
DBMS/SQL
Lesson 07 Database Objects
Capgemini

Lesson Objectives

To understand the following Database Objects:

- Basic Data Types
- Data Integrity
- Examples of CREATE TABLE
- Examples of ALTER TABLE
- Database Objects
- Index
- View
- Tips and Tricks

Overview

A database is a collection of structures with appropriate authorizations and accesses that are defined.
The structures in the database like tables, indexes, etc. are called as objects in the database.
All objects that belong to the same user are said to be the "schema" for the particular user.
Information about existing objects can be retrieved from dba_/user_/all_objects.

Database Objects:
Following is a list of Database objects:

- Tables
- Views
- Indexes
- Clusters
- Synonyms
- Sequences
- Procedures
- Functions
- Packages
- Triggers .

7.1 Basic Data Types

Basic Data Types

Given below are the basic Data Types:

Datatype	Description
CHAR(n)	Stores fixed length string. Maximum length = 2000 bytes For example: NAME CHAR(15)
VARCHAR2(n)	Stores variable length string. Maximum length = 4000 bytes For example: DESCRIPTION VARCHAR2(100)
LONG(n)	Stores variable length string . Maximum length = 2 GIGA bytes For example: SYNOPSIS LONG(5000)
NUMBER(p,s)	Stores numeric data . Range is 1E-129 to 9.99E125 Max Number of significant digits = 38 For example: SALARY NUMBER(9,2)
DATE	Stores DATE. Range from January 1, 4712 BC to December 31, 9999 AD. Both DATE and TIME are stored. Requires 7 bytes. For example: HIREDATE DATE
RAW(n)	Stores data in binary format such as signature, photograph. Maximum size = 255 bytes
LONG RAW(n)	Same as RAW. Maximum size = 2 Gigabytes

Basic Data Types supported in SQL:

Scalar Data Types:

- The traditional Data Types are called “Scalar Data Types”. The slide discusses some of the Scalar Data Types.

7.1 Basic Data Types

Basic Data Types contd..

Datatype	Description
TIMESTAMP	**Stores the time to be stored as a date with fractional seconds. Extension to the DATA datatype** **There are some variations of the data type**

Basic Data Types supported in SQL:

The TimeStamp data type is introduced in Oracle 9i. This data type provides support for time zones. It is an extension of the DATE datatype.

TIMESTAMP [(fractional_seconds_precision)] – It stores the year, month & day of the date data type. In addition to that it also stores hour, minute, second and fractional second value.

7.1: Database Objects

Table

Tables are objects, which store the user data.
Use the CREATE TABLE statement to create a table, which is the basic structure to hold data.
For example:

```
CREATE TABLE book_master
(book_code number,
  book_name varchar2(50),
  book_pub_year number,
  book_pub_author varchar2(50));
```

Creating tables is done with the create table command. You can add rows to a table with the INSERT statement, after creating a table.
The above create table command does the following:

- Defines the table name
- Defines the columns in the table and the datatypes of those columns

In above example, we create a table called BOOK_MASTER which has 4 columns. The first column and third column is defined as NUMBER datatype. This means we will be storing numbers in this column. The second and fourth columns are of VARCHAR2 datatype. We will be storing text data in these columns
Syntax:

```
CREATE TABLE table_name
(
{col_name.col_datatype [[CONSTRAINT
const_name][col_constraint]]},...
[table_constraint],...
)
[AS query]
```

- If a table is created as shown in the slide, then there is no restriction on the data that can be stored in the table.
- However, if we wish to put some restriction on the data, which can be stored in the table, then we must supply some “constraints” for the columns. We will see the Constraints as next topic.

7.2: Data Integrity

What is Data Integrity?

Data Integrity:

- "Data Integrity" allows to define certain "data quality requirements" that must be met by the data in the database.
- Oracle uses "Integrity Constraints" to prevent invalid data entry into the base tables of the database.
  - You can define "Integrity Constraints" to enforce the business rules you want to associate with the information in a database.
  - If any of the results of a "DML statement" execution violate an "integrity constraint", Oracle rolls back the statement and returns an error.

