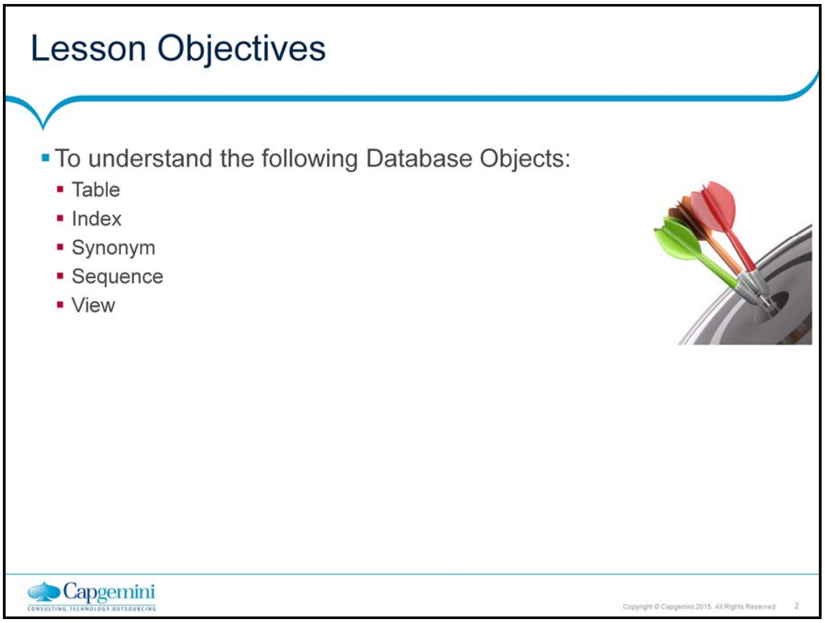
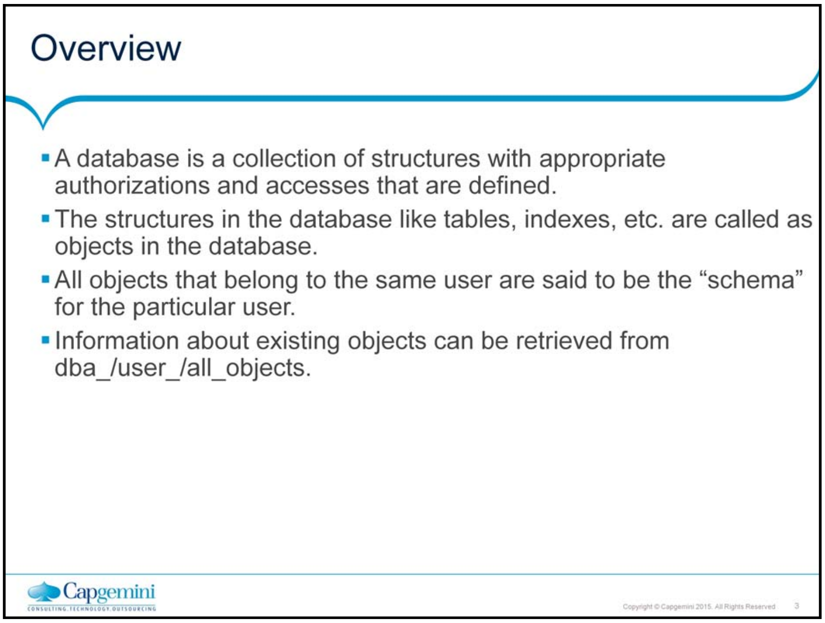
DBMS SQL

Lesson 09 Database Objects



Lesson Objectives

- To understand the following Database Objects:
  - Table
  - Index
  - Synonym
  - Sequence
  - View



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

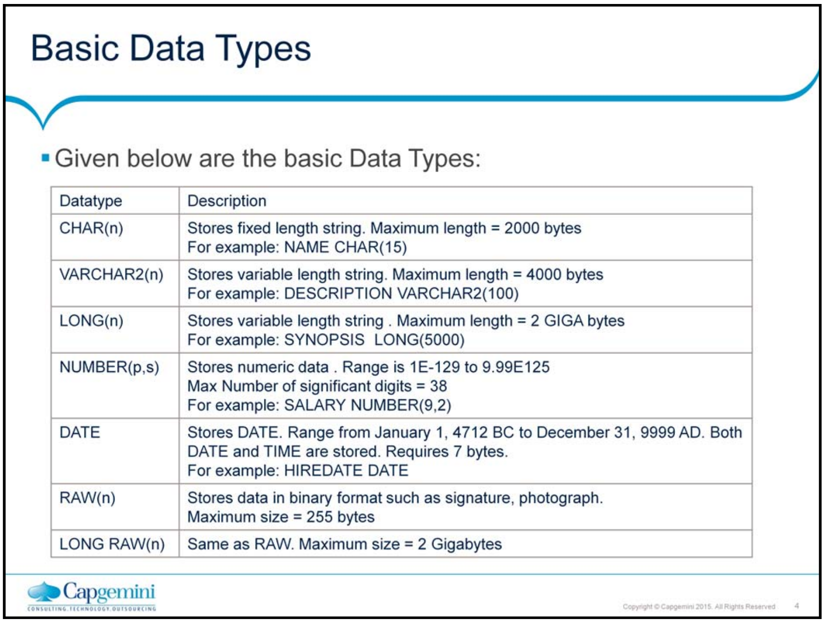
Sequences

Procedures

Functions

Packages

Triggers .



Basic Data Types

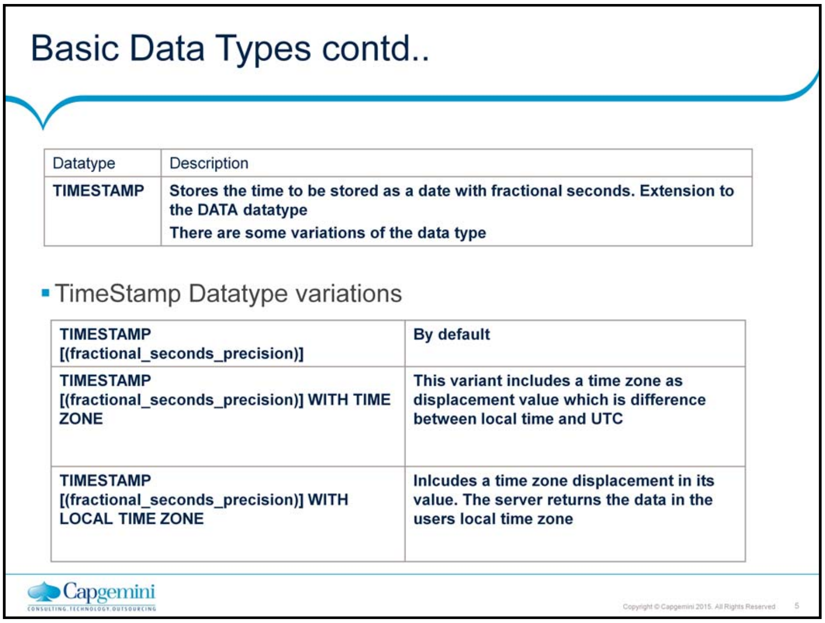
- Given below are the basic Data Types:

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.



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

- TimeStamp Datatype variations

TIMESTAMP [(fractional_seconds_precision)]	By default
TIMESTAMP [(fractional_seconds_precision)] WITH TIME ZONE	This variant includes a time zone as displacement value which is difference between local time and UTC
TIMESTAMP [(fractional_seconds_precision)] WITH LOCAL TIME ZONE	Inlcudes a time zone displacement in its value. The server returns the data in the users local time zone

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.

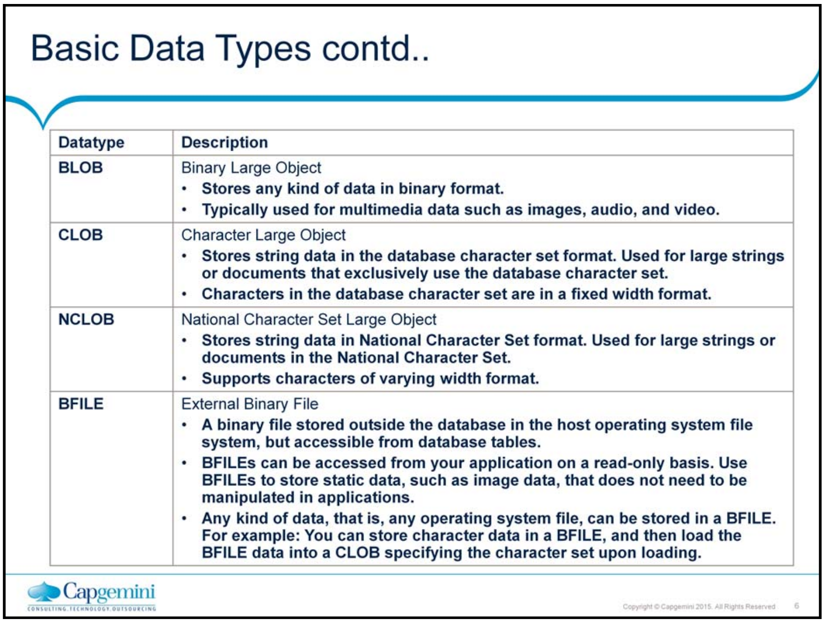
The slide lists the variations supported for TIMESTAMP datatype

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.

TIMESTAMP [(fractional_seconds_precision)] WITH TIME ZONE - This variant includes a time zone as displacement value which is difference between local time and UTC (Universally Coordinated Time)

TIMESTAMP [(fractional_seconds_precision)] WITH LOCAL TIME ZONE – this variant includes time zone displacement in its value. This displacement value is not stored as part of the column data. The server returns the data in users local session time zone

In each variant the fractional_second_precision specifies the number of digits in the fractional part of the second and can be a number in the range of 0 to 9. This is an optional value and the default value is 6.



Basic Data Types contd..

Datatype	Description
BLOB	Binary Large Object • **Stores any kind of data in binary format.** • **Typically used for multimedia data such as images, audio, and video.**
CLOB	Character Large Object • **Stores string data in the database character set format. Used for large strings or documents that exclusively use the database character set.** • **Characters in the database character set are in a fixed width format.**
NCLOB	National Character Set Large Object • **Stores string data in National Character Set format. Used for large strings or documents in the National Character Set.** • **Supports characters of varying width format.**
BFILE	External Binary File • **A binary file stored outside the database in the host operating system file system, but accessible from database tables.** • **BFILEs can be accessed from your application on a read-only basis. Use BFILEs to store static data, such as image data, that does not need to be manipulated in applications.** • **Any kind of data, that is, any operating system file, can be stored in a BFILE. For example: You can store character data in a BFILE, and then load the BFILE data into a CLOB specifying the character set upon loading.**

Basic Data Types supported in SQL:

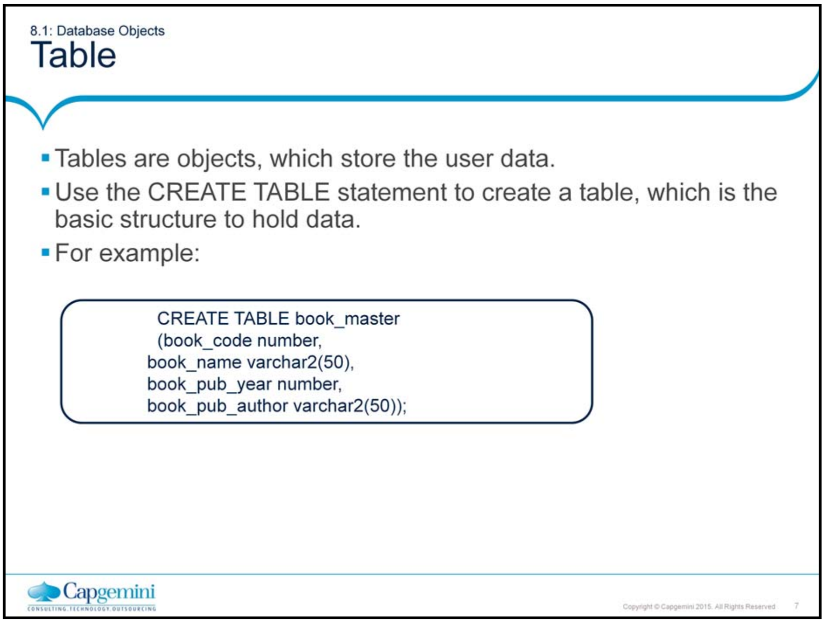
Oracle8i and Oracle9i's support for LOB (Large Object) data types is preferred over Oracle7’s support for LONG and LONG RAWs in the following ways:

LOB Capacity: With Oracle8 and Oracle8i, LOBs can store up to 4GB of data. This doubles the 2GB of data that LONG and LONG RAW data types can store.

Number of LOB columns in a table: An Oracle8, Oracle8i, or Oracle9i table can have multiple LOB columns. Each LOB column in the same table can be of a different type. In Oracle7 Release 7.3 and higher, tables are limited to a single LONG or LONG RAW column.

Random piece-wise access: LOBs support random access to data, but LONGs support only sequential access.

LOBs can also be object attributes.



