."CUST_MSTR"."DOB_INC")
```

The above **error** message confirms that the mandatory field constraints are applied successfully.

***Caution***


The **NOT NULL** constraint can only be applied at column level. Although **NOT NULL** can be applied as a **CHECK** constraint, Oracle Corp recommends that this should **not be done**.

## The CHECK Constraint

Business Rule validations can be applied to a table column by using **CHECK** constraint. **CHECK** constraints must be specified as a logical expression that evaluates either to **TRUE** or **FALSE**.

**Note**

A **CHECK** constraint takes substantially longer to execute as compared to NOT NULL, PRIMARY KEY, FOREIGN KEY or UNIQUE. Thus CHECK constraints must be avoided if the constraint can be defined using the Not Null, Primary key or Foreign key constraint.

### *CHECK constraint defined at the column level:*

**Syntax:**

**<ColumnName> <Datatype>(<Size>) CHECK (<Logical Expression>)**

**Example 13:**
Drop the table CUST_MSTR, if already exists. Create a table **CUST_MSTR** with the following check constraints:

- Data values being inserted into the column **CUST_NO** must start with the capital letter **C**
- Data values being inserted into the column **FNAME, MNAME and LNAME** should be in **upper case** only

```
CREATE TABLE CUST_MSTR("CUST_NO" VARCHAR2(10) CHECK(CUST_NO LIKE 'C%'),
    "FNAME" VARCHAR2(25) CHECK (FNAME = UPPER(FNAME)),
    "MNAME" VARCHAR2(25) CHECK (MNAME = UPPER(MNAME)),
    "LNAME" VARCHAR2(25) CHECK (LNAME = UPPER(LNAME)), "DOB_INC" DATE,
    "OCCUP" VARCHAR2(25), "PHOTOGRAPH" VARCHAR2(25), "SIGNATURE"  VARCHAR2(25),
    "PANCOPY" VARCHAR2(1), "FORM60" VARCHAR2(1));
```

**Output:**

```
Table created.
```

### *CHECK Constraint Defined At The Table Level:*

**Syntax:**

**CHECK (<Logical Expression>)**

**Example 14:**
Drop the table CUST_MSTR, if already exists. Create a table **CUST_MSTR** with the following check constraints:

- Data values being inserted into the column **CUST_NO** must start with the capital letter **C**
- Data values being inserted into the column **FNAME, MNAME and LNAME** should be in **upper case** only

```
CREATE TABLE CUST_MSTR("CUST_NO" VARCHAR2(10), "FNAME" VARCHAR2(25),
    "MNAME" VARCHAR2(25), "LNAME" VARCHAR2(25), "DOB_INC" DATE,
    "OCCUP" VARCHAR2(25), "PHOTOGRAPH" VARCHAR2(25), "SIGNATURE"  VARCHAR2(25),
    "PANCOPY" VARCHAR2(1), "FORM60" VARCHAR2(1),
    CHECK (CUST_NO LIKE 'C%'), CHECK (FNAME = UPPER(FNAME)),
    CHECK (MNAME = UPPER(MNAME)), CHECK (LNAME = UPPER(LNAME)));
```

**Output:**

```
Table created.
```

Execute the following INSERT INTO statements to verify whether check constraints are applied:

```
INSERT INTO CUST_MSTR (CUST_NO, FNAME, MNAME, LNAME, DOB_INC, OCCUP, PHOTOGRAPH,
                       SIGNATURE, PANCOPY, FORM60)
    VALUES('O14', 'SHARANAM', 'CHAITANYA', 'SHAH', '03-Jan-1981', 'Business', null, null, 'N', 'Y');
```

**Output:**

```
INSERT INTO CUST_MSTR (CUST_NO, FNAME, MNAME, LNAME, DOB_INC, OCCUP,
PHOTOGRAPH, SIGNATURE, PANCOPY,
*
ERROR at line 1:
ORA-02290: check constraint (DBA_BANKSYS.SYS_C003055) violated
```