Data Integrity: Integrity Constraint

- An Integrity Constraint is a declarative method of defining a rule for a column of a table.

Example of Data Integrity:

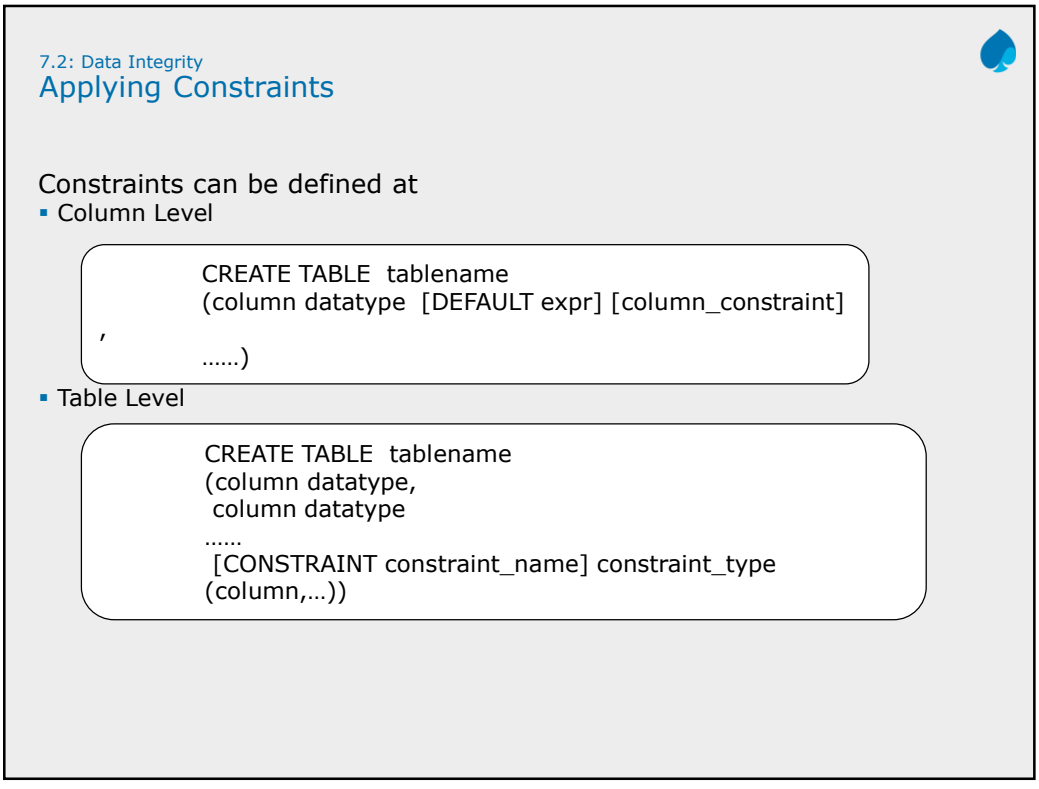
- Assume that you define an "Integrity Constraint" for the Staff_Sal column of the Staff_Master table.
- This Integrity Constraint enforces the rule that "no row in this table can contain a numeric value greater than 10,000 in this column".
- If an INSERT or UPDATE statement attempts to violate this "Integrity Constraint", Oracle rolls back the statement and returns an "information error" message.

7.2: Data Integrity

Advantages

Advantages of Integrity Constraints:

- Integrity Constraints have advantages over other alternatives. They are:
  - Enforcing "business rules" in the code of a database application.
  - Using "stored procedures" to completely control access to data.
  - Enforcing "business rules" with triggered stored database procedures.