8.1: 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

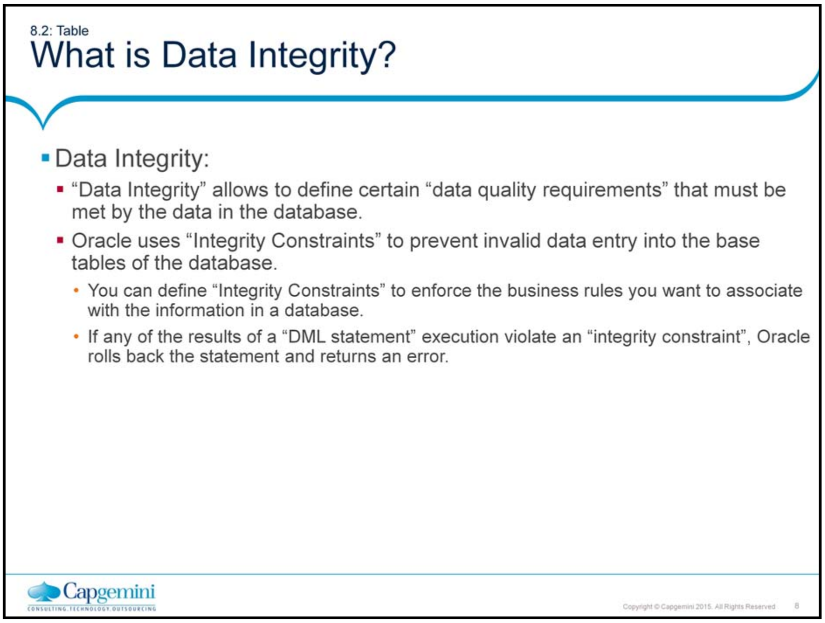
Syntax:

```
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.



8.2: Table

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

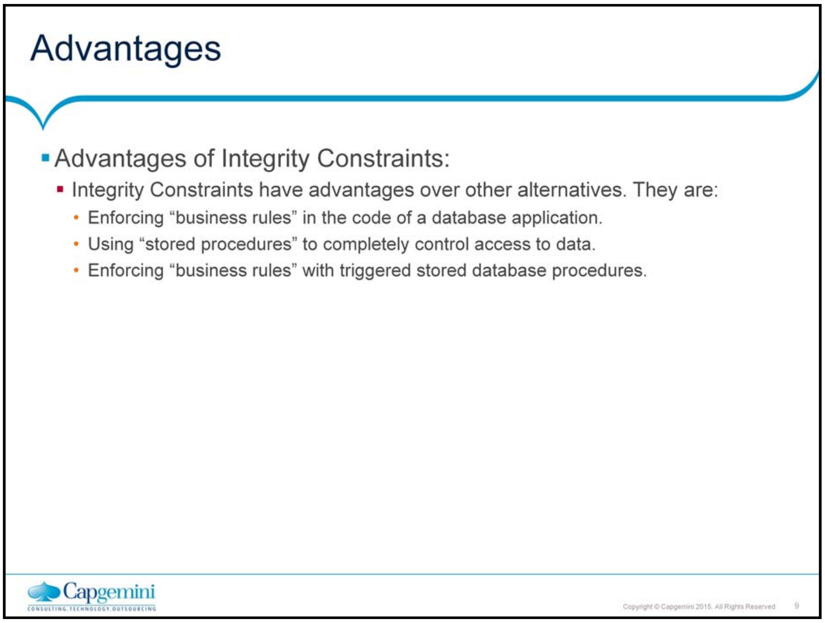
An Integrity Constraint is a declarative method of defining a rule for a column of a table.

Example of Data Integrity:

Assume that you define an "Integrity Constraint" for the Staff_Sal column of the Staff_Master table.

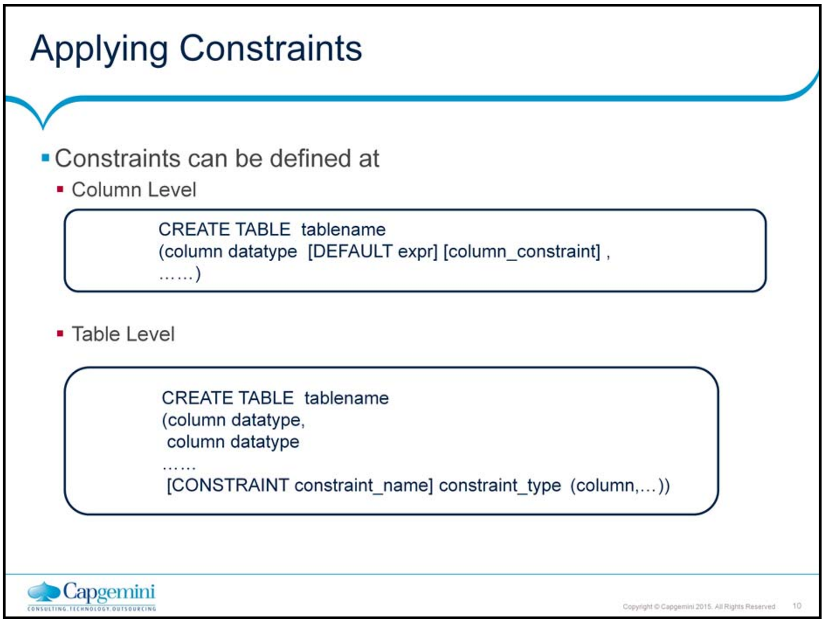
This Integrity Constraint enforces the rule that "no row in this table can contain a numeric value greater than 10,000 in this column".

If an INSERT or UPDATE statement attempts to violate this "Integrity Constraint", Oracle rolls back the statement and returns an "information error" message.



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

- Constraints can be defined at
  - Column Level

```
CREATE TABLE  tablename
(column datatype  [DEFAULT expr] [column_constraint] ,
……)
```

  - Table Level

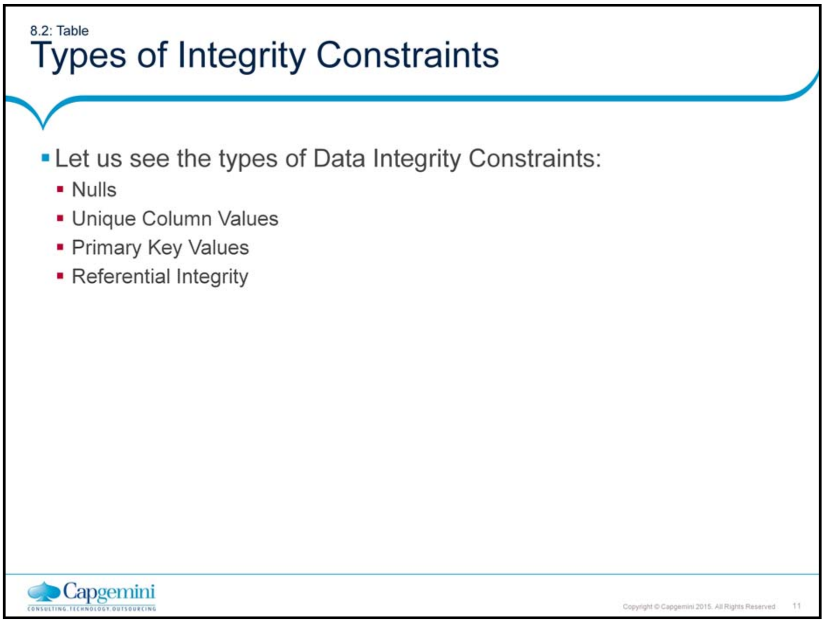
```
CREATE TABLE  tablename
(column datatype,
 column datatype
……
 [CONSTRAINT constraint_name] constraint_type  (column,…))
```

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



8.2: Table

Types of Integrity Constraints

- Let us see the types of Data Integrity Constraints:
  - Nulls
  - Unique Column Values
  - Primary Key Values
  - 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”.

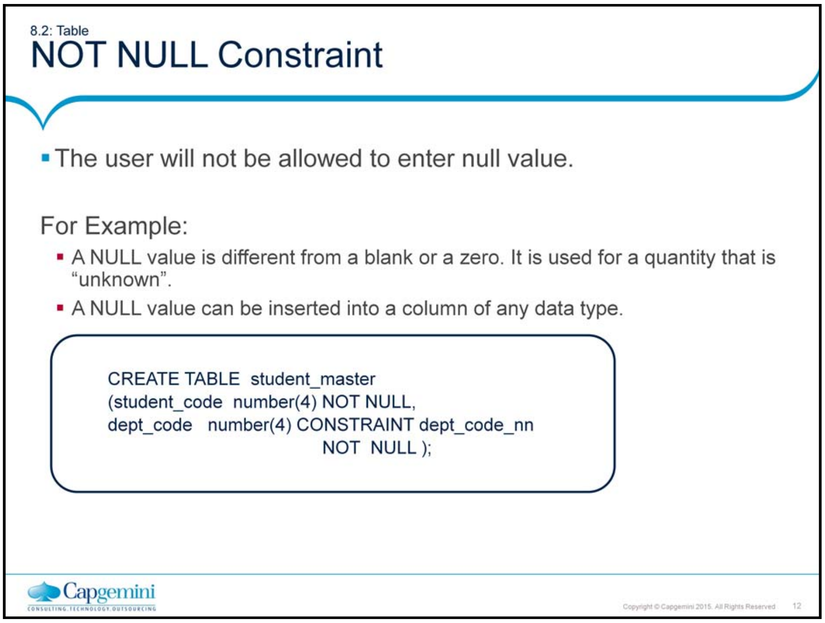
FOREIGN KEY constraints for the rules associated with referential integrity. A “Referential Integrity” rule defined on a key (a column or set of columns) in one table guarantees that “the values in that key, match the values in a key in a related table (the referenced value)”. Oracle currently supports 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



8.2: 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.

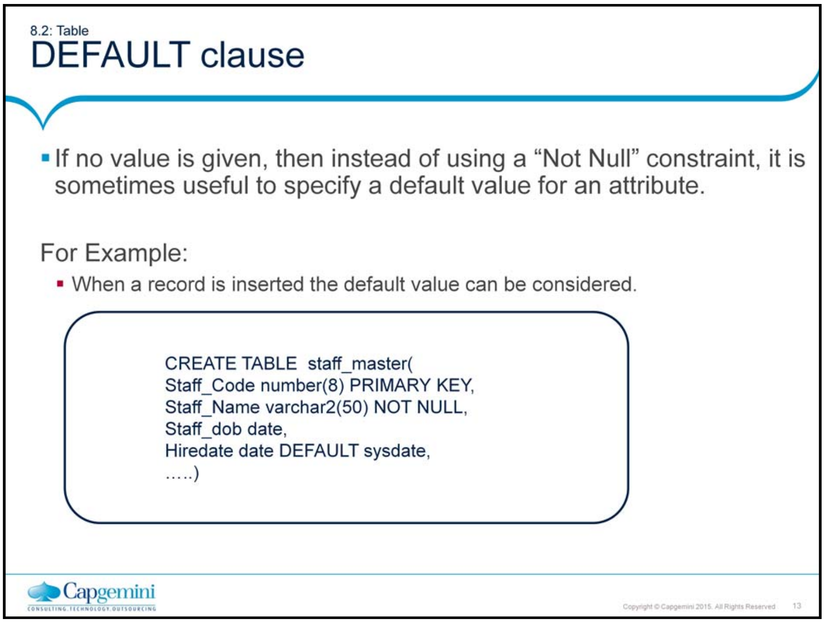
A NULL value is not equivalent to the value of "zero" if the data type is number, and it is not equivalent to "spaces" if the data type is character.

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.



8.2: Table

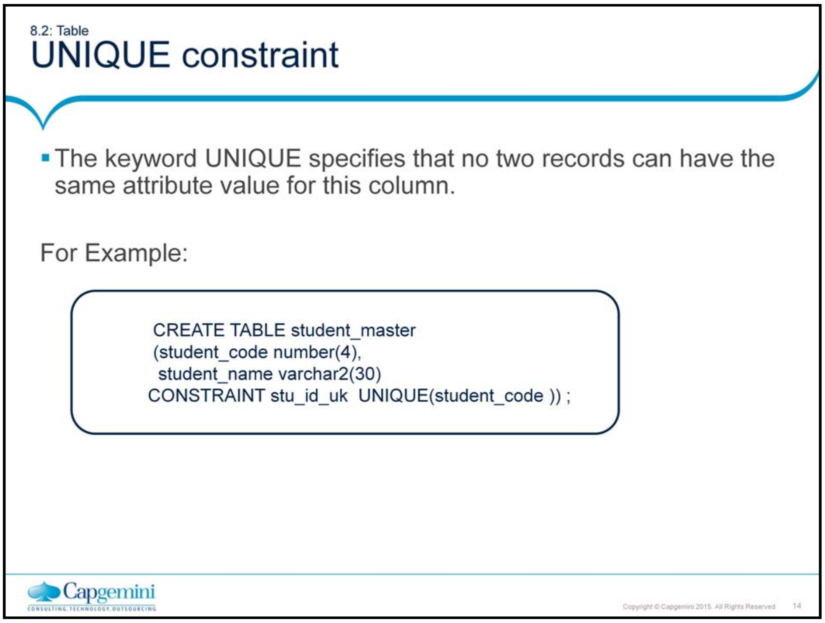
DEFAULT clause

- 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,
…..)
```



8.2: 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 )) ;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

UNIQUE constraint:

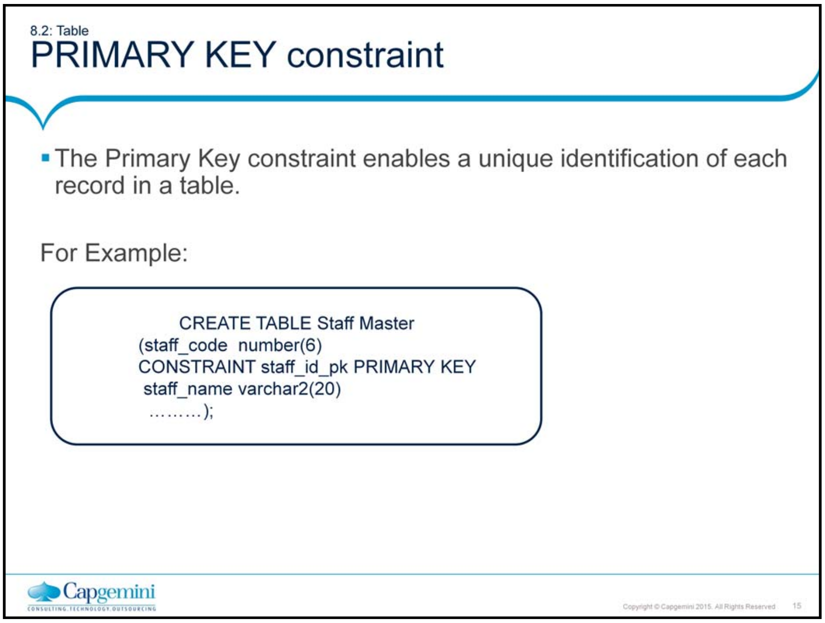
The UNIQUE constraint does not allow duplicate values in a column.

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).



8.2: 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.

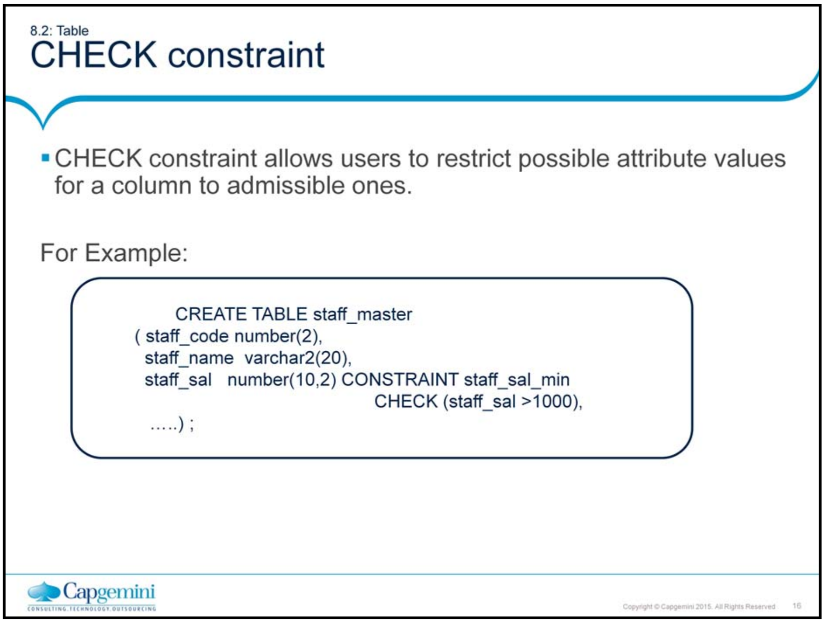
Additionally, a table can have at the most one PRIMARY KEY.