```
INSERT INTO CUST_MSTR (CUST_NO, FNAME, MNAME, LNAME, DOB_INC, OCCUP, PHOTOGRAPH,
                       SIGNATURE, PANCOPY, FORM60)
    VALUES('C14', 'sharanam', 'CHAITANYA', 'SHAH', '03-Jan-1981', 'Business', null, null, 'N', 'Y');
```

**Output:**

```
INSERT INTO CUST_MSTR (CUST_NO, FNAME, MNAME, LNAME, DOB_INC, OCCUP,
PHOTOGRAPH, SIGNATURE, PANCOPY,
*
ERROR at line 1:
ORA-02290: check constraint (DBA_BANKSYS.SYS_C003056) violated
```

```
INSERT INTO CUST_MSTR (CUST_NO, FNAME, MNAME, LNAME, DOB_INC, OCCUP, PHOTOGRAPH,
                       SIGNATURE, PANCOPY, FORM60)
    VALUES('C14', 'SHARANAM', 'Chaitanya', 'SHAH', '03-Jan-1981', 'Business', null, null, 'N', 'Y');
```

**Output:**

```
INSERT INTO CUST_MSTR (CUST_NO, FNAME, MNAME, LNAME, DOB_INC, OCCUP,
PHOTOGRAPH, SIGNATURE, PANCOPY,
*
ERROR at line 1:
ORA-02290: check constraint (DBA_BANKSYS.SYS_C003057) violated
```

```
INSERT INTO CUST_MSTR (CUST_NO, FNAME, MNAME, LNAME, DOB_INC, OCCUP, PHOTOGRAPH,
                       SIGNATURE, PANCOPY, FORM60)
    VALUES('C14', 'SHARANAM', 'CHAITANYA', 'sHAh', '03-Jan-1981', 'Business', null, null, 'N', 'Y');
```

**Output:**

```
INSERT INTO CUST_MSTR (CUST_NO, FNAME, MNAME, LNAME, DOB_INC, OCCUP,
PHOTOGRAPH, SIGNATURE, PANCOPY,
*
ERROR at line 1:
ORA-02290: check constraint (DBA_BANKSYS.SYS_C003058) violated
```

The above **error** messages confirm that the check constraints are applied successfully.

When using **CHECK** constraints, consider the ANSI / ISO standard, which states that a CHECK constraint is violated only if the condition evaluates to **False**. A check constraint is not violated if the condition evaluates to **True**.

**Note**

If the expression in a check constraint does not return a **true / false**, the value is **Indeterminate** or **Unknown.** Unknown values do not violate a check constraint condition. For example, consider the following CHECK constraint for SellPrice column in the Product_Master table:

**CHECK (SellPrice > 0 )**

At first glance, this rule may be interpreted as "do not allow a row in the **Product_Master** table unless **the Sellprice** is greater than 0". However, note that if a row is inserted with a **null SellPrice**, the row **does not violate** the CHECK constraint because the entire check condition is evaluated as **unknown**.

In this particular case, prevent such violations by placing the **not null** integrity constraint along with the check constraint on **SellPrice** column of the table **Product_Master**.

### *Restrictions On CHECK Constraints*

A **CHECK** integrity constraint requires that a condition be **true** or **unknown** for the row to be processed. If an SQL statement causes the condition to evaluate to **false**, an appropriate error message is displayed and processing stops.

A **CHECK** constraint has the following limitations:

- The condition must be a **Boolean** expression that can be evaluated using the values in the row being inserted or updated.
- The condition cannot contain **subqueries** or **sequences**.
- The condition cannot include the SYSDATE, UID, USER or USERENV SQL functions.

## DEFINING DIFFERENT CONSTRAINTS ON A TABLE

**Example 15:**

Drop the table FD_MSTR, if already exists. Create FD_MSTR table where

- The FD_SER_NO is a primary key to this table
- The BRANCH_NO is the foreign key referencing the table BRANCH_MSTR
- The CORP_CUST_NO is the foreign key referencing the table CUST_MSTR
- The VERI_EMP_NO is a foreign key referencing the table EMP_MSTR
- The CORP_CNST_TYPE will hold wither of the following values:
  ØS, 1C, 2C, 3C, 4C, 5C, 6C, 7C indicating different types of companies

