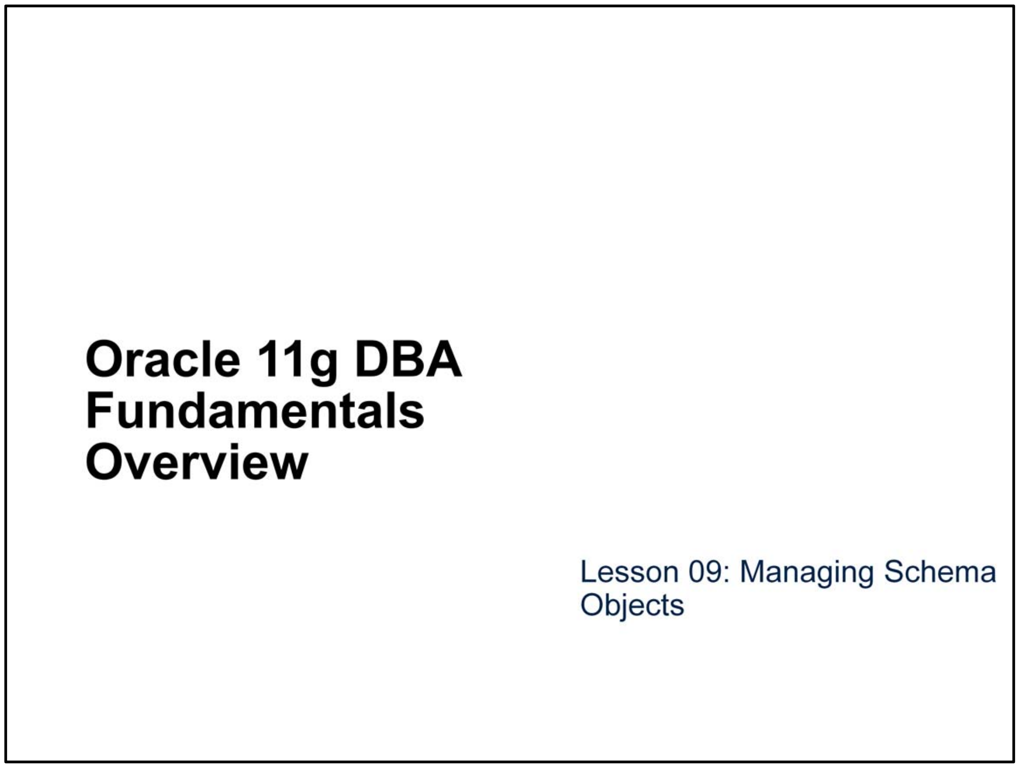
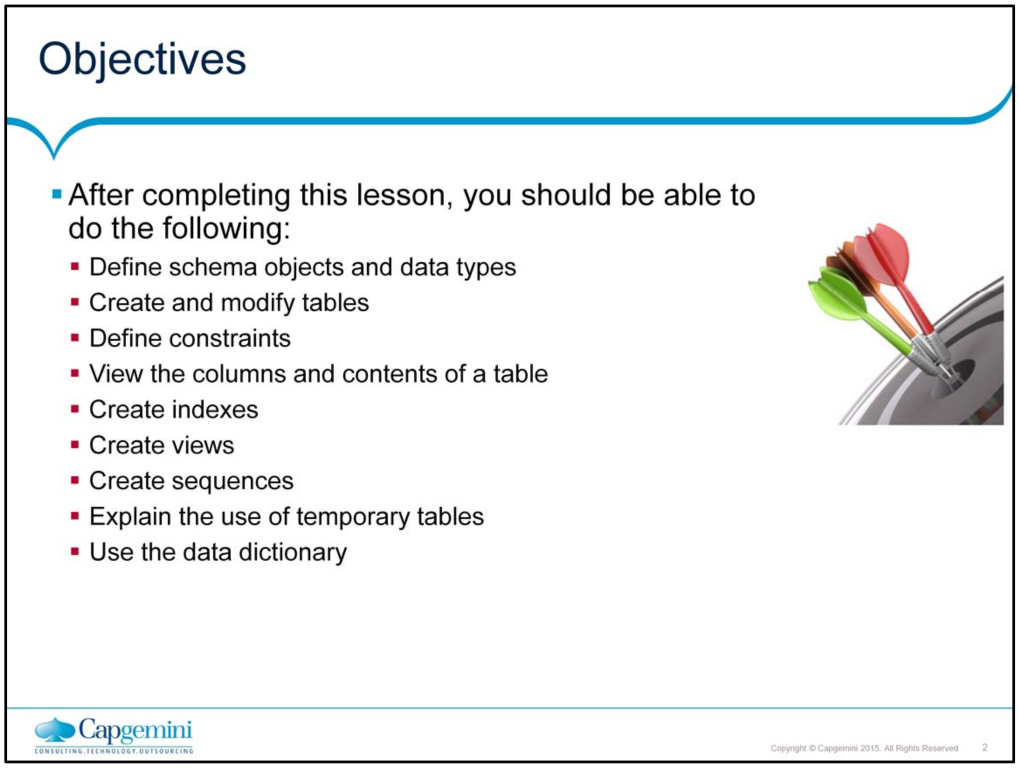


