Introduction to VIEWs

- Introduction to VIEWs
  - A VIEW is a database object that contains no data of its own
  - It is a virtual table whose contents are taken from other tables through the execution of a query
  - The changes in the tables are automatically reflected in the VIEWs
  - A VIEW is created with the CREATE VIEW command
  - A VIEW is queried just like querying a table
  - A user can not distinguish between a table and a VIEW
  - Most of the operations on a table can also be carried out with VIEWs, with some restrictions

A VIEW is a virtual table and is like a window to a table. It does not have any data of its own, but derives the data from the table it is associated with.

The advantages of a VIEW are as follows :

- Any updation of rows in the table will automatically reflect in the VIEWs
- As a VIEW does not store any data, the redundancy problem does not arise
- Critical data in the base table is safeguarded as access to such data can be controlled using VIEWs
- Complicated queries are simplified using VIEWs

Example :

* Create a VIEW for the employees belonging to department 20 :

```
CREATE VIEW dept20 AS

SELECT * FROM emp
WHERE deptno = 20;
```

After creating a VIEW, it can be queried just like querying a table.

```
SELECT * FROM dept20;
```

When the FROM clause contains the name of a VIEW, the query associated with it is executed on the underlying base table(s).

- **Applying column constraints**

- Applying column constraints
  - Constraints are a part of the table definition that limit the values entered into its columns

• NOT NULL	: Prevent a column from permitting NULLs
• UNIQUE	: Ensures uniqueness of the values in a column
• PRIMARY KEY	: Same as UNIQUE, but only one column per table
• CHECK	: Controls the value of a column(s) being inserted
• DEFAULT	: Assigns a default value for the column(s), at the time of insertion when no value is given for that column
• REFERENCES	: Assigns a Foreign Key constraint to maintain "Referential Integrity"

While creating a table, constraints on the values that can be entered into its column(s), can be placed. SQL will reject any value that violates the criteria that were defined. The two basic types of constraints are **column constraints** and **table constraints**. The difference between the two is that column constraints apply only to individual columns, whereas table constraints apply to groups of one or more columns.

The constraints are declared at the time of creating a table with the CREATE TABLE command.

- **NOT NULL**

This constraint is placed immediately after the datatype of a column. Any attempts to put NULL values in that column will be rejected.

Example :

If the columns RATE, ROL and ROQ of an item, in the table ITEMMAST created earlier, were to be modified, so as not to allow NULLs, the CREATE TABLE command would contain the following changes :

```
CREATE TABLE itemmast
      (
            ITNO  NUMBER(4),
            NAME  VARCHAR2(20),
            QOH   NUMBER(5),
            CLASS CHAR(1),
            UOM   CHAR(4),
            ROL   NUMBER(5)  NOT  NULL,
            ROQ   NUMBER(5)  NOT  NULL,
            RATE  NUMBER(8, 2)  NOT  NULL
      );
```

- **UNIQUE**

This constraint ensures that the values entered into a column are unique. This is required in the case of the primary key of a table. This can apply only to columns that have also been declared as NOT NULL because it does not make any sense to allow one row of a table to be NULL and exclude other NULLs as duplicates.

Example :

The following changes will be made in the ITEMMAST table, to declare the ITNO and NAME columns as unique.

```
CREATE TABLE itemmast
      (
            ITNO   NUMBER(4)    NOT  NULL  UNIQUE,
            NAME   VARCHAR2(20)  NOT  NULL  UNIQUE,
            QOH    NUMBER(5),
            CLASS  CHAR(1),
            UOM    CHAR(4),
            ROL    NUMBER(5)  NOT  NULL,
            ROQ    NUMBER(5)  NOT  NULL,
            RATE   NUMBER(8, 2)  NOT  NULL
      );
```

We can also define a group of columns as unique with a UNIQUE table constraint. Declaring a group of columns as unique, differs from declaring the individual columns as unique. In the former case, it is the combination of values of the columns that are unique.

In the above example, if the description of an item can be duplicated, then we can not declare the NAME column as unique. So, the combination of ITNO and NAME need to be declared as unique. The table definition will then be :

```
CREATE TABLE itemmast
      (
            ITNO   NUMBER(4)    NOT  NULL,
            NAME   VARCHAR2(20)  NOT  NULL,
            QOH    NUMBER(5),
            CLASS  CHAR(1),
            UOM    CHAR(4),
            ROL    NUMBER(5)  NOT  NULL,
            ROQ    NUMBER(5)  NOT  NULL,
            RATE   NUMBER(8, 2)  NOT  NULL,
            UNIQUE       (ITNO, NAME)
      );
```

- **PRIMARY KEY**

While normalizing tables, we have discussed primary keys for data structures. It is a column in a table which must contain unique value which can be used to identify each and every row of a table uniquely. Therefore, we declare the primary key of a table with NOT NULL and UNIQUE constraints. However, SQL supports primary keys directly with the **PRIMARY KEY** constraint. Functionally it is the same as the UNIQUE constraint, except that only one PRIMARY KEY can be defined for a given table. PRIMARY KEYs will not allow NULL values.

In a table, the PRIMARY KEY can also apply to multiple columns, forcing a unique combination of values.