```
CREATE TABLE "DBA_BANKSYS"."FD_MSTR"("FD_SER_NO" VARCHAR2(10),
    "SF_NO" VARCHAR2(10), "BRANCH_NO" VARCHAR2(10), "INTRO_CUST_NO" VARCHAR2(10),
    "INTRO_ACCT_NO" VARCHAR2(10), "INTRO_SIGN" VARCHAR2(1),
    "ACCT_NO" VARCHAR2(10), "TITLE" VARCHAR2(30), "CORP_CUST_NO" VARCHAR2(10),
    "CORP_CNST_TYPE" VARCHAR(4), "VERI_EMP_NO" VARCHAR2(10),
    "VERI_SIGN" VARCHAR2(1), "MANAGER_SIGN" VARCHAR2(1),
    CONSTRAINT PK PRIMARY KEY (FD_SER_NO, CORP_CUST_NO),
    CONSTRAINT FK_BR FOREIGN KEY (BRANCH_NO) REFERENCES BRANCH_MSTR,
    CONSTRAINT FK_CU FOREIGN KEY (CORP_CUST_NO) REFERENCES CUST_MSTR,
    CONSTRAINT FK_EM FOREIGN KEY (VERI_EMP_NO) REFERENCES EMP_MSTR,
    CONSTRAINT CHK CHECK (CORP_CNST_TYPE IN ('ØS', '1C', '2C', '3C', '4C', '5C', '6C', '7C')));
```

**Output:**

```
Table created.
```

## THE USER_CONSTRAINTS TABLE

A table can be created with multiple constraints attached to its columns. If a user wishes to see the table structure along with its constraints, Oracle provides the **DESCRIBE <TableName>** command.

This command displays only the column names, data type, size and the NOT NULL constraint. The information about the other constraints that may be attached to the table columns such as the PRIMARY KEY, FOREIGN KEY, and so on, is not available using the DESCRIBE verb.

Oracle stores such information in a table called **USER_CONSTRAINTS**. Querying **USER_CONSTRAINTS** provides information bound to the names of all the constraints on the table. **USER_CONSTRAINTS** comprises of multiple columns, some of which are described below:

**USER_CONSTRAINTS Table:**

| Column Name | Description |
|---|---|
| OWNER | The owner of the constraint. |
| CONSTRAINT_NAME | The name of the constraint |
| TABLE_NAME | The name of the table associated with the constraint |
| CONSTRAINT_TYPE | The type of constraint:<br>**P**: Primary Key Constraint<br>**R**: Foreign Key Constraint<br>**U**: Unique Constraint<br>**C**: Check Constraint |
| SEARCH_CONDITION | The search condition used (for CHECK Constraints) |
| R_OWNER | The owner of the table referenced by the FOREIGN KEY constraints |
| R_CONSTRAINT_NAME | The name of the constraint referenced by a FOREIGN KEY constraint. |

**Example 16:**
View the constraints of the table CUST_MSTR

```
SELECT OWNER, CONSTRAINT_NAME, CONSTRAINT_TYPE FROM USER_CONSTRAINTS
    WHERE TABLE_NAME = 'CUST_MSTR';
```

**Output:**