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)
;
```



8.2: Table

CHECK constraint

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

For Example:

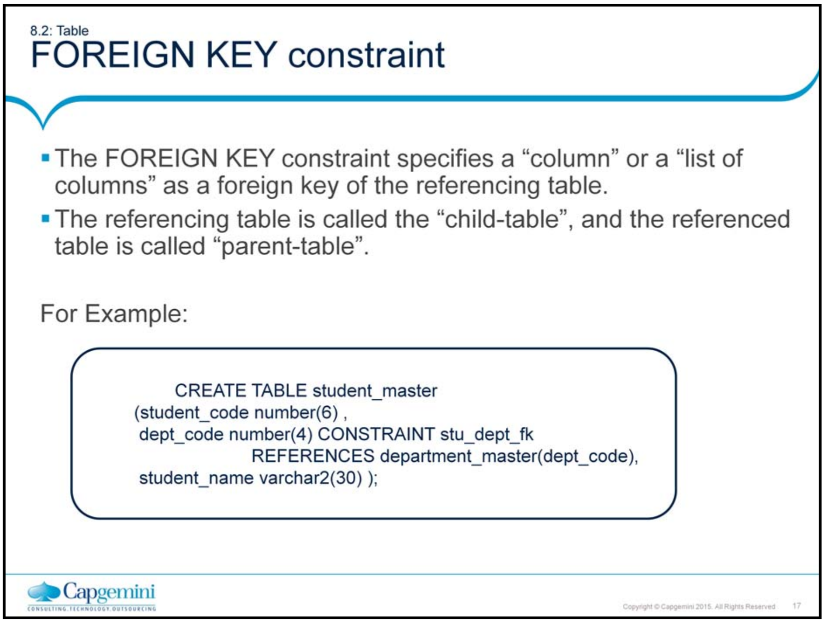
```
        CREATE TABLE staff_master
( staff_code number(2),
  staff_name  varchar2(20),
  staff_sal   number(10,2) CONSTRAINT staff_sal_min
                                  CHECK (staff_sal >1000),
  …..) ;
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

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.



8.2: 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

FOREIGN KEY Constraint Keywords:

In the given example, FOREIGN KEY has been declared as a Table constraint.

```
CREATE TABLE Dept
(
Deptno          NUMBER(2) CONSTRAINT
DEPTNO_P_KEY PRIMARY KEY,
Dname    VARCHAR2(14)      NOT NULL,

Loc              VARCHAR2(13)      NOT NULL

   );
CREATE TABLE Emp
   (
   Empno NUMBER(4) CONSTRAINT P_KEY
PRIMARY KEY,
   Ename VARCHAR2(10)  CONSTRAINT
ENAME_NOT_NULL  NOT NULL,
   Deptno NUMBER(2),
Job CHAR(9) CONSTRAINT
JOB_ALL_UPPER
CHECK (Job =UPPER(Job)),
   Hiredate DATE DEFAULT SYSDATE,
   CONSTRAINT DEPTNO_F_KEY FOREIGN
KEY (Deptno)
    REFERENCES Dept(Deptno)
   );
```

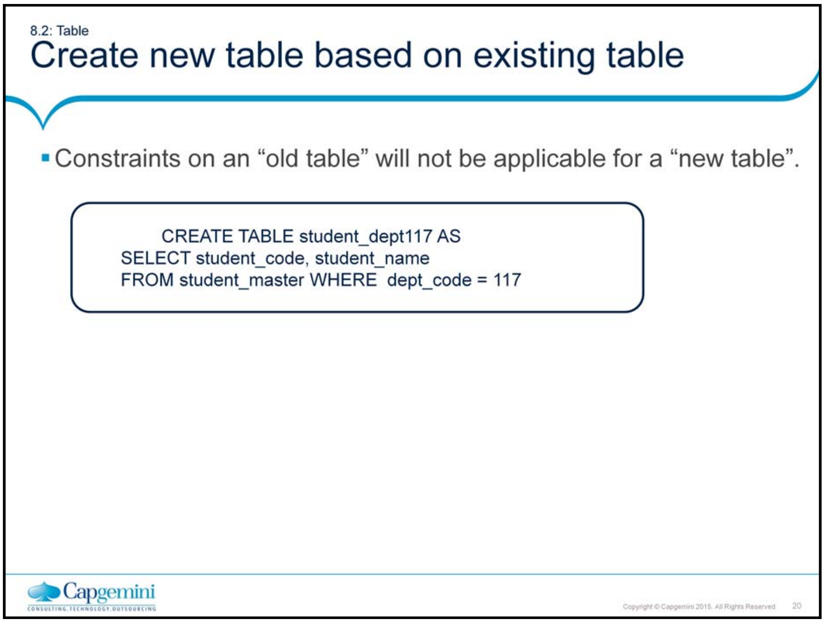
Creation of database objects: Tables (contd.):

A table can have a maximum of 1000 columns. Only one column of type LONG is allowed per table.

Table_name,col_name,const_name	A string upto 30 characters length. Can be made up to A-Z,0-9,$,_,# Must begin with a non-numeric ORACLE data characters.
Col_datatype	One of the previously mentioned types.
Col_constraint	A restriction on the column can be of following types: PRIMARY KEY, NOT NULL, UNIQUE, FOREIGN KEY, CHECK, Can be named. Only one PRIMARY KEY is allowed per table.
Table_constraint	A restriction on single or multiple columns. The types are same as in col_constraint. (NULL constraint is not allowed here).
AS query	Query is an SQL statement using SELECT command. SELECT command returns the rows from tables. It is useful if the table being created is based on an existing table. New table need not have all the columns of the old table. Names of columns can be different. All or part of the data can be copied.

Any object created by a user is accessible to the user and the DBA only.

To make the object accessible to other users, the creator or the DBA must explicitly give permission to others.



8.2: Table

Create new table based on existing table

- Constraints on an “old table” will not be applicable for a “new table”.

```
        CREATE TABLE student_dept117 AS
SELECT student_code, student_name
FROM student_master WHERE  dept_code = 117
```

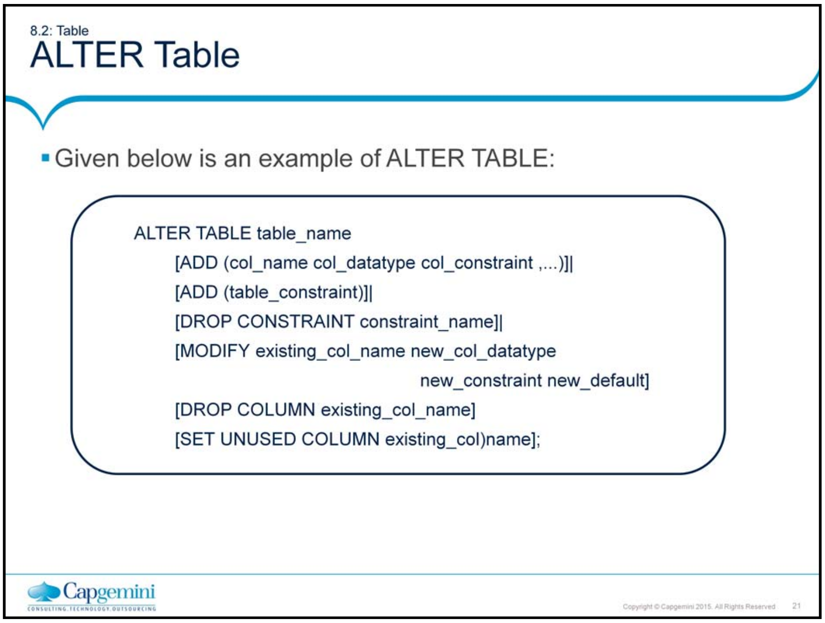
Copyright © Capgemini 2015. All Rights Reserved 20

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 117.

Constraints on an old table will not be applicable for the new table except NOT NULL constraint.



8.2: 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 cannot be removed from an existing table by using ALTER TABLE.

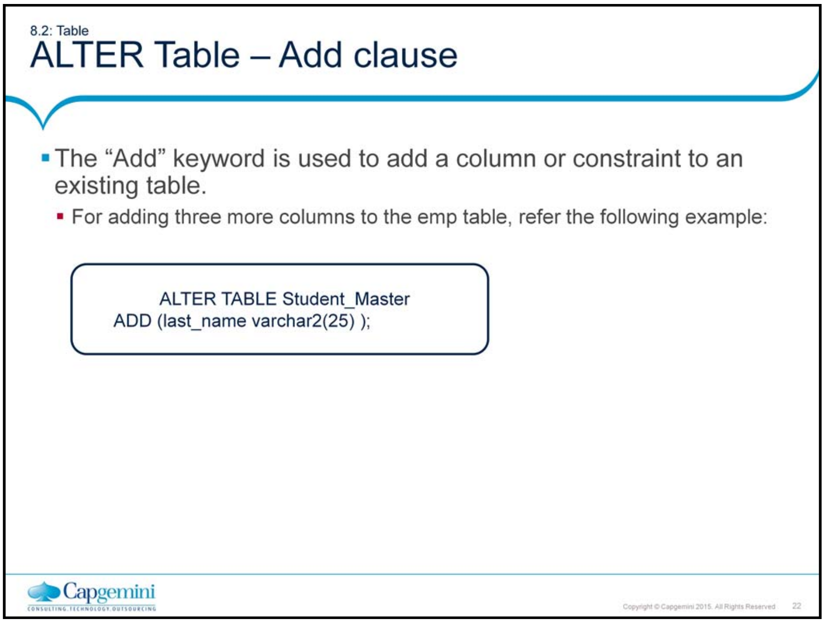
The uses of modifying columns are:

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.



8.2: 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) );
```

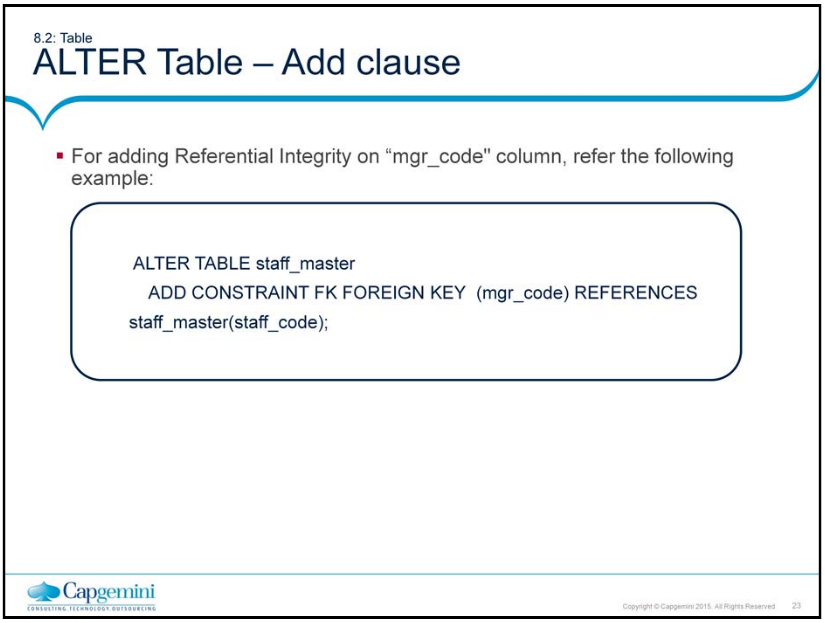
Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

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);
```



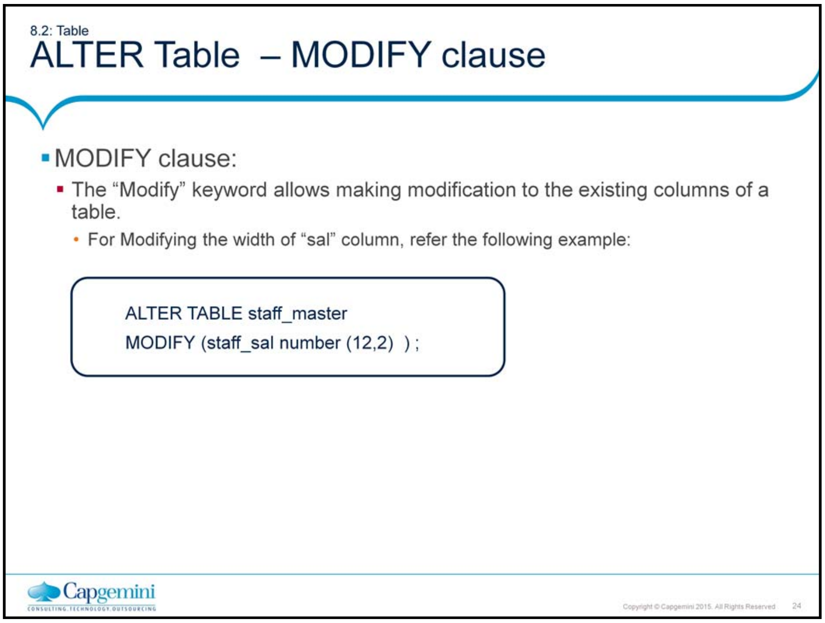
8.2: Table

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);
```

23



8.2: 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)  ) ;
```

24

ALTER TABLE- Modify Clause

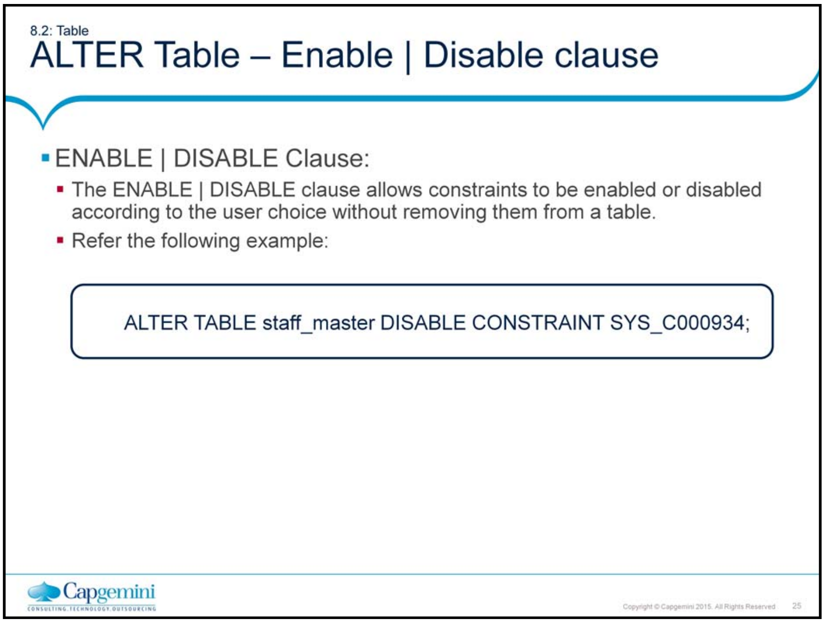
The use of modifying the columns with the Enable | Disable clause are:

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”.