Applying Constraints:

In Oracle you can apply constraints

Column Level - References a single column and is defined within a specification for the owning column; can be any type of integrity constraint

Table Level - References one or more columns and is defined separately from the definitions of the columns in the table; can define any constraints except NOT NULL

7.2: Data Integrity

Types of Integrity Constraints

Let us see the types of Data Integrity Constraints:

- Nulls
- Default
- Unique Column Values
- Primary Key Values
- Check
- Referential Integrity

Types of Data Integrity:

- Oracle supports the following Integrity Constraints:
  - NOT NULL constraints for the rules associated with nulls in a column. “Null” is a rule defined on a single column that allows or disallows, inserts or updates on rows containing a “null” value (the absence of a value) in that column.
  - UNIQUE key constraints for the rule associated with unique column values. A “unique value” rule defined on a column (or set of columns) allows inserting or updating a row only if it contains a “unique value” in that column (or set of columns).
  - PRIMARY KEY constraints for the rule associated with primary identification values. A “primary key” value rule defined on a key (a column or set of columns) specifies that “each row in the table can be uniquely identified by the values in the key”.
  - FOREIGN KEY constraints for the rules associated with referential integrity. A “Referential Integrity” rule defined on a key (a column or set of columns) in one table guarantees that “the values in that key, match the values in a key in a related table (the referenced value)”. Oracle currently supports the use of FOREIGN KEY integrity constraints to define the referential integrity actions, including:
    - update and delete No Action
    - delete CASCADE
    - delete SET NULL
  - CHECK constraints for complex integrity rules

7.3: Examples of CREATE TABLE

NOT NULL Constraint

The user will not be allowed to enter null value.

For Example:

- A NULL value is different from a blank or a zero. It is used for a quantity that is "unknown".
- A NULL value can be inserted into a column of any data type.

```
CREATE TABLE  student_master
(student_code  number(4) NOT NULL,
 dept_code   number(4) CONSTRAINT dept_code_nn
                                    NOT  NULL );
```

NOT NULL constraint:

- 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 the data entry or because the field is not applicable in every case.
- If the column is created as NULLABLE, in the absence of a user-defined value the DBMS will place a NULL value in the column.
- A NULL value is different from a blank or a zero. It is used for a quantity that is "unknown".
- "Null" is a rule defined on a single column that allows or disallows, inserts or updates on rows containing a "null" value (the absence of a value) in that column.
- A NULL value can be inserted into a column of any data type.
- **Principles of NULL values:**
  - Setting a NULL value is appropriate when the "actual value" is unknown, or when a value is not meaningful.
  - A NULL value is not equivalent to the value of "zero" if the data type is number, and it is not equivalent to "spaces" if the data type is character.
  - A NULL value will evaluate to NULL in any expression
    For example: 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, and CHECK constraints that may be attached to the column.

7.3: Examples of CREATE TABLE

DEFAULT clause

If no value is given, then instead of using a "Not Null" constraint, it is sometimes useful to specify a default value for an attribute.

For Example:

- When a record is inserted the default value can be considered.

```
CREATE TABLE  staff_master(
Staff_Code number(8) PRIMARY KEY,
Staff_Name varchar2(50) NOT NULL,
Staff_dob date,
Hiredate date DEFAULT sysdate,
…..)
```

7.3: Examples of CREATE TABLE

UNIQUE constraint

The keyword UNIQUE specifies that no two records can have the same attribute value for this column.

For Example:

```
CREATE TABLE student_master
(student_code number(4),
 student_name varchar2(30) ,
 CONSTRAINT stu_id_uk  UNIQUE(student_code )) ;
```

UNIQUE constraint:

- The UNIQUE constraint does not allow duplicate values in a column.
  - If the UNIQUE constraint encompasses two or more columns, then two equal combinations are not allowed.
  - However, if a column is not explicitly defined as NOT NULL, then NULLS can be inserted multiple number of times.
- A UNIQUE constraint can be extended over multiple columns.
- A “unique value” rule defined on a column (or set of columns) allows inserting or updating a row only if it contains a “unique value” in that column (or set of columns).

7.3: Examples of CREATE TABLE

PRIMARY KEY constraint

The Primary Key constraint enables a unique identification of each record in a table.