```
OWNER          CONSTRAINT_NAME  CONSTRAINT_TYPE
-------------- ---------------- ---------------
DBA_BANKSYS    SYS_C003027      C
DBA_BANKSYS    SYS_C003028      C
DBA_BANKSYS    SYS_C003029      C
DBA_BANKSYS    SYS_C003030      C
```

## DEFINING INTEGRITY CONSTRAINTS VIA THE ALTER TABLE COMMAND

Integrity constraints can be defined using the **constraint** clause, in the **ALTER TABLE** command.

**Note**

Oracle **will not allow** constraints defined using the **ALTER TABLE**, to be applied to the table if data previously placed in the table **violates such constraints**.

If a Primary key constraint was being applied to a table in retrospect and the column has duplicate values in it, the Primary key constraint **will not** be set to that column.

The following examples show the definitions of several integrity constraints:

**Example 17:**
Alter the table EMP_MSTR by adding a primary key on the column EMP_NO.

**ALTER TABLE** EMP_MSTR **ADD PRIMARY KEY** (EMP_NO);

**Output:**
```
Table altered.
```

**Example 18:**
Add FOREIGN KEY constraint on the column **VERI_EMP_NO** belonging to the table **FD_MSTR**, which references the table **EMP_MSTR**. Modify column **MANAGER_SIGN** to include the **NOT NULL** constraint

**ALTER TABLE** FD_MSTR **ADD CONSTRAINT F_EmpKey FOREIGN KEY**(VERI_EMP_NO)
**REFERENCES** EMP_MSTR **MODIFY**(MANAGER_SIGN **NOT NULL**);

**Output:**
```
Table altered.
```

## DROPPING INTEGRITY CONSTRAINTS VIA THE ALTER TABLE COMMAND

Integrity constraint can be dropped if the rule that it enforces is no longer **true** or if the constraint is no longer **needed**. Drop the constraint using the **ALTER TABLE** command with the **DROP** clause. The following examples illustrate the dropping of integrity constraints:

**Example 19:**
Drop the PRIMARY KEY constraint from EMP_MSTR.

**ALTER TABLE** EMP_MSTR **DROP PRIMARY KEY;**

**Output:**
```
Table altered.
```

**Example 20:**
Drop FOREIGN KEY constraint on column VERI_EMP_NO in table FD_MSTR

**ALTER TABLE** FD_MSTR **DROP CONSTRAINT F_EmpKey;**

**Output:**
```
Table altered.
```

**Note**

Dropping UNIQUE and PRIMARY KEY constraints **also drops all** associated indexes.

## DEFAULT VALUE CONCEPTS

At the time of table creation a **default value** can be assigned to a column. When a record is loaded into the table, and the column is left empty, the Oracle engine will automatically load this column with the default value specified. The data type of the default value should match the data type of the column. The **DEFAULT** clause can be used to specify a default value for a column.

**Syntax:**

**<ColumnName> <Datatype>(<Size>) DEFAULT <Value>;**

**Example 21:**
Create ACCT_MSTR table where the column CURBAL is the number and by default it should be zero. The other column STATUS is a varchar2 and by default it should have character A. (Refer to table definitions in the chapter 6)

```
CREATE TABLE "DBA_BANKSYS"."ACCT_MSTR"("ACCT_NO" VARCHAR2(10),
    "SF_NO" VARCHAR2(10), "LF_NO" VARCHAR2(10), "BRANCH_NO" VARCHAR2(10),
    "INTRO_CUST_NO" VARCHAR2(10), "INTRO_ACCT_NO" VARCHAR2(10),
    "INTRO_SIGN" VARCHAR2(1), "TYPE" VARCHAR2(2), "OPR_MODE" VARCHAR2(2),
    "CUR_ACCT_TYPE" VARCHAR2(4), "TITLE" VARCHAR2(30),
    "CORP_CUST_NO" VARCHAR2(10), "APLNDT" DATE, "OPNDT" DATE,
    "VERI_EMP_NO" VARCHAR2(10), "VERI_SIGN" VARCHAR2(1),
    "MANAGER_SIGN" VARCHAR2(1), "CURBAL" NUMBER(8, 2) DEFAULT 0,
    "STATUS" VARCHAR2(1) DEFAULT 'A');
```

**Output:**