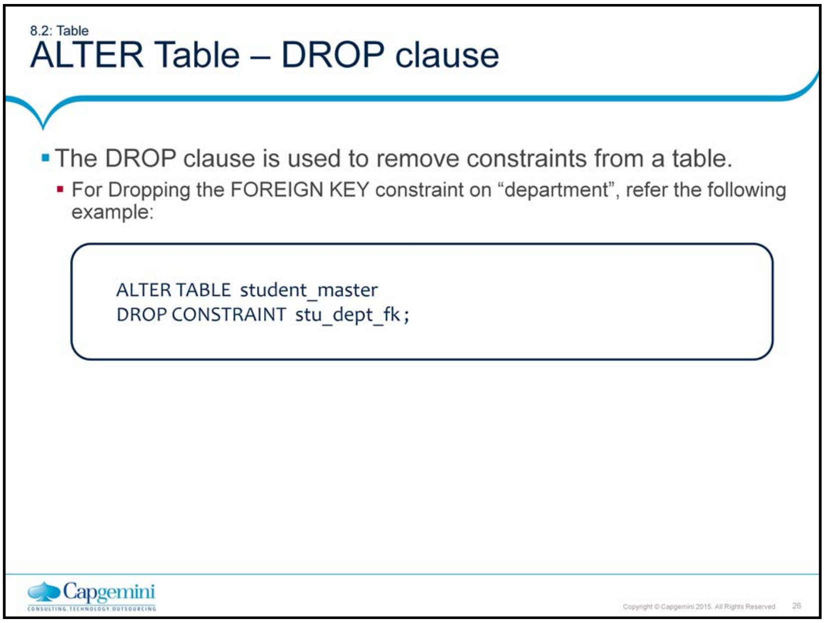


8.2: Table

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;
```


8.2: 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:

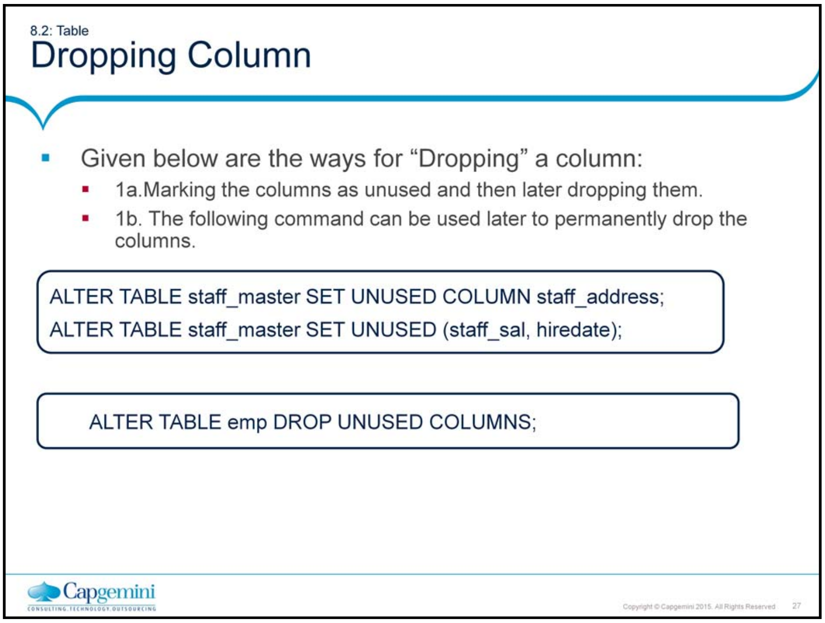
```
ALTER TABLE  student_master
DROP CONSTRAINT  stu_dept_fk ;
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

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;
```



8.2: 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;
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

Ways for "Dropping" a column:

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.

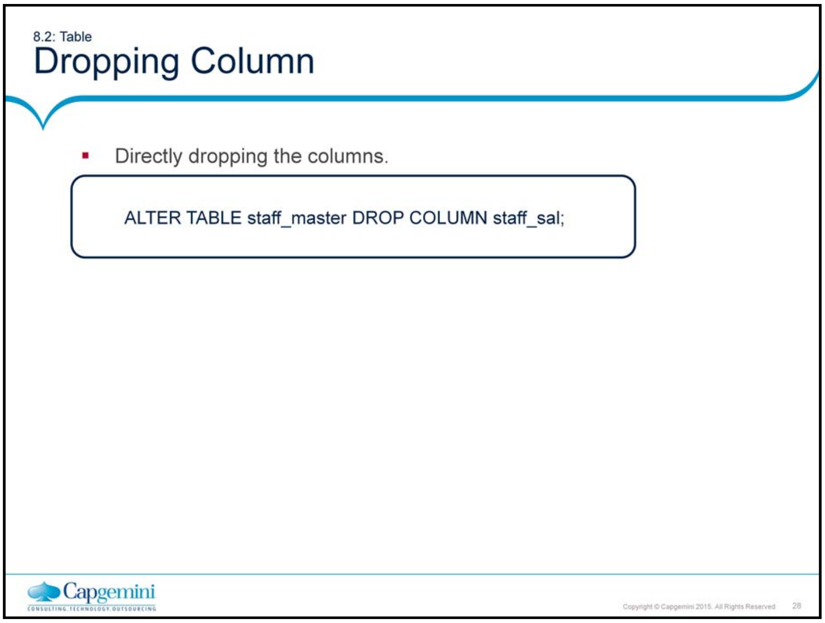
Marking the columns as unused does not release the space occupied by them back to the database.

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.



8.2: Table

Dropping Column

- Directly dropping the columns.

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

Ways for “Dropping” a column (contd.):

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.

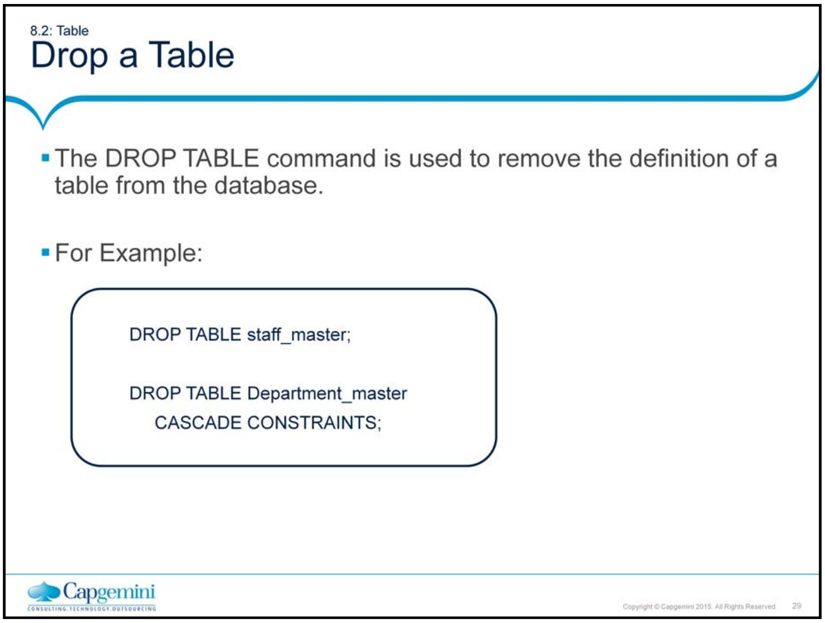
All “constraints” that reference a target column are removed.

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



8.2: Table

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

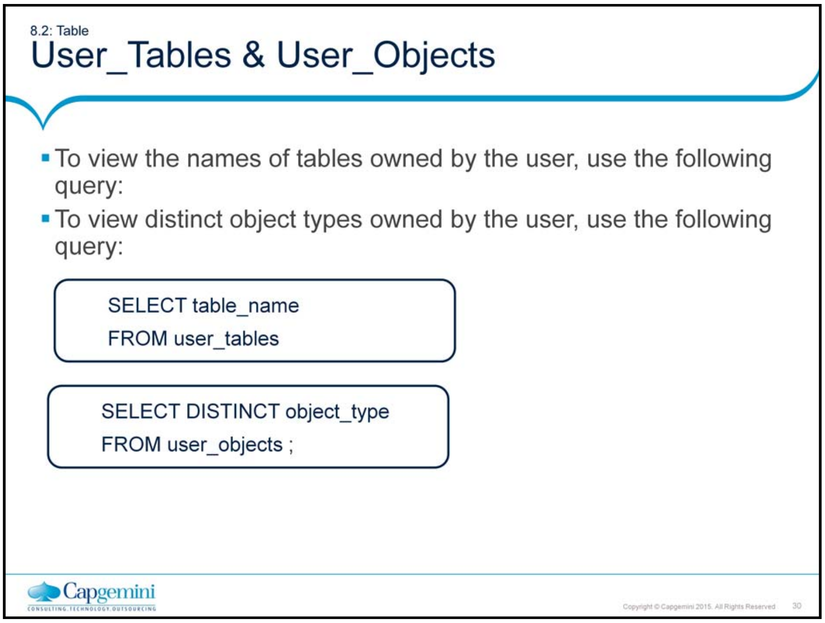
Deleting objects that exist in the database is an easy task. Just say:

```
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-name CASCADE CONSTRAINTS;
```



8.2: Table

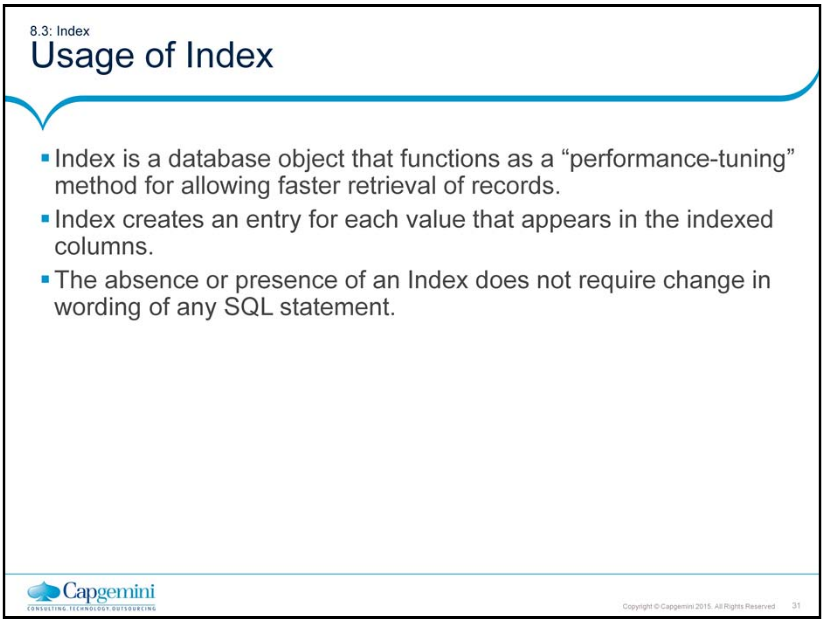
User_Tables & User_Objects

- 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 ;
```

30



8.3: 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

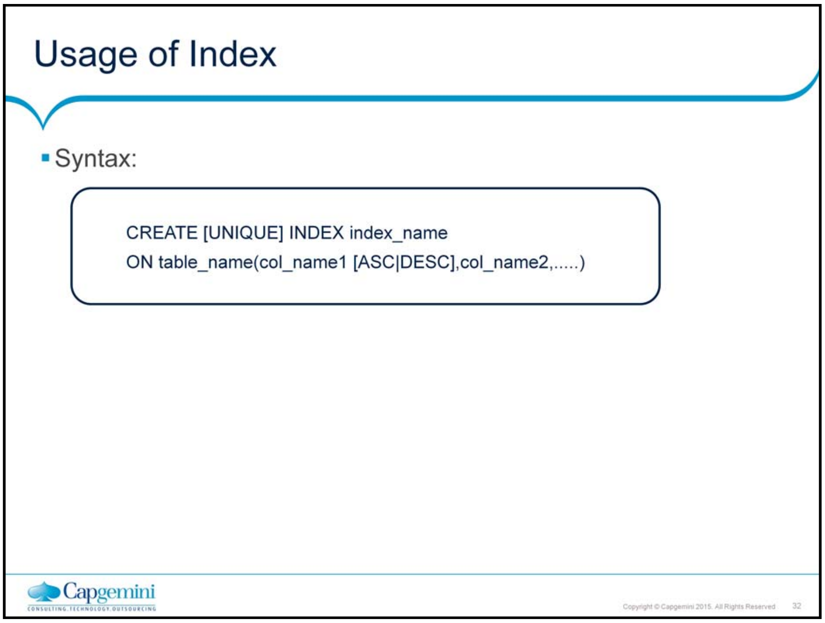
- Is used by the Oracle server to speed up the retrieval of rows by using a pointer

- 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



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.

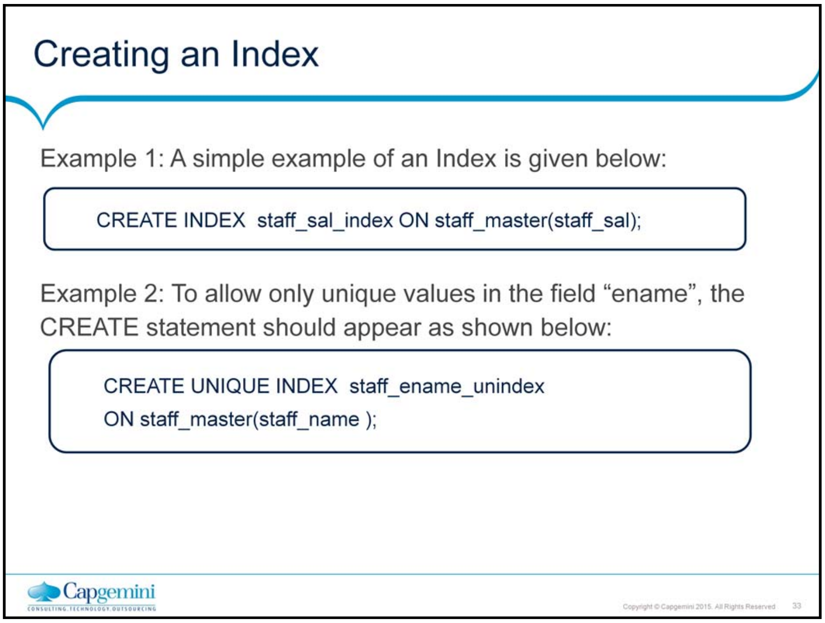
An Index is in an ascending order, by default.

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.



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?

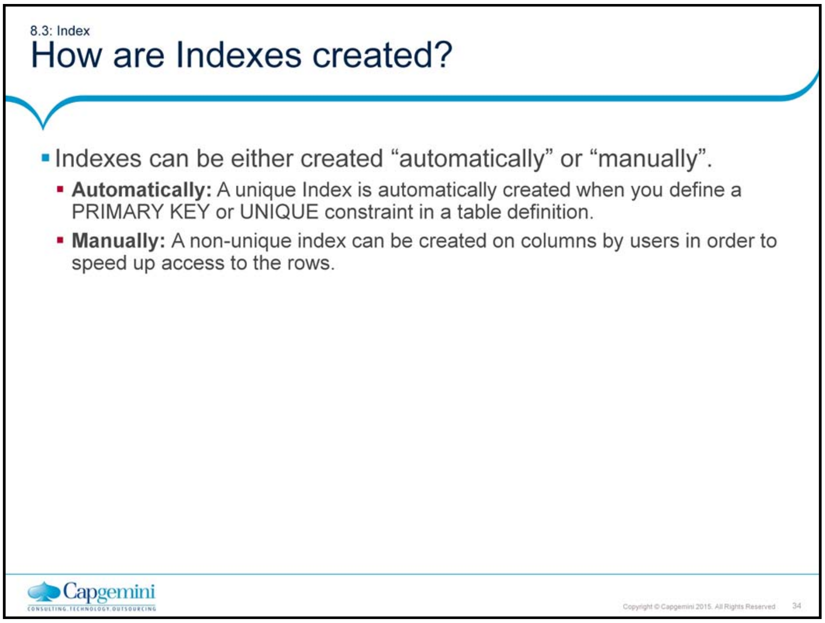
You should create Indexes, only if:

- 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.