Example :

In the ITEMMAST table, the ITNO column can also be declared with a PRIMARY KEY :

```
CREATE TABLE itemmast
        (
                ITNO   NUMBER(4) PRIMARY KEY,
                NAME   VARCHAR2(20) NOT  NULL UNIQUE,
                QOH    NUMBER(5),
                CLASS  CHAR(1),
                UOM    CHAR(4),
                ROL    NUMBER(5)  NOT  NULL,
                ROQ    NUMBER(5)  NOT  NULL,
                RATE   NUMBER(8, 2)  NOT  NULL,
        );
```

If we require to declare the combination of ITNO and NAME as the primary key, then this can be defined as :

```
CREATE TABLE itemmast
        (
                ITNO   NUMBER(4)     NOT  NULL,
                NAME   VARCHAR2(20)  NOT  NULL,
                QOH    NUMBER(5),
                CLASS  CHAR(1),
                UOM    CHAR(4),
                ROL    NUMBER(5)  NOT  NULL,
                ROQ    NUMBER(5)  NOT  NULL,
                RATE   NUMBER(8, 2)  NOT  NULL,
                PRIMARY KEY (ITNO, NAME)
        );
```

- **CHECK**

SQL provides the CHECK constraint, which allows the user to define a condition, that a value entered into the table, has to satisfy, before it can be accepted. The CHECK constraint consists of the keyword CHECK followed by parenthesized conditions. Any attempt to update or insert column values that will make the condition false, will be rejected.

Example :

The following changes will be made in the ITEMMAST table, if the CLASS of an item is to be only 'A', 'B' or 'C' so that no other value for this column, is accepted, besides keeping a check that the ROQ and ROL can not be 0 (zero).

```
CREATE TABLE itemmast
        (
                ITNO   NUMBER(4)    PRIMARY KEY,
                NAME   VARCHAR2(20)   NOT   NULL,
                QOH    NUMBER(5),
                CLASS  CHAR(1)   CHECK   (CLASS IN ('A', 'B', 'C')),
                UOM    CHAR(4),
                ROL    NUMBER(5)   CHECK  (ROL >0),
                ROQ    NUMBER(5)   CHECK  (ROQ >0),
                RATE   NUMBER(8, 2)   NOT   NULL,
        );
```

Furthermore, if the ITEMMAST table was to include the checks stating that the rate of any item under class 'A', should be less than 1000.00; class 'B', should be more than 1000.00, but less than 4500.00 and class 'C', should be greater than 4500.00, besides ROL & ROQ not being '0', the changes in the CREATE TABLE statement will be as follows :

```
CREATE TABLE itemmast
        (
                ITNO   NUMBER(4)    PRIMARY KEY,
                NAME   VARCHAR2(20)   NOT   NULL,
                QOH    NUMBER(5),
                CLASS  CHAR(1) NOT NULL,
                UOM    CHAR(4),
                ROL    NUMBER(5),
                ROQ    NUMBER(5),
                RATE   NUMBER(8, 2)   NOT   NULL,
                CHECK  (((CLASS = 'A'   AND   RATE <1000)
                       OR (CLASS = 'B' AND RATE>1000 AND RATE <4500)
                              OR (CLASS = 'C'   AND   RATE > 4500))
                              AND
                              (ROL >0   AND   ROQ > 0)
                              )
        );
```

Instead of assigning such a complex condition, a VIEW can be defined with the WITH CHECK OPTION clause that has all the conditions in its definition. Users could access the VIEW instead of the table. Another advantage with a VIEW is that any time the constraints can be modified according to the requirements.

- **DEFAULT**

While inserting a row into a table without having values for every column, SQL must insert a default value to fill in the excluded column(s), or the command will be rejected. The most common default value is NULL. This can be used with columns **not** defined with a NOT NULL.

Default value assignments are defined in the CREATE TABLE command in the same way as column constraints. These are actually not constraints, but merely specify what happens if the user does not enter value(s) for the given column(s).

Example :

We need to have 100 as QOH for every item in the ITEMMAST table, in case the user does not enter any value for it :

```
CREATE TABLE itemmast
      (
            ITNO  NUMBER(4)   NOT  NULL  PRIMARY KEY,
            NAME  VARCHAR2(20)  NOT  NULL,
            QOH   NUMBER(5)  DEFAULT 100,
            CLASS CHAR(1) NOT NULL,
            UOM   CHAR(4),
            ROL   NUMBER(5),
            ROQ   NUMBER(5),
            RATE  NUMBER(8, 2)  NOT  NULL,
            CHECK (((CLASS = 'A'  AND  RATE <1000)
                        OR (CLASS = 'B'  AND  RATE  <4500)
                        OR (CLASS = 'C'  AND  RATE > 4500))
                  AND
                        (ROL >0  AND  ROQ > 0)
                        )
      );
```

Assigning a default value of '0' to numeric columns helps in arithmetic computation because the columns having NULL values are ignored with relational operators, and produce incorrect result with arithmetic operators.

- **REFERENCES**

A foreign key is a combination of columns with values based on the primary key values from another table. A foreign key constraint, also known as Referential Integrity Constraint, specifies that the values of the foreign key correspond to actual values of the primary key in the other table.

Example :

* Create an item transaction table with Referential Integrity :

```
CREATE TABLE ittran (
      itno          NUMBER(4) REFERENCES  ITEMMAST(itno),
      trantype      CHAR(1)check (Trantype IN ('I','R')),
      trandate      DATE,
      qty           NUMBER(5) );
```

The table with which the referential integrity is being specified (ITEMMAST), must already exist.