```
Table created.
```

**Note**

- The data type of the default value should match the data type of the column
- Character and date values will be specified in single quotes
- If a column level constraint is defined on the column with a default value, the default value clause must precede the constraint definition

Thus the syntax will be:

**<ColumnName> <Datatype>(<Size>) DEFAULT <Value> <constraint definition>**

## SELF REVIEW QUESTIONS

### FILL IN THE BLANKS

1. Business rules, which are enforced on data being stored in a table, are called _____.
2. If the column was created as _____ Oracle will place a NULL value in the column in the absence of a user-defined value.
3. When a column is defined as not null, then that column becomes a _____ column.
4. The _____ constraint can only be applied at column level.
5. A _____ value can be inserted into the columns of any data type.
6. A single column primary key is called a _____ key.
7. The data held across the primary key column must be _____.
8. _____ keys represent relationships between tables.
9. The table in which the foreign key is defined is called a Foreign table or _____ table.

10. The default behavior of the foreign key can be changed by using the _____ option.

11. _____ constraints must be specified as a logical expression that evaluates either to TRUE or FALSE.

12. In a CHECK constraint the condition must be a _____ expression that can be evaluated using the values in the row being inserted or updated.

13. _____ constraints can be defined using the constraint clause, in the ALTER TABLE command.

14. Dropping UNIQUE and PRIMARY KEY constraints also drops all associated _____.

**TRUE OR FALSE**

15. Business rules that have to be applied to data are completely System dependent.

16. Constraints super control the data being entered into a table for temporary storage.

17. A NULL value is equivalent to a value of zero.

18. Setting a NULL value is appropriate when the actual value is unknown.

19. A table cannot contain multiple unique keys.

20. Oracle ignores any UNIQUE, FOREIGN KEY, CHECK constraints on a NULL value.

21. A primary key column in a table is an optional column.

22. Standard business rules do not allow multiple entries for the same product.

23. The master table can be referenced in the foreign key definition by using the clause REFERENCES tablename.columnname when defining the foreign key.

24. A CHECK constraint consists of subqueries and sequences.

25. The USER_CONSTRAINTS command displays only the column names, data type, size and the NOT NULL constraint.

26. Drop the constraint using the DROP TABLE command with the DELETE clause.

27. At the time of table creation a default value can be assigned to a column.

28. If a column level constraint is defined on the column with a default value, the default value clause must precede the constraint definition.

## HANDS ON EXERCISES

**1. Create the tables described below:**

**Table Name:** **CLIENT_MASTER**
**Description:** Used to store client information.

| Column Name | Data Type | Size | Default | Attributes |
|---|---|---|---|---|
| CLIENTNO | Varchar2 | 6 | | Primary Key / first letter must start with 'C' |
| NAME | Varchar2 | 20 | | Not Null |
| ADDRESS1 | Varchar2 | 30 | | |

Details for **CLIENT_MASTER** table continued.

| Column Name | Data Type | Size | Default | Attributes |
|---|---|---|---|---|
| ADDRESS2 | Varchar2 | 30 | | |
| CITY | Varchar2 | 15 | | |
| PINCODE | Number | 8 | | |
| STATE | Varchar2 | 15 | | |
| BALDUE | Number | 10,2 | | |

**Table Name:** **PRODUCT_MASTER**
**Description:** Used to store product information.

| Column Name | Data Type | Size | Default | Attributes |
|---|---|---|---|---|