8.3: 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:

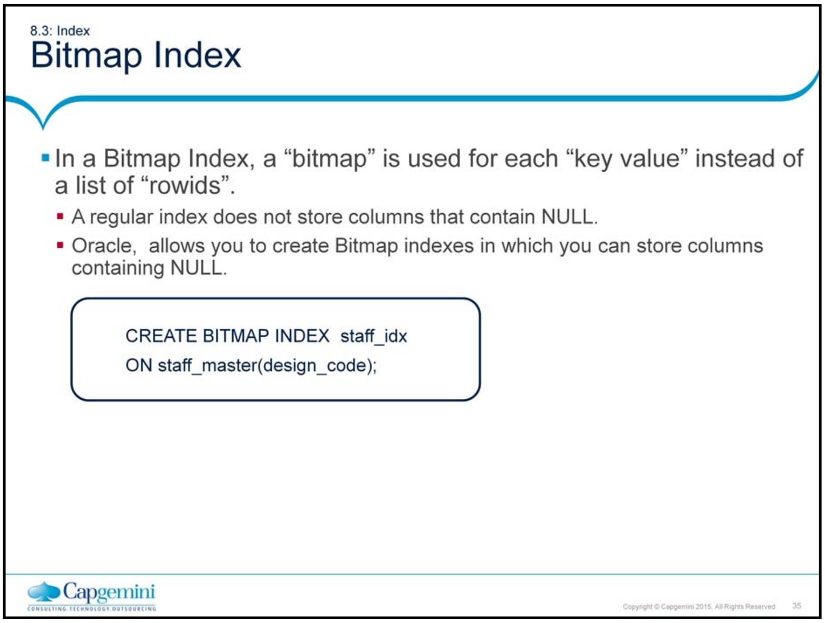
Two types of Indexes can be created.

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".



8.3: Index

Bitmap Index

- In a Bitmap Index, a "bitmap" is used for each "key value" instead of a list of "rowids".
  - A regular index does not store columns that contain NULL.
  - Oracle, allows you to create Bitmap indexes in which you can store columns containing NULL.

```
CREATE BITMAP INDEX  staff_idx
ON staff_master(design_code);
```

Bitmap Index:

The Bitmap Index is useful for some types of SQL statements like:

SELECT COUNT(*) FROM staff_master;

Bitmap Indexes are used to tune queries that use non-selective columns in their limiting conditions.

A Bitmap Index can have a maximum of 30 columns.

In Bitmap structures, a "two-dimensional array" is created with one column for every row in the table, which is indexed.

Each column represents a distinct value within the bitmapped index. This two-dimensional array represents "each value within the index" multiplied by the "number of rows" in the table.

In the "row retrieval" stage, the Oracle decompresses the bitmap into the RAM data buffers, such that it can be rapidly scanned for matching values.

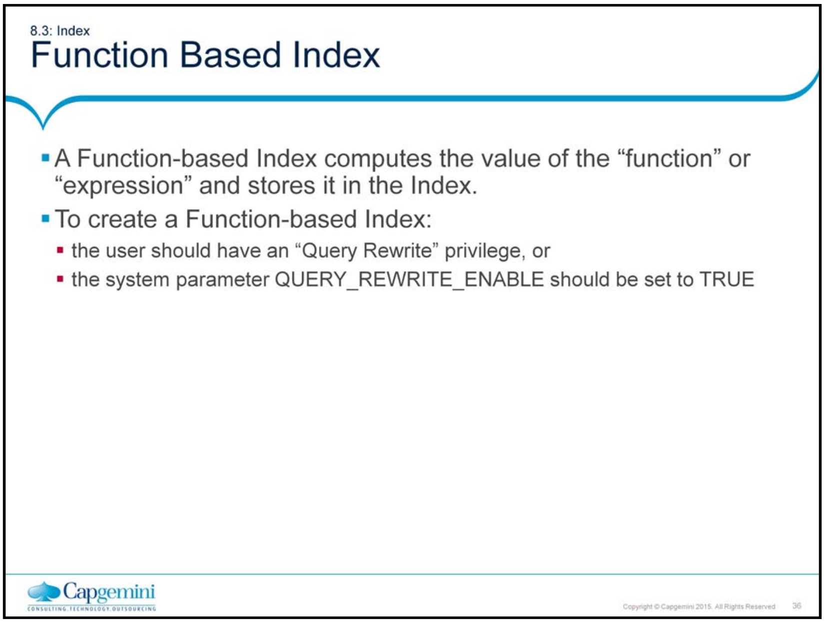
These matching values are delivered to Oracle in the form of a Row-ID list.

These Row-ID values may directly access the required information.

The real benefit of bitmapped indexing occurs when "one table" includes "multiple bitmapped indexes".

Each individual column may have low cardinality.

The creation of multiple bitmapped indexes provides a very powerful method for rapidly answering difficult SQL queries.



8.3: Index

Function Based Index

- A Function-based Index computes the value of the “function” or “expression” and stores it in the Index.
- To create a Function-based Index:
  - the user should have an “Query Rewrite” privilege, or
  - the system parameter QUERY_REWRITE_ENABLE should be set to TRUE

Function-based Index:

Function-based Indexes allow creation of indexes in PL/SQL and Java, on their:

Expressions

Internal functions

User-written functions

Function-based indexes ensure that the Oracle designer is able to use an index for it’s query.

Prior to Oracle8, the usage of a “built-in” function was not able to match the performance of an “Index”. Consequently, Oracle performed the dreaded full-table scan.

Examples of SQL with Function-based queries include the following:

Select * from customer where substr(cust_name,1,4) = ‘BURL’;

Select * from customer where to_char(order_date,’MM’) = ’01;

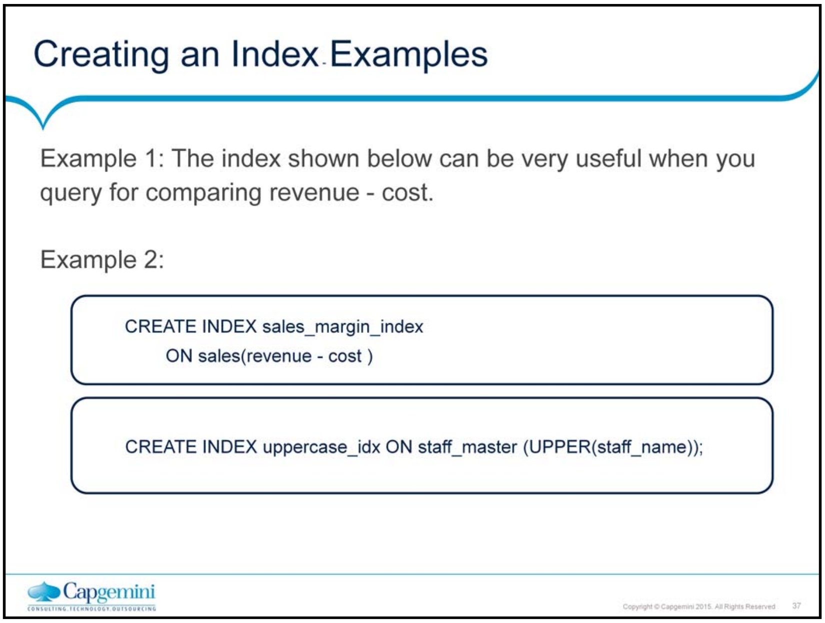
Select * from customer where upper(cust_name) = ‘JONES’;

Select * from customer where initcap(first_name) = ‘Mike’;

In Oracle, the dbms always interrogates the WHERE clause of the SQL statement to check the existence of a matching Index.

By using Function-based Indexes, the Oracle designer can create a matching index, which exactly matches the predicates within the SQL WHERE clause.

This ensures that the query is retrieved with a minimal amount of disk I/O at the fastest possible speed.



Creating an Index Examples

Example 1: The index shown below can be very useful when you query for comparing revenue - cost.

Example 2:

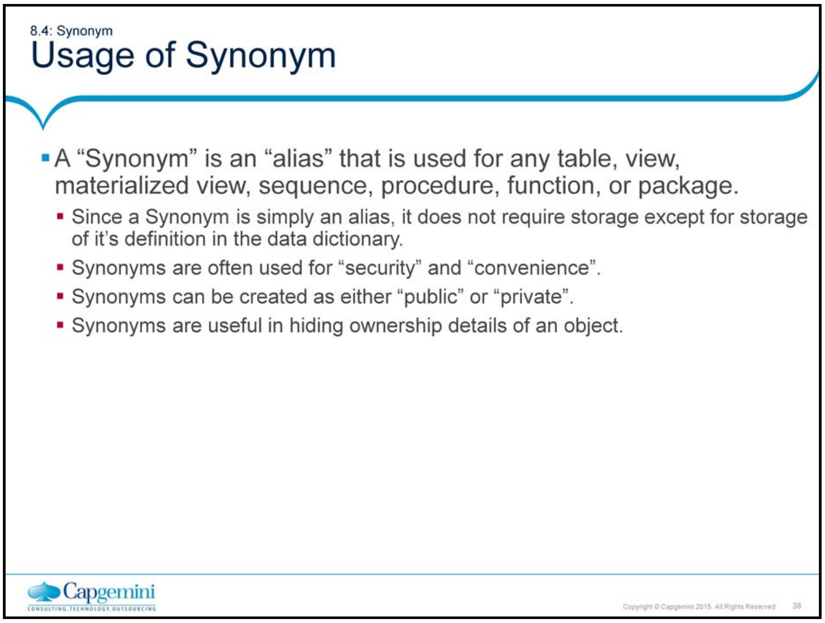
```
CREATE INDEX sales_margin_index
      ON sales(revenue - cost )
```

```
CREATE INDEX uppercase_idx ON staff_master (UPPER(staff_name));
```

Note:

The USER_INDEXES data dictionary view contains the name of the index and its uniqueness.

The USER_IND_COLUMNS view contains the index name, the table name, and the column name.



8.4: Synonym

Usage of Synonym

- A “Synonym” is an “alias” that is used for any table, view, materialized view, sequence, procedure, function, or package.
  - Since a Synonym is simply an alias, it does not require storage except for storage of it’s definition in the data dictionary.
  - Synonyms are often used for “security” and “convenience”.
  - Synonyms can be created as either “public” or “private”.
  - Synonyms are useful in hiding ownership details of an object.

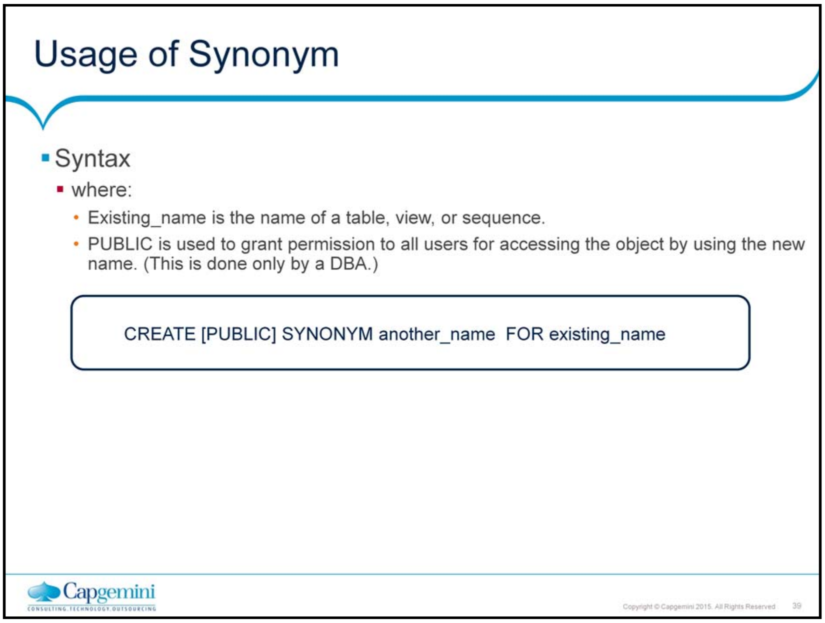
Synonym:

A Synonym simplifies access to objects. A Synonym is simply another name for an object.

With Synonyms, you can:

Ease referring to a table owned by another user.

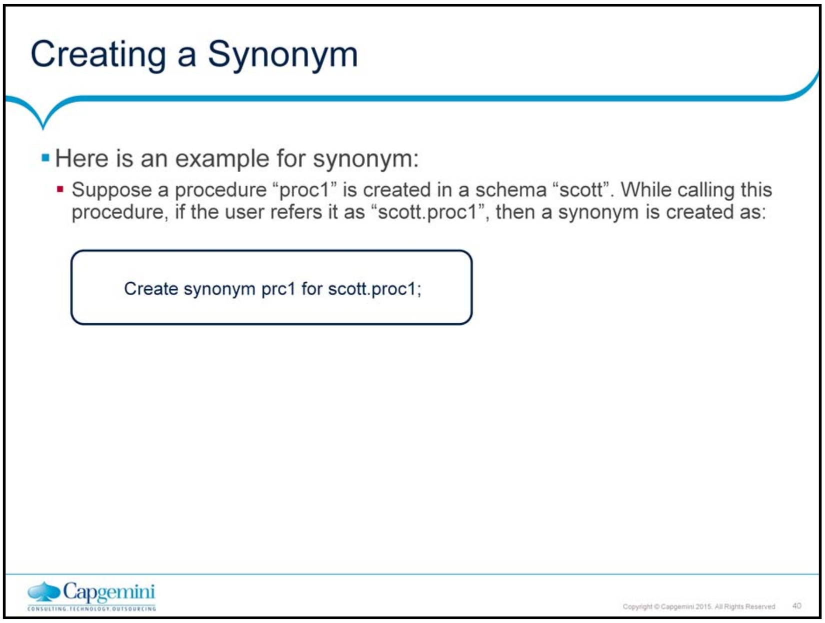
Shorten lengthy object names.