For Example:

```
CREATE TABLE Staff Master
(staff_code  number(6)
CONSTRAINT staff_id_pk PRIMARY KEY,
 staff_name varchar2(20)
  ……);
```

PRIMARY KEY constraint:

- On a technical level, a PRIMARY KEY combines a UNIQUE constraint and a NOT NULL constraint.
- Additionally, a table can have at the most one PRIMARY KEY.
- After creating a PRIMARY KEY, it can be referenced by a FOREIGN KEY.
- A “primary key” value rule defined on a key (a column or set of columns) specifies that “each row in the table can be uniquely identified by the values in the key”.
- The example on the slide defines the primary key constraint at the column level. The same example is seen below with the constraint defined at table level

```
CREATE TABLE Staff Master
(staff_code  number(6) ,
 staff_name varchar2(20),
  ……
CONSTRAINT staff_id_pk  PRIMARY KEY
(staff_code))
;
```

7.3: Examples of CREATE TABLE

CHECK constraint

CHECK constraint allows users to restrict possible attribute values for a column to admissible ones.

For Example:

```
CREATE TABLE staff_master
( staff_code number(2),
  staff_name  varchar2(20),
  staff_sal   number(10,2) CONSTRAINT staff_sal_min
                           CHECK (staff_sal >1000),
  …..) ;
```

CHECK constraint:

- A CHECK constraint allows to state a minimum requirement for the value in a column.
- If more complicated requirements are desired, an INSERT trigger must be used.

7.3: Examples of CREATE TABLE

FOREIGN KEY constraint

The FOREIGN KEY constraint specifies a "column" or a "list of columns" as a foreign key of the referencing table.
The referencing table is called the "child-table", and the referenced table is called "parent-table".

For Example:

```
CREATE TABLE student_master
(student_code number(6) ,
 dept_code number(4) CONSTRAINT stu_dept_fk
                    REFERENCES
department_master(dept_code),
 student_name varchar2(30) );
```

FOREIGN KEY Constraint Keywords:

- Given below are a few Foreign Key constraint keywords:
  - FOREIGN KEY: Defines the column in the child table at the table constraint level.
  - REFERENCES: Identifies the table and column in the parent table.
  - ON DELETE CASCADE: Deletes the dependent rows in the child table when a row in the parent table is deleted.
  - ON DELETE SET NULL: Converts dependent FOREIGN KEY values to NULL.
- You can query the USER_CONSTRAINTS table to view all constraint definitions and names.
- You can view the columns associated with the constraint names in the USER_CONS_COLUMNS view.

- In EMP table, for deptno column, if we want to allow only those values that already exist in deptno column of the DEPT table, we must enforce what is known as REFERENTIAL INTEGRITY. To enforce REFERENTIAL INTEGRITY, declare deptno field of DEPT table as PRIMARY KEY, and deptno field of EMP table as FOREIGN KEY as follows

7.3: Examples of CREATE TABLE

Create new table based on existing table

Constraints on an "old table" will not be applicable for a "new table".

```
CREATE TABLE student_dept10 AS
SELECT student_code, student_name
FROM student_master WHERE  dept_code = 10
```

Creating new table based on an existing table:

- The example in the slide shows how to create a new table based on an existing table.
  - When the new table will be created, it will contain student information from department 10.
  - Constraints on an old table will not be applicable for the new table except NOT NULL constraint.

7.4: Examples of ALTER TABLE

ALTER Table

Given below is an example of ALTER TABLE:

```
ALTER TABLE table_name
    [ADD (col_name col_datatype col_constraint ,...)]|
    [ADD (table_constraint)]|
    [DROP CONSTRAINT constraint_name]|
    [MODIFY existing_col_name new_col_datatype
                                    new_constraint new_default]
    [DROP COLUMN existing_col_name]
    [SET UNUSED COLUMN existing_col)name];
```

Examples of ALTER TABLE:

- Table_name must be an existing table.
- A column can be removed from an existing table by using ALTER TABLE.
- The uses of modifying columns are:
  - Can increase the width of a character column, any time.
  - Can increase the number of digits in a number, any time.
  - Can increase or decrease the number of decimal places in a number column, any time. Any reduction on precision and scale can be on empty columns only.
  - Can add only NOT NULL constraint by using Column constraints. All other constraints have to be specified as Table constraints.