| PRODUCTNO | Varchar2 | 6 | | Primary Key / first letter must start with 'P' |
| DESCRIPTION | Varchar2 | 15 | | Not Null |
| PROFITPERCENT | Number | 4,2 | | Not Null |
| UNITMEASURE | Varchar2 | 10 | | Not Null |
| QTYONHAND | Number | 8 | | Not Null |
| REORDERLVL | Number | 8 | | Not Null |
| SELLPRICE | Number | 8,2 | | Not Null, Cannot be 0 |
| COSTPRICE | Number | 8,2 | | Not Null, Cannot be 0 |

**Table Name:** **SALESMAN_MASTER**
**Description:** Used to store salesman information working for the company.

| Column Name | Data Type | Size | Default | Attributes |
|---|---|---|---|---|
| SALESMANNO | Varchar2 | 6 | | Primary Key / first letter must start with 'S' |
| SALESMANNAME | Varchar2 | 20 | | Not Null |
| ADDRESS1 | Varchar2 | 30 | | Not Null |
| ADDRESS2 | Varchar2 | 30 | | |
| CITY | Varchar2 | 20 | | |
| PINCODE | Number | 8 | | |
| STATE | Varchar2 | 20 | | |
| SALAMT | Number | 8,2 | | Not Null, Cannot be 0 |
| TGTTOGET | Number | 6,2 | | Not Null, Cannot be 0 |
| YTDSALES | Number | 6,2 | | Not Null |
| REMARKS | Varchar2 | 60 | | |

**Table Name:** **SALES_ORDER**
**Description:** Used to store client's orders.

| Column Name | Data Type | Size | Default | Attributes |
|---|---|---|---|---|
| ORDERNO | Varchar2 | 6 | | Primary Key / first letter must start with 'O' |
| CLIENTNO | Varchar2 | 6 | | Foreign Key references ClientNo of Client_Master table |
| ORDERDATE | Date | | | Not Null |
| DELYADDR | Varchar2 | 25 | | |
| SALESMANNO | Varchar2 | 6 | | Foreign Key references SalesmanNo of Salesman_Master table |
| DELYTYPE | Char | 1 | F | Delivery: part (P) / full (F) |
| BILLYN | Char | 1 | | |
| DELYDATE | Date | | | Cannot be less than Order_Date |
| ORDERSTATUS | Varchar2 | 10 | | Values ('In Process', 'Fulfilled', 'BackOrder', 'Cancelled') |

**Table Name:** SALES_ORDER_DETAILS
**Description:** Used to store client's orders with details of each product ordered.

| Column Name | Data Type | Size | Default | Attributes |
|---|---|---|---|---|
| ORDERNO | Varchar2 | 6 | | Foreign Key references OrderNo of Sales_Order table |
| PRODUCTNO | Varchar2 | 6 | | Foreign Key references ProductNo of Product_Master table |
| QTYORDERED | Number | 8 | | |
| QTYDISP | Number | 8 | | |
| PRODUCTRATE | Number | 10,2 | | |

**2. Insert the following data into their respective tables:**

a) Re-insert the data generated for tables CLIENT_MASTER, PRODUCT_MASTER, and SALESMAN_MASTER. Refer to hands-on exercised for **Chapter 07:Interactive SQL-Part I**.

b) Data for Sales_Order table:

| OrderNo | ClientNo | OrderDate | SalesmanNo | DelyType | BillYN | DelyDate | OrderStatus |
|---|---|---|---|---|---|---|---|
| O19001 | C00001 | 12-June-04 | S00001 | F | N | 20-July-02 | In Process |
| O19002 | C00002 | 25-June-04 | S00002 | P | N | 27-June-02 | Cancelled |
| O46865 | C00003 | 18-Feb-04 | S00003 | F | Y | 20-Feb-02 | Fulfilled |
| O19003 | C00001 | 03-Apr-04 | S00001 | F | Y | 07-Apr-02 | Fulfilled |
| O46866 | C00004 | 20-May-04 | S00002 | P | N | 22-May-02 | Cancelled |
| O19008 | C00005 | 24-May-04 | S00004 | F | N | 26-July-02 | In Process |

c) Data for Sales_Order_Details table:

| OrderNo | ProductNo | QtyOrdered | QtyDisp | ProductRate |
|---|---|---|---|---|
| O19001 | P00001 | 4 | 4 | 525 |
| O19001 | P07965 | 2 | 1 | 8400 |
| O19001 | P07885 | 2 | 1 | 5250 |
| O19002 | P00001 | 10 | 0 | 525 |
| O46865 | P07868 | 3 | 3 | 3150 |
| O46865 | P07885 | 3 | 1 | 5250 |
| O46865 | P00001 | 10 | 10 | 525 |
| O46865 | P0345 | 4 | 4 | 1050 |
| O19003 | P03453 | 2 | 2 | 1050 |
| O19003 | P06734 | 1 | 1 | 12000 |
| O46866 | P07965 | 1 | 0 | 8400 |
| O46866 | P07975 | 1 | 0 | 1050 |
| O19008 | P00001 | 10 | 5 | 525 |
| O19008 | P07975 | 5 | 3 | 1050 |