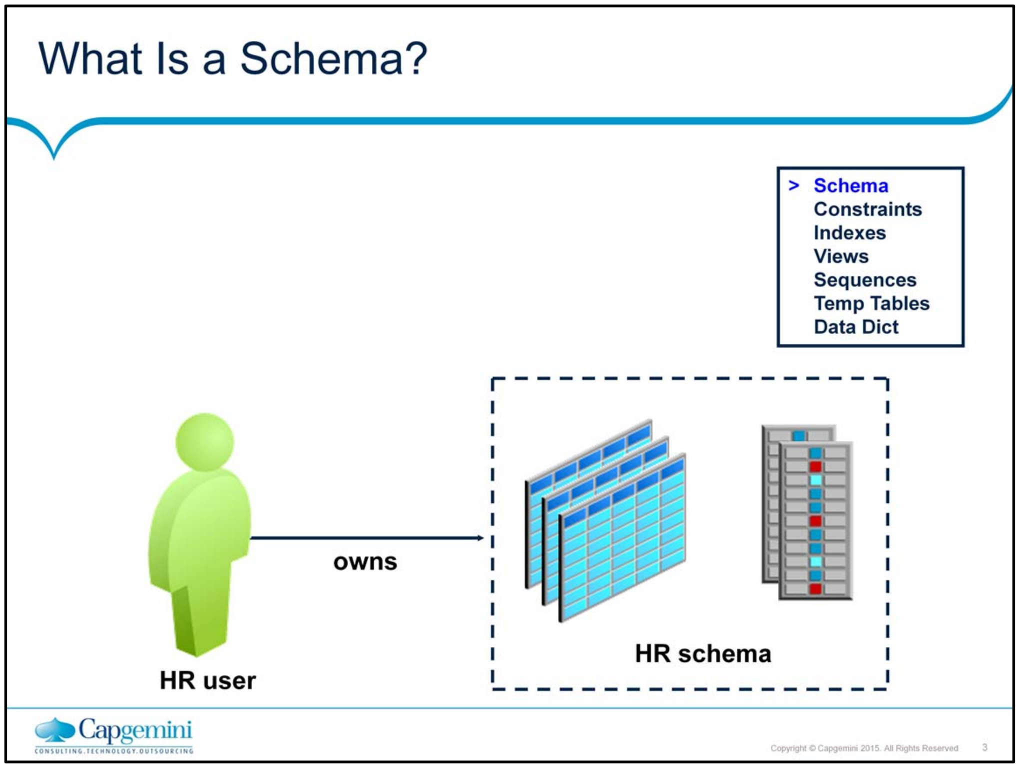
Oracle 11g DBA Fundamentals Overview

Lesson 09: Managing Schema Objects



Objectives

- After completing this lesson, you should be able to do the following:
  - Define schema objects and data types
  - Create and modify tables
  - Define constraints
  - View the columns and contents of a table
  - Create indexes
  - Create views
  - Create sequences
  - Explain the use of temporary tables
  - Use the data dictionary



What Is a Schema?

What Is a Schema?