7.4: Examples of ALTER TABLE

ALTER Table – Add clause

The "Add" keyword is used to add a column or constraint to an existing table.

- For adding three more columns to the emp table, refer the following example:

```
ALTER TABLE Student_Master
ADD (last_name varchar2(25) );
```

ALTER TABLE – Add clause:

- The ADD clause allows to add a column or constraint. You can also add multiple columns in one statement separated by comma.
- A column with constraints can also be added as shown in the following example:

```
ALTER TABLE Department_Master
ADD (dept_name varchar2(10) NOT NULL);
```

7.4: Examples of ALTER TABLE

ALTER Table – Add clause

For adding Referential Integrity on "mgr_code" column, refer the following example:

```
ALTER TABLE staff_master
   ADD CONSTRAINT FK FOREIGN KEY  (mgr_code)
REFERENCES staff_master(staff_code);
```

7.4: Examples of ALTER TABLE

ALTER Table – MODIFY clause

MODIFY clause:

- The "Modify" keyword allows making modification to the existing columns of a table.
  - For Modifying the width of "sal" column, refer the following example:

```
ALTER TABLE staff_master
MODIFY (staff_sal number (12,2)  ) ;
```

ALTER TABLE- Modify Clause

The use of modifying the columns with the Enable | Disable clause are:

- Can increase "column width" of a character any time.
- Can increase the "number of digits" in a number at any time.
- Can increase or decrease the "number of decimal places" in a number column at any time. Any reduction on "precision" and "scale" can only be on empty columns.
- Can only add the NOT NULL constraint by using "column constraints". Rest all other constraints have to be specified as "table constraints".

7.4: Examples of ALTER TABLE

ALTER Table – Enable | Disable clause

ENABLE | DISABLE Clause:

- The ENABLE | DISABLE clause allows constraints to be enabled or disabled according to the user choice without removing them from a table.
- Refer the following example:

```
ALTER TABLE staff_master DISABLE CONSTRAINT
SYS_C000934;
```

7.4: Examples of ALTER TABLE

ALTER Table – DROP clause

The DROP clause is used to remove constraints from a table.

- For Dropping the FOREIGN KEY constraint on "department", refer the following example:

```
ALTER TABLE  student_master
DROP CONSTRAINT  stu_dept_fk ;
```

ALTER TABLE – Drop Clause

- To remove the PRIMARY KEY constraint on the DEPARTMENT table and drop the associated FOREIGN KEY constraint on the DEPT_CODE column.

```
ALTER TABLE department_master
DROP PRIMARY KEY CASCADE;
```

7.4: Examples of ALTER TABLE

Dropping Column

Given below are the ways for "Dropping" a column:

- 1a.Marking the columns as unused and then later dropping them.
- 1b. The following command can be used later to permanently drop the columns.

```
ALTER TABLE staff_master SET UNUSED COLUMN staff_address;
ALTER TABLE staff_master SET UNUSED (staff_sal, hiredate);
```

```
ALTER TABLE emp DROP UNUSED COLUMNS;
```

Ways for "Dropping" a column:

1. **Marking the columns as "Unused" and then later dropping them:**

- Oracle onwards a new feature to "drop" and "set" the "unused columns" in a table is added
  - First command as shown in the slide, allows you to mark a column as unused and
  - Second command as shown in the following slide, lets you drop the unused column from the table to create more free space.
    - This feature drops the column from a table and releases any space back to the segment.
- **Note that:**
  - Columns once marked as unused cannot be recovered.
  - Marking the columns as unused does not release the space occupied by them back to the database.
    - Until you actually drop these columns, they continue to count towards the absolute limit of 1000 columns per table.
    - If you mark a column of data type LONG as UNUSED, you cannot add another LONG column to the table until you actually drop the unused LONG column.
- The advantage of the marking column as "unused" and then dropping them is that marking the columns is much faster process than dropping the columns.
- You can refer to the data dictionary table USER_UNUSED_COL_TABS to get information regarding the tables with columns marked as unused.