Usage of Synonym

- Syntax
  - where:
    - Existing_name is the name of a table, view, or sequence.
    - PUBLIC is used to grant permission to all users for accessing the object by using the new name. (This is done only by a DBA.)

```
CREATE [PUBLIC] SYNONYM another_name  FOR existing_name
```



Creating a Synonym

- Here is an example for synonym:
  - Suppose a procedure “proc1” is created in a schema “scott”. While calling this procedure, if the user refers it as “scott.proc1”, then a synonym is created as:

```
Create synonym prc1 for scott.proc1;
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

Creating and Removing Synonyms:

You can create a shortened name for the SALGRADE table as shown in the following example:

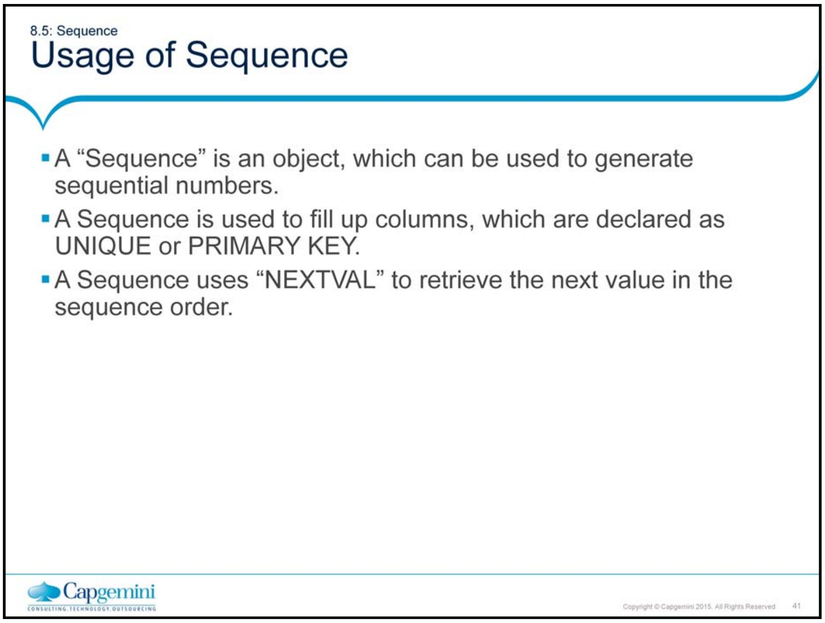
```
CREATE SYNONYM  s
FOR salgrade
Synonym Created.
```

You can create a shortened name for the DEPT_SUM_VU view as shown in the following example:

```
CREATE SYNONYM  d_sum
FOR  dept_sum_vu;
Synonym Created.
```

You can drop a Synonym as shown in the following example:

```
DROP SYNONYM d_sum;
Synonym dropped.
```



8.5: Sequence

Usage of Sequence

- A “Sequence” is an object, which can be used to generate sequential numbers.
- A Sequence is used to fill up columns, which are declared as UNIQUE or PRIMARY KEY.
- A Sequence uses “NEXTVAL” to retrieve the next value in the sequence order.

Sequence:

A Sequence:

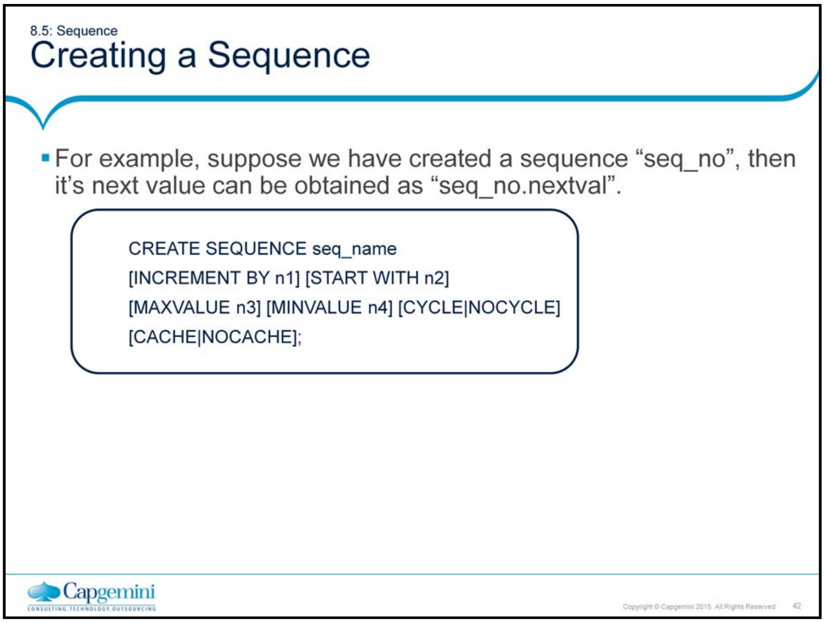
automatically generates unique numbers.

is a “sharable object”.

is typically used to create a PRIMARY KEY value.

replaces application code.

speeds up the efficiency of accessing sequence values when cached in memory.



8.5: Sequence

Creating a Sequence

- For example, suppose we have created a sequence “seq_no”, then it’s next value can be obtained as “seq_no.nextval”.

```
CREATE SEQUENCE seq_name
[INCREMENT BY n1] [START WITH n2]
[MAXVALUE n3] [MINVALUE n4] [CYCLE|NOCYCLE]
[CACHE|NOCACHE];
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

Sequence:

In the example shown in the slide:

START WITH indicates the first NUMBER in the series.

INCREMENT BY is the difference between consecutive numbers. If n1 is negative, then the numbers that are generated are in a descending order.

MAXVALUE and MINVALUE indicate the extreme values.

CYCLE indicates that once the extreme is reached, it starts the cycle again with n2.

NOCYCLE means that once the extreme is reached, it stops generating numbers.

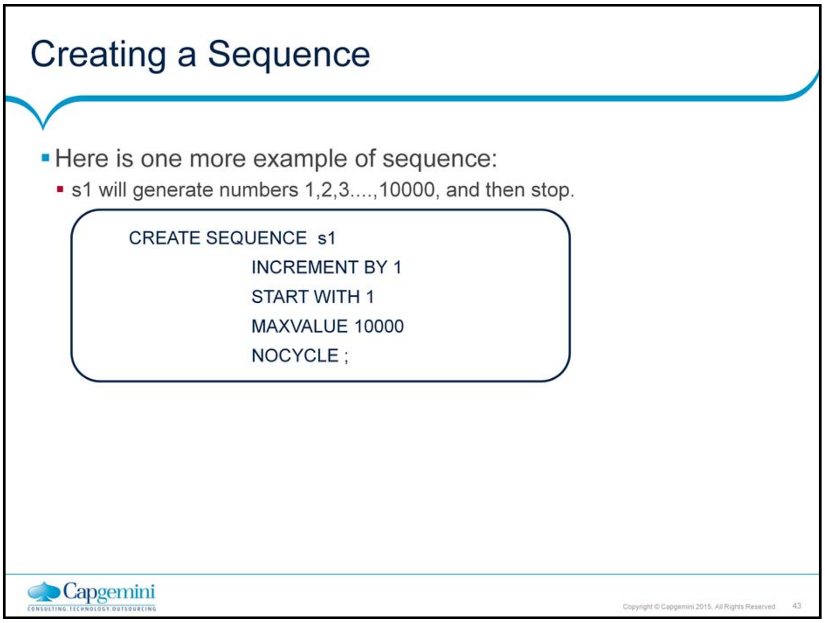
CACHE caches the specified number of sequence values in the memory. This speeds access, but all cached numbers are lost when the database is shut down. The default value is 20

Confirming Sequences

As shown below, you need to verify your sequence values in the USER_SEQUENCES data dictionary table.

```
SELECT sequence_name, min_value, max_value,
          increment_by, last_number
FROM user_sequences;
```

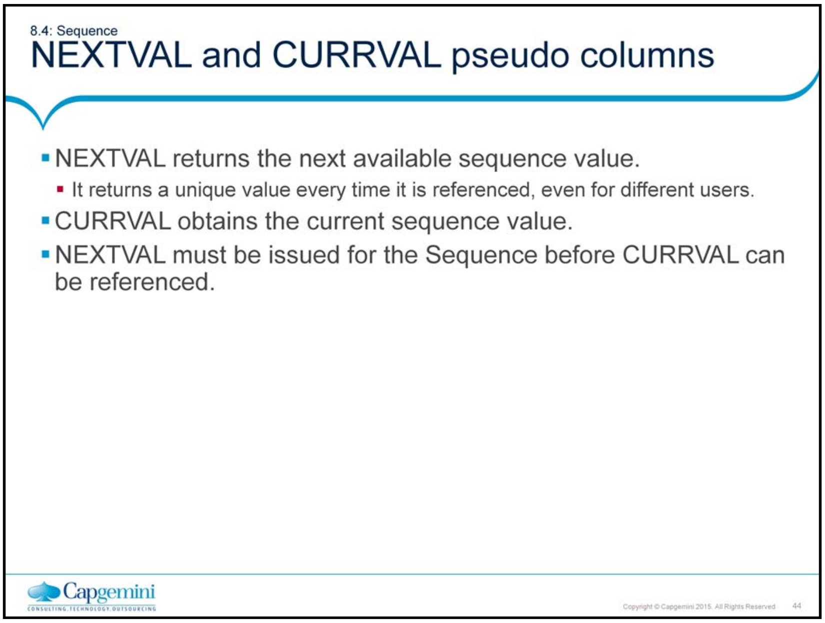
The LAST_NUMBER column displays the next available sequence number if NOCACHE is specified.



Creating a Sequence

- Here is one more example of sequence:
  - s1 will generate numbers 1,2,3....,10000, and then stop.

```
CREATE SEQUENCE  s1
            INCREMENT BY 1
            START WITH 1
            MAXVALUE 10000
            NOCYCLE ;
```


8.4: Sequence

NEXTVAL and CURRVAL pseudo columns

- NEXTVAL returns the next available sequence value.
  - It returns a unique value every time it is referenced, even for different users.
- CURRVAL obtains the current sequence value.
- NEXTVAL must be issued for the Sequence before CURRVAL can be referenced.

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 44

Referencing a Sequence:

After you create a Sequence, it generates sequential numbers that can be used in your tables. You can reference the Sequence values by using the NEXTVAL and CURRVAL pseudocolumns.

NEXTVAL and CURRVAL Pseudocolumns:

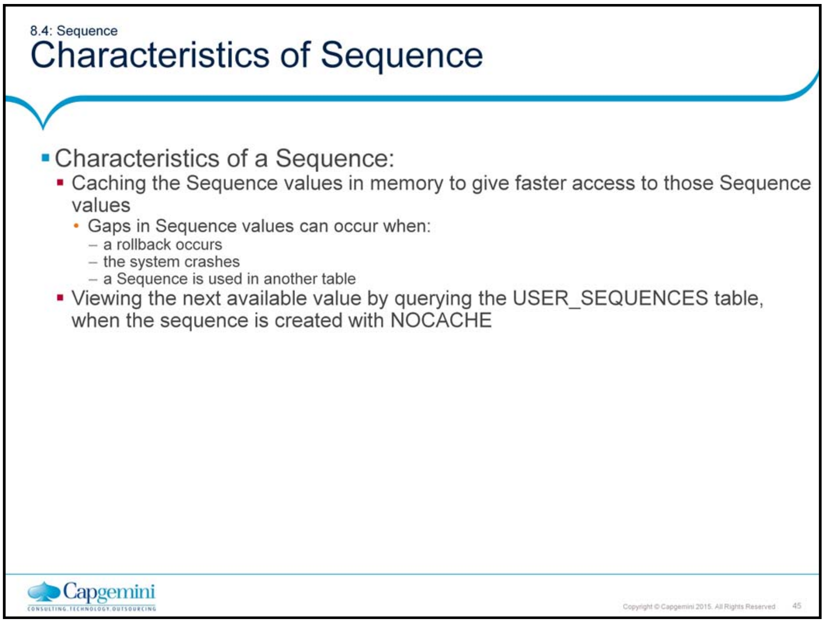
The NEXTVAL pseudocolumn is used to extract successive sequence numbers from a specified Sequence. You must qualify NEXTVAL with the sequence name. When you reference sequence.NEXTVAL, a new sequence number is generated and the current sequence number is placed in CURRVAL.

The CURRVAL pseudocolumn is used to refer a Sequence number that the current user has just generated.

NEXTVAL must be used to generate a sequence number in the session of the current user, before CURRVAL can be referenced.

You must qualify CURRVAL with the sequence name.

When “sequence.CURRVAL” is referenced, the last value returned to that user’s process is displayed.



8.4: Sequence

Characteristics of Sequence

- Characteristics of a Sequence:
  - Caching the Sequence values in memory to give faster access to those Sequence values
    - Gaps in Sequence values can occur when:
      - a rollback occurs
      - the system crashes
      - a Sequence is used in another table
  - Viewing the next available value by querying the USER_SEQUENCES table, when the sequence is created with NOCACHE

Copyright © Capgemini 2015. All Rights Reserved 45

By using a sequence you can Insert a new employee with empno 1 as shown below:

You can view the current value for the S1 sequence as shown below:

Usage of a Sequence:

Caching Sequence Values

Cache sequences in memory to provide faster access to those "sequence values".

The first time you refer the Sequence, the cache is populated.

Each request for the next Sequence value is retrieved from the cached sequence.

After the last sequence value is used, the next request for the Sequence pulls another cache of sequences into the memory.

Gaps in the Sequence

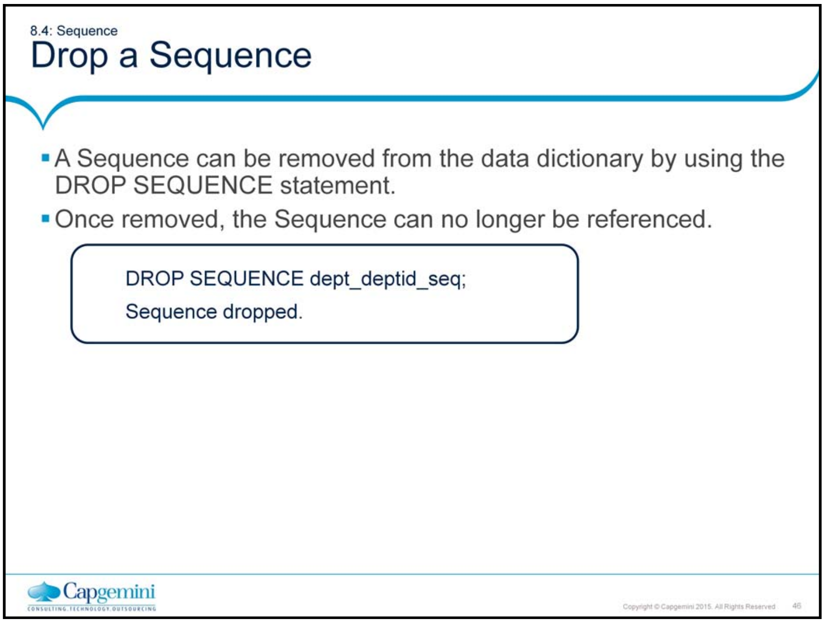
Although "sequence generators" issue "sequential numbers" without gaps, this action occurs independent of a commit or rollback. Therefore, if you roll back a statement containing a Sequence, the number is lost.

Another event that can cause gaps in the Sequence is a system crash. If the Sequence caches the values in memory, then those values are lost if there is a system crash.

Since Sequences are not directly tied to tables, the same Sequence can be used for multiple tables. If you do so, each table can contain gaps in the Sequential numbers.

Viewing the Next Available Sequence Value without Incrementing It

If the Sequence is created with NOCACHE, it is possible to view the next available Sequence value without incrementing it by querying the USER_SEQUENCES table.



8.4: Sequence

Drop a Sequence

- A Sequence can be removed from the data dictionary by using the DROP SEQUENCE statement.
- Once removed, the Sequence can no longer be referenced.

```
DROP SEQUENCE dept_deptid_seq;
Sequence dropped.
```

Removing a Sequence

To remove a sequence from the data dictionary, use the DROP SEQUENCE statement.

To remove the Sequence, you must be the owner of the Sequence or possess the DROP ANY SEQUENCE privilege.

Syntax:

```
DROP   SEQUENCE    sequence;
```

where: sequence is the name of the sequence generator.