7.4: Examples of ALTER TABLE

Dropping Column

- Directly dropping the columns.

```
ALTER TABLE staff_master DROP COLUMN staff_sal;
```

Ways for "Dropping" a column (contd.):

2. DROP COLUMN:

- This feature allows directly dropping the column.
- For DROP COLUMN command shown in the slide, to work successfully, the table should be exclusively locked by the user by giving the command.
  - All "indexes" defined on any of the target columns are also dropped.
  - All "constraints" that reference a target column are removed.
- Note that this command should be used with caution.
- The CASCADE CONSTRAINTS clause is used along with the DROP COLUMN clause.
- The CASCADE CONSTRAINTS clause drops all referential integrity constraints that refer to the primary and unique keys defined on the dropped columns.
- The CASCADE CONSTRAINTS clause also drops all multicolumn constraints defined on the dropped columns

7.5: Database Objects

Drop a Table

The DROP TABLE command is used to remove the definition of a table from the database.

For Example:

```
DROP TABLE staff_master;

DROP TABLE Department_master
    CASCADE CONSTRAINTS;
```

Deleting Database Objects: Tables

- Deleting objects that exist in the database is an easy task. Just say:

```
DROP Obj_Type obj_name;
```

- A table that is dropped cannot be recovered. When a table is dropped, dependent objects such as indexes are automatically dropped. Synonyms and views created on the table remain, but give an error if they are referenced.
- You cannot delete a table that is being referenced by another table. To do so use the following:

```
DROP  TABLE table-name CASCADE CONSTRAINTS;
```

7.5: Database Objects

Rename a Table

The RENAME command is used to give a new name to the table. Views can also be renamed using this command

For Example:

```
RENAME  staff_master TO new_staffmaster;
```

Renaming Database Objects: Tables

- Renaming objects that exist in the database is an easy task. Just say:

```
RENAME Existing_TableName TO new_tablename;
```

- You can RENAME a table that is being referenced by another table.

7.5: Database Objects

Truncating a Table

The TRUNCATE command is used to permanently remove the data from a table, keeping the table structure intact.

For Example:

```
TRUNCATE TABLE staff_master ;
```

Truncating Database Objects: Tables

- Truncating objects that exist in the database is an easy task. Just say:

```
TRUNCATE TABLE  Existing_TableName
```

7.5: Database Objects

User_Tables & User_Objects

To view the names of tables owned by the user, use the following query:
To view distinct object types owned by the user, use the following query:

```
SELECT table_name
FROM user_tables
```

```
SELECT DISTINCT object_type
FROM user_objects ;
```

7.6: Index

Usage of Index

Index is a database object that functions as a “performance-tuning” method for allowing faster retrieval of records.
Index creates an entry for each value that appears in the indexed columns.
The absence or presence of an Index does not require change in wording of any SQL statement.

Index

• An index

- Is a schema object

- Is used by the Oracle server to speed up the retrieval of rows by using a pointer

- Can reduce disk I/O by using a rapid path access method to locate data quickly

- Is independent of the table it indexes

- Is used and maintained automatically by the Oracle Server.

- Oracle uses indexes to avoid the need for large-table, full-table scans and disk sorts, which are required when the SQL optimizer cannot find an efficient way to service the SQL query

7.6: Index

Usage of Index

Syntax:

```
CREATE [UNIQUE] INDEX index_name
ON table_name(col_name1 [ASC|DESC],col_name2,.....)
```

Index:

- An index creates an entry for each value that appears in the indexed columns (by default, Oracle creates B-tree indexes).
- The absence or presence of an Index does not require change in wording of any SQL statement.
  - An Index is merely a fast access path to the data.
  - An Index only affects the speed of execution.
- There are different types of Indexes, which can be created by the user.
- A “Table” can have any number of “Indexes”.
- An Index can be on a single column or on multiple columns.
  - In Oracle, up to 32 columns can be included in one Index.
- Updation of all Indexes is handled by RDBMS.
- UNIQUE ensures that all rows in a Table are unique.
- An Index is in an ascending order, by default.
- The RDBMS decides when to use the Index while accessing the data.
- Removal of an Index does not affect the Table on which it is based.
- When you create a PRIMARY or a UNIQUE constraint on the “Table”, a UNIQUE index is automatically created.
  - The name of the constraint is used for the Index name.

7.6: Index

Creating an Index

Example 1: A simple example of an Index is given below:

```
CREATE INDEX  staff_sal_index ON
staff_master(staff_sal);
```

Example 2: To allow only unique values in the field "ename", the CREATE statement should appear as shown below:

```
CREATE UNIQUE INDEX  staff_ename_unindex
ON staff_master(staff_name );
```

Note:

- More Indexes on a Table does not mean faster queries.
  - Each DML operation that is committed on a Table with Indexes imply that the Indexes must be updated.
  - The more number of Indexes are associated with a Table, the more effort the Oracle server must make to update all the Indexes after a DML operation.

When do you create an Index?

- You should create Indexes, only if:
  - the column contains a wide range of values
  - the column contains a large number of NULL values
  - one or more columns are frequently used together in a WHERE clause or join condition
  - the table is large and most queries are expected to retrieve less than 2–4% of the rows
- If you want to enforce uniqueness, you should define a UNIQUE constraint in the Table definition. Then a "unique index" will be automatically created.
- It is usually not worth creating an Index, if:
  - The Table is small.
  - The columns are not often used as a condition in the query.
  - Most queries are expected to retrieve more than 2 to 4 percent of the rows in the table.
  - The Table is frequently updated.
  - The indexed columns are referenced as part of an expression.

7.6: Index

How are Indexes created?

Indexes can be either created "automatically" or "manually".

- Automatically: A unique Index is automatically created when you define a PRIMARY KEY or UNIQUE constraint in a table definition.
- Manually: A non-unique index can be created on columns by users in order to speed up access to the rows.

Types of Indexes:

- Two types of Indexes can be created.
  - One type of index is an "unique index". The Oracle server automatically creates this index when you define a column in a table that has a PRIMARY KEY or a UNIQUE key constraint. The name of the Index is same as the name given to the constraint.
  - The other type of index is a "non-unique index", which can be created by a user.
    For example: You can create a FOREIGN KEY column index for a "join" in a query in order to improve retrieval speed.
    Note: You can manually create a "unique index". However, it is recommended that you create a "UNIQUE constraint", which implicitly creates a "unique index".

7.7 View

Usage of View

A View can be thought of as a "stored query" or a "virtual table", i.e. a logical table based on one or more tables.

- A View can be used as if it is a table.
- A View does not contain data.

Why do we use Views?

Views are used:

- To restrict data access.
- To make complex queries easy.
- To provide data independence.
- To present different views of the same data.

Features of Simple and Complex Views:

Features	Simple View	Complex View
Number of tables	One	One or more
Contains functions	No	Yes
Contains groups of data	No	Yes
DML operations through a View.	Yes	Not always

7.7 View

Usage of View

Syntax

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view [(alias[,
alias]...)] AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

View:

Creating a View:

- You can embed a sub-query within the CREATE VIEW statement.
- The sub-query can contain complex SELECT syntax.

Characteristics of a View:

- View is a logical table that is based on one or more Tables.
- View can be used as if it is a Table.
- View does not contain data.
- Whenever a View is accessed, the query is evaluated. Thus a View is dynamic.
- Any changes made in the View affects the Tables on which the View is based.
- View helps to hide the following from the user:
  - Ownership details of a Table, and
  - Complexity of the query used to retrieve the data
- "With check option" implies you cannot insert a row in the underlying Table, which cannot be selected by using the View.
- If you use a ORDER BY clause in the View, then it automatically becomes a “Read-only View”.

7.7: View

Creating a View

Given below is an example of a simple View:

```
CREATE VIEW staff_view
    AS
    SELECT * FROM staff_master
    WHERE hiredate >'01-jan-82';
```

Restrictions while using Views:

- Updating / Inserting rows in the base table **is possible** by using Views, however, with some restrictions. This is possible only if the View is based on a single table.
- The restrictions are :
  - Updation / Insertion not possible if View is based on two tables. However, this can be done in ORACLE 8 onwards.
  - Insertion is not allowed if the underlying table has any NOT NULL columns, which are not included in the View.
  - Insertion / Updation is not allowed if any column of the View referenced in UPDATE / INSERT contains “functions” or “calculations”.
  - Insertion / Updation / Deletion is not allowed if View contains GROUP BY or DISTINCT clauses in the query.