For more information, refer Oracle9i SQL Reference, “DROP SEQUENCE”.

contd.

Altering a Sequence

- To alter a Sequence use the following syntax:

```
ALTER SEQUENCE s1
INCREMENT BY n1
MAXVALUE n3
CYCLE;
```

  - s1 is an existing Sequence.

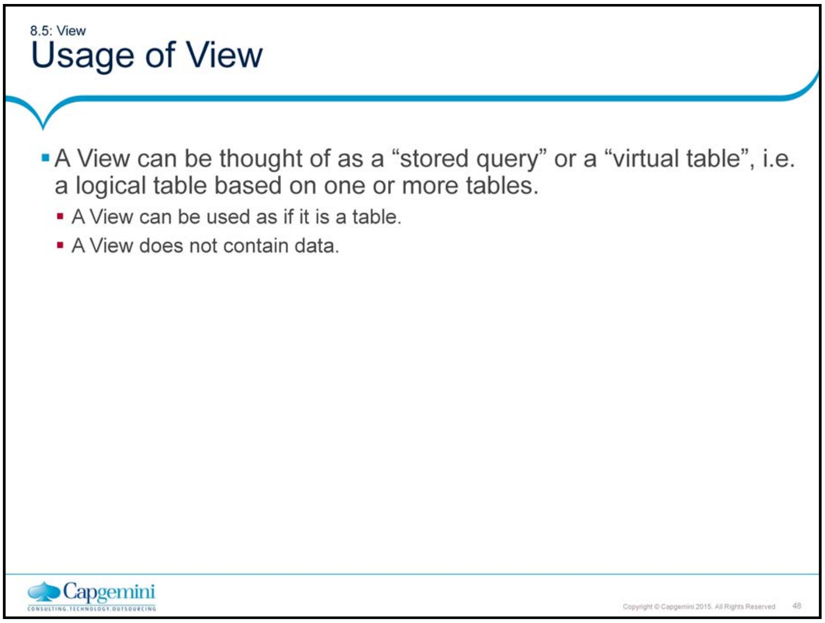
- To make the Sequence start from a new number, drop the Sequence and create it again.
- ALTER SEQUENCE has no effect on numbers that are already generated.

```
ALTER SEQUENCE s1
INCREMENT BY 5
MAXVALUE 25000
CYCLE;
```

Guidelines for modifying Sequences:

Given below are a few guidelines for modifying Sequences:

- You must be the owner or have the ALTER privilege for the Sequence.
- Only future sequence numbers are affected.
- The Sequence must be dropped, and re-created to restart the Sequence at a different number.
- Some validation should be performed.



8.5: 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.

Copyright © Capgemini 2015. All Rights Reserved 48

Why do we use Views?

Views are used:

To restrict data access.

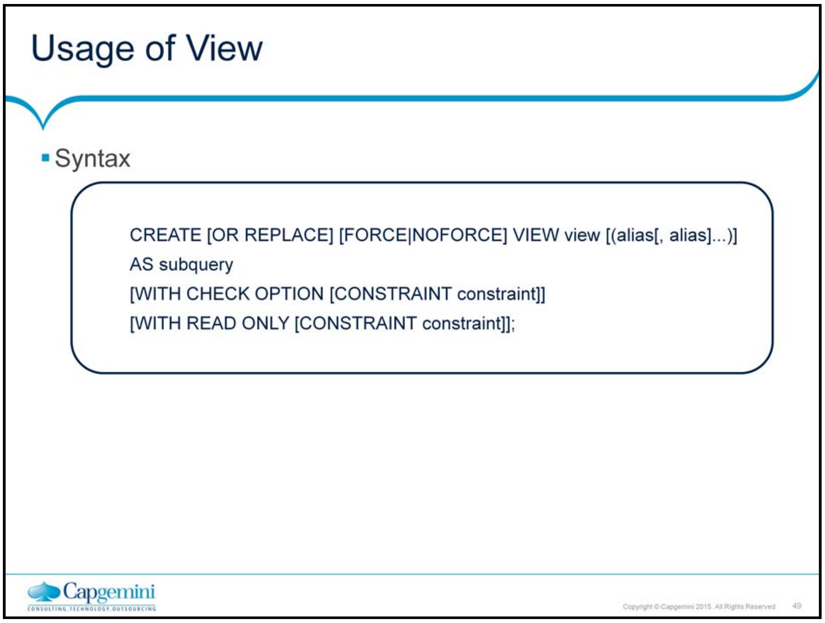
To make complex queries easy.

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



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”.

Understanding the syntax

OR REPLACE - re-creates the view if it already exists

FORCE - creates the view regardless of whether or not the base tables exist

NOFORCE - creates the view only if the base tables exist. This is default

View - is the name of the view

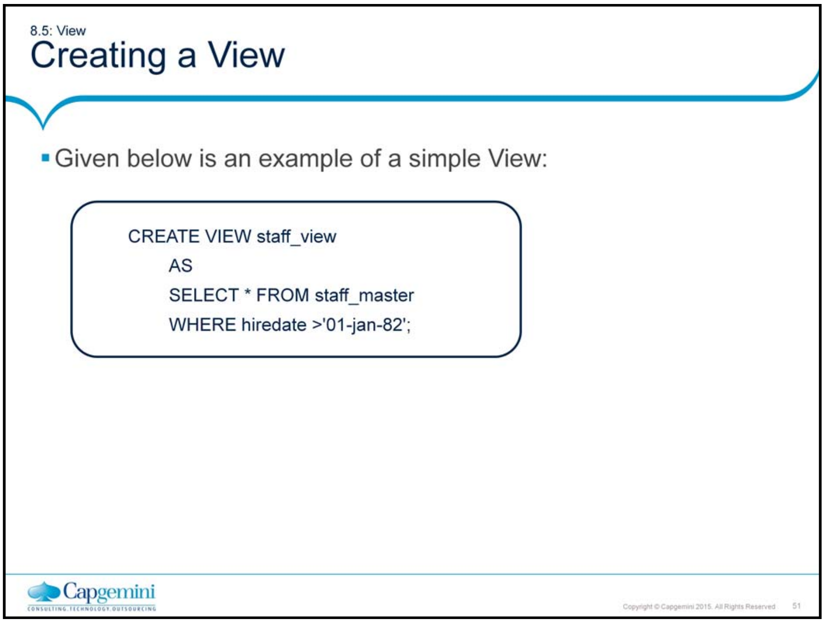
Alias - specifies names for the expressions selected by the view's query

Subquery - is a complete and valid SELECT statement

WITH CHECK OPTION - specifies that only rows accessible to the view can be inserted or updated

Constraint - is the name assigned to the CHECK OPTION

WITH READ ONLY - ensures that no DML operations can be performed on this view



8.5: 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';
```

Copyright © Capgemini 2015. All Rights Reserved 51

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.

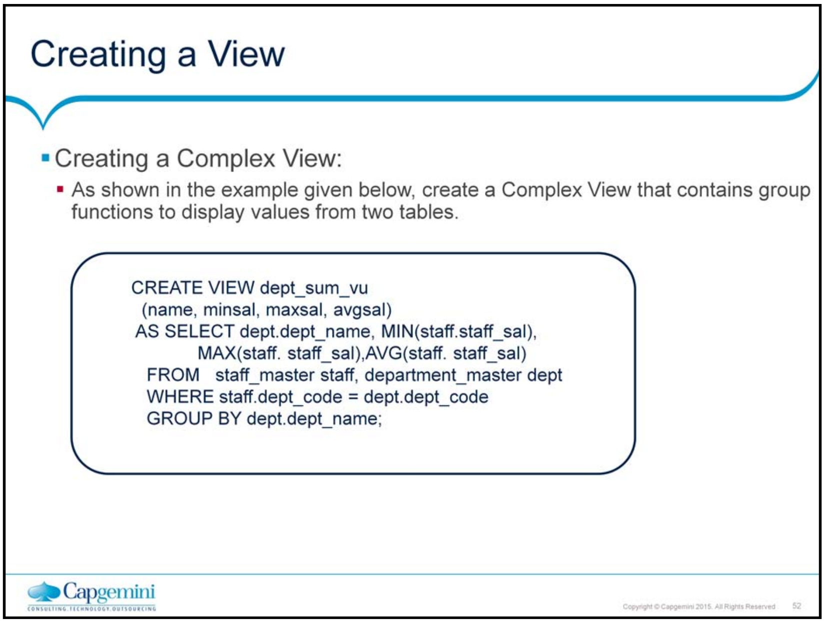
The restrictions are :

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.



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.

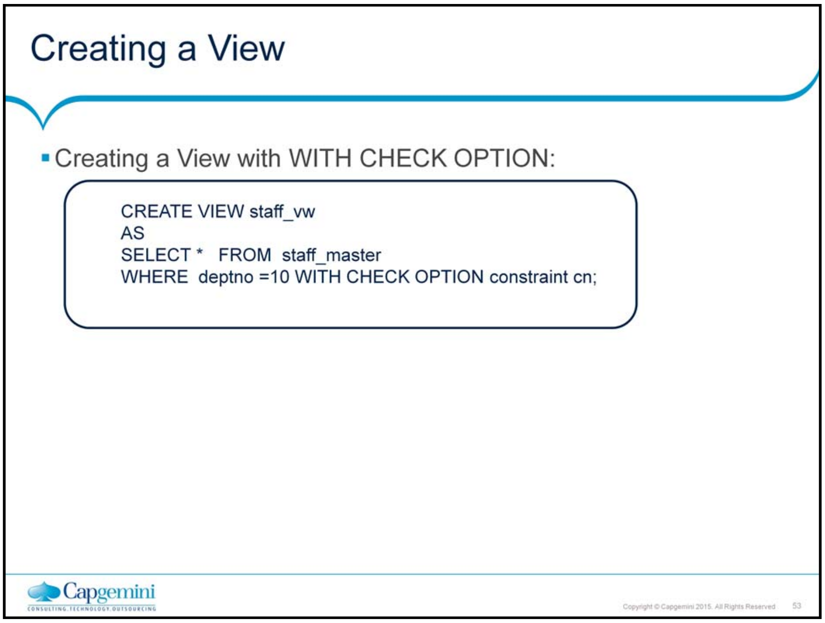
```
CREATE VIEW dept_sum_vu
 (name, minsal, maxsal, avgsal)
AS SELECT dept.dept_name, MIN(staff.staff_sal),
          MAX(staff. staff_sal),AVG(staff. staff_sal)
  FROM   staff_master staff, department_master dept
  WHERE staff.dept_code = dept.dept_code
  GROUP BY dept.dept_name;
```

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.



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

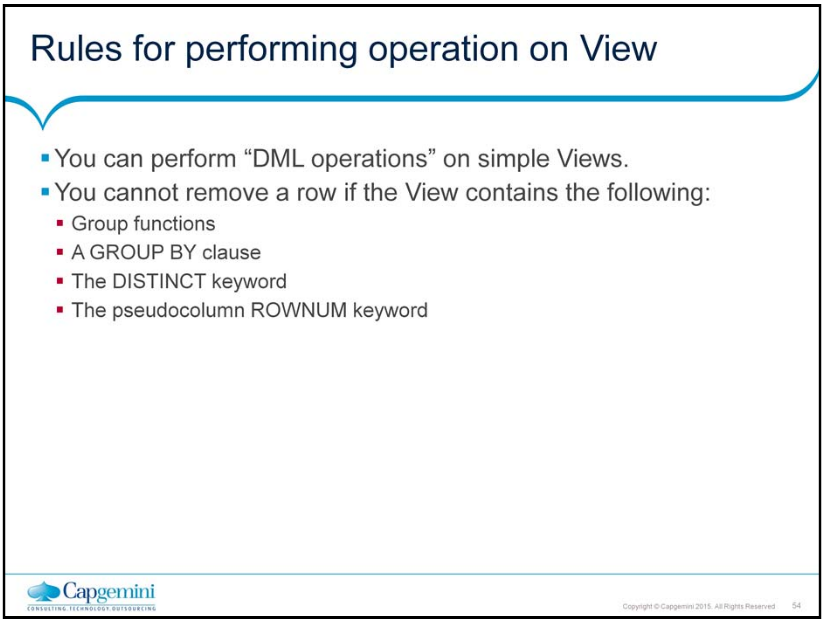
With the given error, the SQL tried to INSERT or UPDATE a record in a view that contained a WITH CHECK OPTION. The resulting INSERT or UPDATE violates the WHERE clause of the view.

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.
```



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

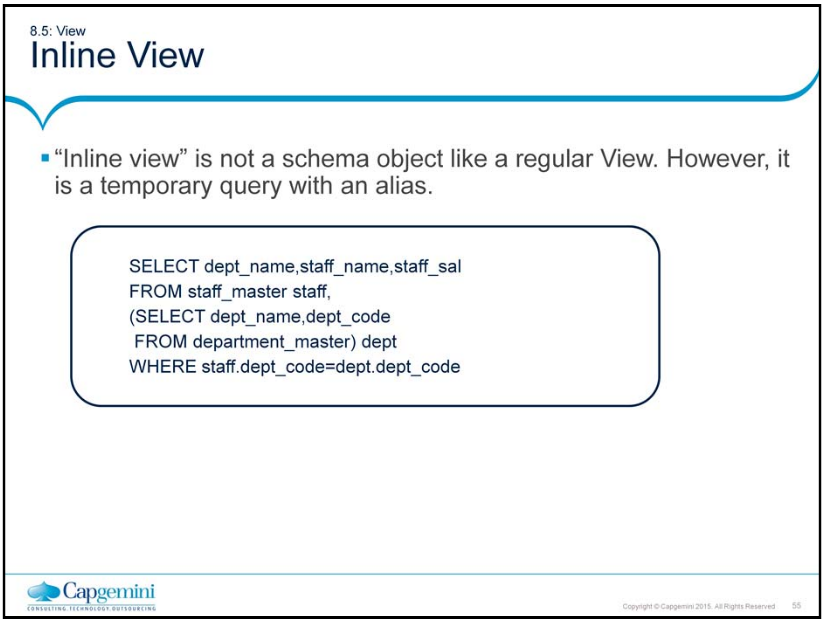
The DISTINCT keyword

The pseudocolumn ROWNUM keyword

Removing a View:

You can remove a View by using the following syntax:

```
DROP VIEW viewname;
```



8.5: View

Inline View

- "Inline view" is not a schema object like a regular View. However, it is a temporary query with an alias.

```
SELECT dept_name,staff_name,staff_sal
FROM staff_master staff,
(SELECT dept_name,dept_code
 FROM department_master) dept
WHERE staff.dept_code=dept.dept_code
```

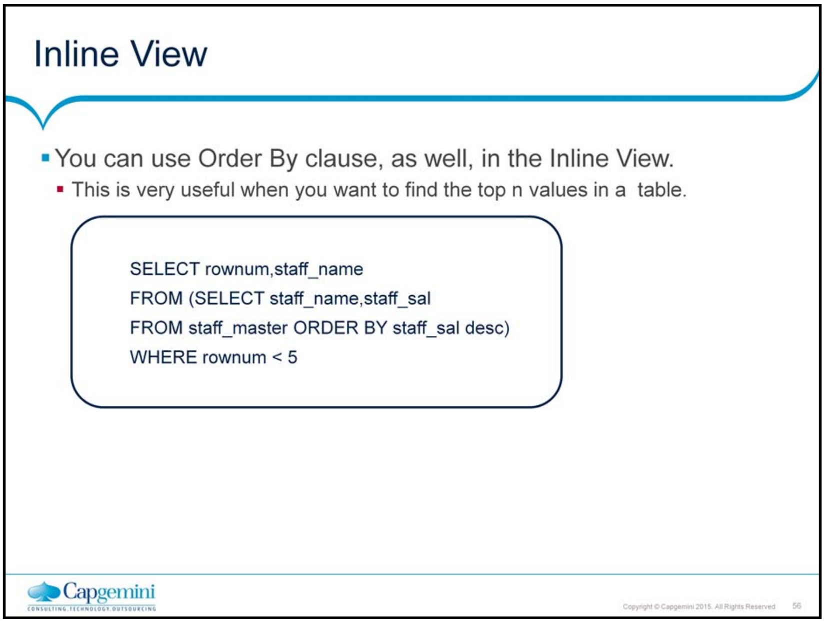
Inline View:

The Inline View is a construct in Oracle SQL, where you can place a query in the SQL FROM clause, just as if the query was a Table name.