7.7 View

Creating a View

Creating a Complex View:

- As shown in the example given below, create a Complex View that contains group functions to display values from two tables.

```
CREATE VIEW dept_sum_vu
  (name, minsal, maxsal, avgsal)
 AS SELECT dept.dept_name, MIN(staff.staff_sal),
           MAX(staff. staff_sal),AVG(staff. staff_sal)
   FROM   staff_master staff, department_master
dept
   WHERE staff.dept_code = dept.dept_code
   GROUP BY dept.dept_name;
```

Modifying a View:

- As shown in the example given below, modify the View “staff_vu” by using the CREATE OR REPLACE VIEW clause. Add an alias for each column name.

```
CREATE OR REPLACE VIEW staff_vu
  (id_number, name, sal, department_id)
AS SELECT staff_code,staff_name,staff_sal,dept_code
   FROM  staff_master
   WHERE   dept_code = 80;
```

- Column aliases in the CREATE VIEW clause are listed in the same order as the columns in the sub-query.

7.7 View

Creating a View

Creating a View with WITH CHECK OPTION:

```
CREATE VIEW staff_vw
AS
SELECT *   FROM  staff_master
WHERE  deptno =10 WITH CHECK OPTION constraint cn;
```

WITH CHECK OPTION Specifies that only the rows accessible to the view can be inserted or updated. Constraint “cn” is the name assigned to the CHECK OPTION constraint in the above example. Any attempt to change the department number for any row in the view fails because it violates the WITH CHECK OPTION constraint. A violation produces:
ORA-01402: view WITH CHECK OPTION where-clause violation
With the given error, the SQL tried to INSERT or UPDATE a record in a view that contained a WITH CHECK OPTION. The resulting INSERT or UPDATE violates the WHERE clause of the view.

Creating a Read-only View:

•If a View is based on a single table, the user may manipulate records in the View, and it’s underlying base table.
•The WITH READ ONLY clause of the CREATE VIEW command can be used to prevent users from manipulating records in a View.

```
CREATE VIEW student_view
    AS
    SELECT * FROM student_master
    WHERE hiredate >'01-jan-82'  WITH  READ ONLY.
```

7.7 View

Rules for performing operation on View

You can perform "DML operations" on simple Views.
You cannot remove a row if the View contains the following:

- Group functions
- A GROUP BY clause
- The DISTINCT keyword
- The pseudocolumn ROWNUM keyword

Rules for performing DML Operations on a View:

- You can perform "DML operations" on data "through a View" only if those operations follow certain rules.
  - You can remove a row from a view, unless it contains any of the following:
    - Group functions
    - A GROUP BY clause
    - The DISTINCT keyword
    - The pseudocolumn ROWNUM keyword

Removing a View:

- You can remove a View by using the following syntax:

```
DROP VIEW viewname;
```

7.9: Tips and Tricks

Guidelines

When creating tables based on subquery the number of specified columns if defined for the table should match to the number of columns in the subquery.

Create an index if

- A column contains a wide range of values
- A column contains a large number of null values
- One or more columns are frequently used together in a WHERE clause or a join condition
- The table is large and most queries are expected to retrieve less than 2 to 4 percent of the rows

7.9: Tips and Tricks

Guidelines

An Index is not very useful if :

- The table is small
- The columns are not often used as a condition in the query
- Most queries are expected to retrieve more than 2 to 4 percent of rows in the table
- The table is updated frequently
- The indexed columns are referenced as part of an expression

Summary

In this lesson, you have learnt:

- What are Database Objects?
- Basic Data Types
- Data Integrity
- Different types of Database Objects:
- Modification of Database Objects

Review – Questions

Question 1: Indexes can be created _____ or _____