In the example shown in the slide, the portion of the query, which is highlighted, is the Inline view.

A new feature has been included from Oracle 8i. You can use ORDER BY clause in the Inline View. This is very useful when you want to find the top n values in a table.

contd.



Inline View

- You can use Order By clause, as well, in the Inline View.
  - This is very useful when you want to find the top n values in a table.

```
SELECT rownum,staff_name
FROM (SELECT staff_name,staff_sal
FROM staff_master ORDER BY staff_sal desc)
WHERE rownum < 5
```

Copyright © Capgemini 2015. All Rights Reserved 56

Inline View (contd.):

Common uses for Inline Views, in Oracle SQL, are:

to simplify complex queries by removing “join” operations, and

to condense several separate queries into a single query

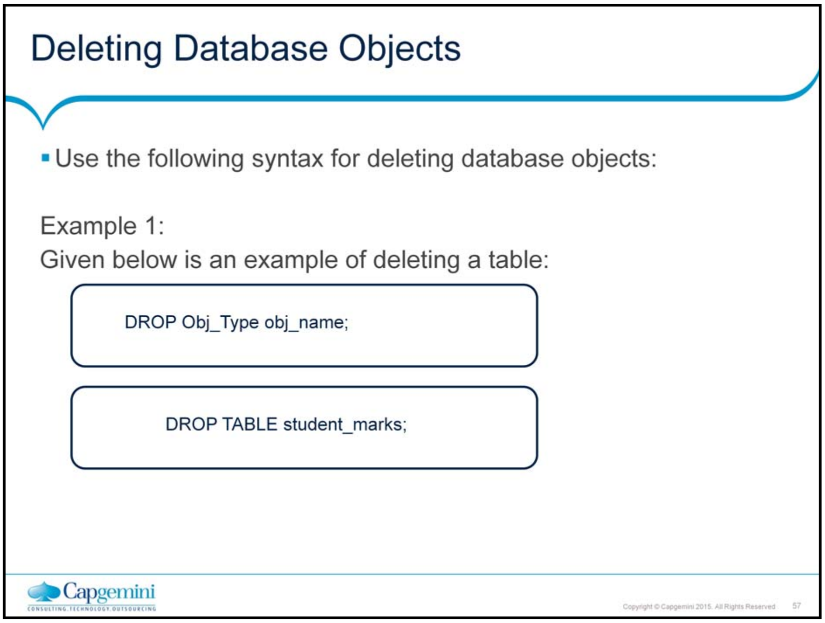
In ANSI standard SQL, it is quite difficult to compare two result sets that are summed together in a single query.

This is a common problem with Oracle SQL, where specific values must be compared to a summary.

Without the use of an Inline View, several separate SQL queries will have to be written:

one to compute the sums from each view, and

another to compare the intermediate result sets



Deleting Database Objects

- Use the following syntax for deleting database objects:

Example 1:
Given below is an example of deleting a table:

```
DROP Obj_Type obj_name;
```

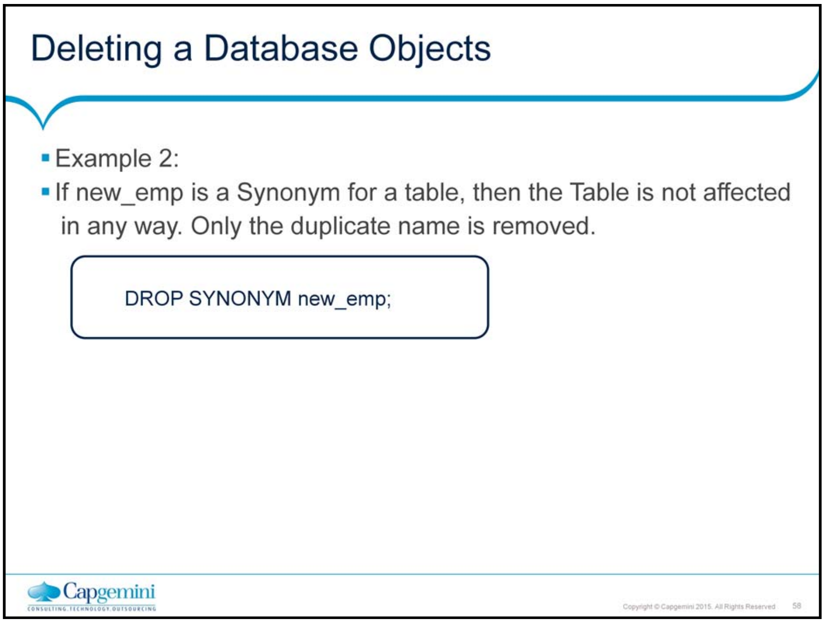
```
DROP TABLE student_marks;
```

Examples:

A table, which is dropped, cannot be recovered.

Dependent objects such as Indexes are automatically dropped.

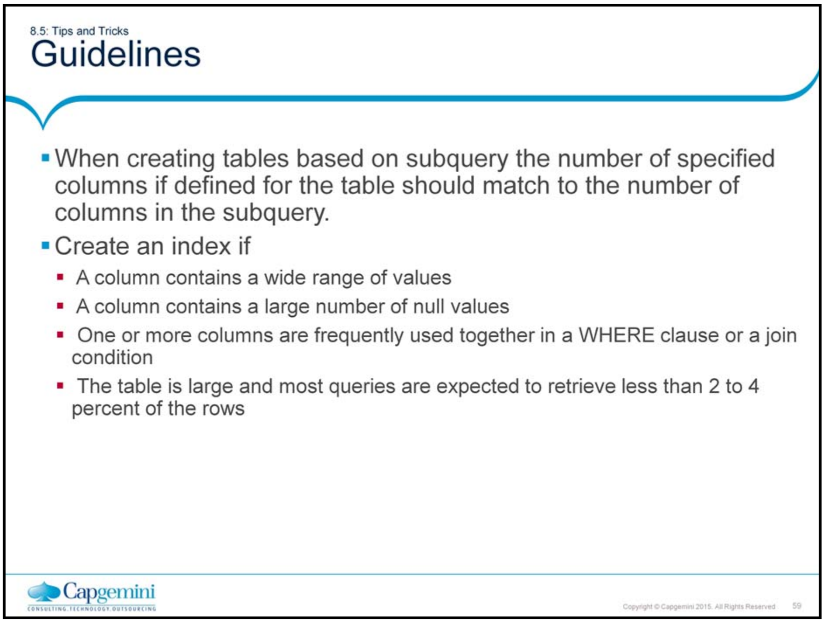
Synonyms and Views remain intact. However, they give an error when referenced.



Deleting a Database Objects

- Example 2:
- If new_emp is a Synonym for a table, then the Table is not affected in any way. Only the duplicate name is removed.

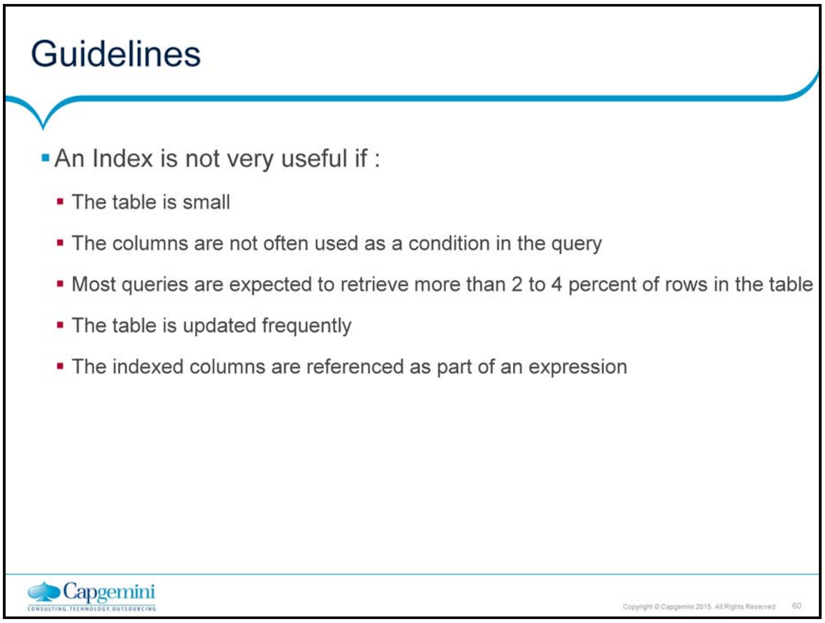
```
DROP SYNONYM new_emp;
```



8.5: Tips and Tricks

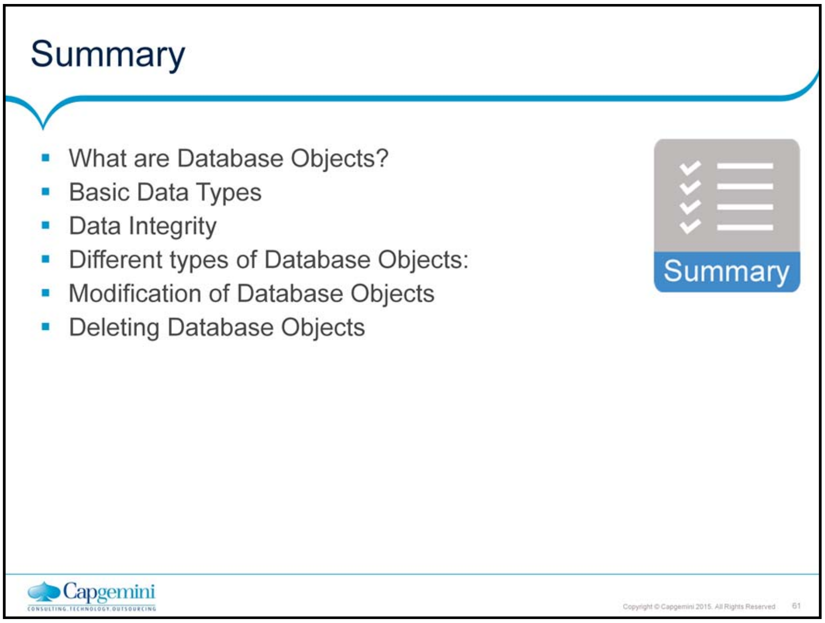
Guidelines

- 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



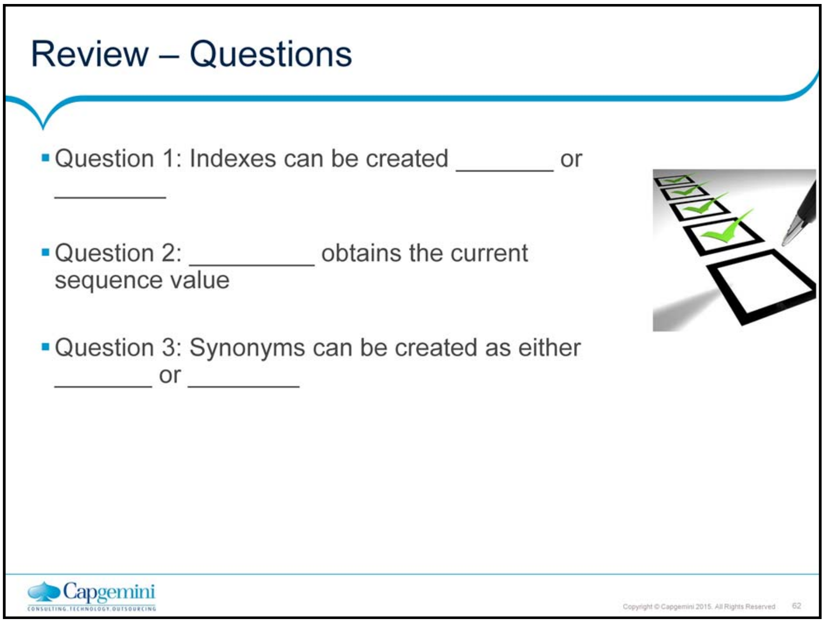
Guidelines

- 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



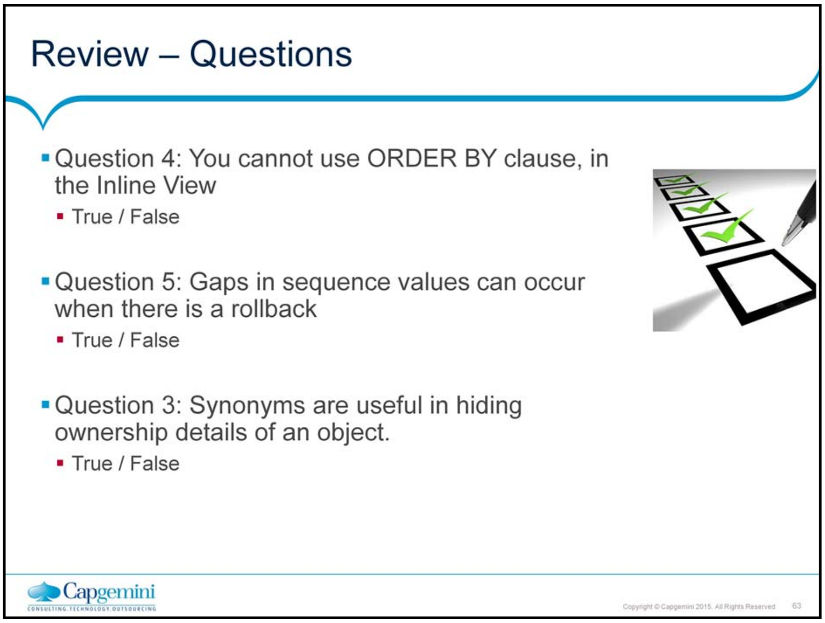
Summary

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



Review – Questions

- Question 1: Indexes can be created ________ or __________
- Question 2: __________ obtains the current sequence value
- Question 3: Synonyms can be created as either ________ or _________



Review – Questions

- Question 4: You cannot use ORDER BY clause, in the Inline View
  - True / False
- Question 5: Gaps in sequence values can occur when there is a rollback
  - True / False
- Question 3: Synonyms are useful in hiding ownership details of an object.
  - True / False