A schema is a collection of database objects that are owned by a particular user. Typically, for a production database, this user does not represent a person, but an application. A schema has the same name as the user that owns the schema. Schema objects are the logical structures that directly refer to database’s data. Schema objects include structures such as tables, views, and indexes.

You can create and manipulate schema objects by using SQL or Enterprise Manager. When you use Enterprise Manager, the underlying SQL is generated for you.

Note: A schema does not necessarily have to be directly related to a single tablespace. You can define configurations such that objects in a single schema can be in different tablespaces, and a tablespace can hold objects from different schemas.

What Is a Schema? (continued)

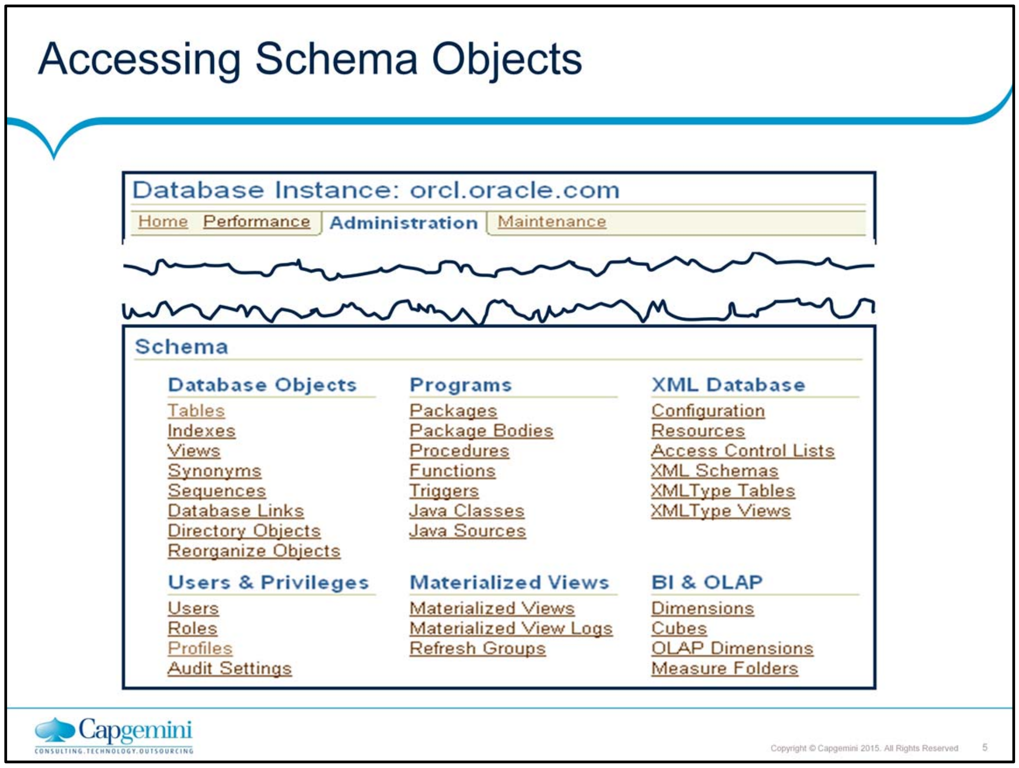
When you create the database, several schemas are created for you. Two of particular importance are the following:

- **SYS schema:** This contains the data dictionary, as described in the lesson titled "Administering User Security."

- **SYSTEM schema:** It contains additional tables and views that store administrative information. This is described in the lesson titled "Administering User Security."

During a complete installation of an Oracle database, sample schemas are installed automatically. Sample schemas serve the purpose of providing a common platform for examples in Oracle documentation and curricula. They are a set of interlinked schemas aimed at providing examples of different levels of complexity and include the following:

- **HR:** The Human Resources schema is a simple schema for introducing basic topics. An extension to this schema supports Oracle Internet Directory demonstrations.
- **OE:** The Order Entry schema deals with matters of intermediate complexity. A multitude of data types are available in the OE schema. The OC (Online Catalog) subschema is a collection of object-relational database objects built inside the OE schema.
- **PM:** The Product Media schema is dedicated to multimedia data types.
- **QS:** The Queued Shipping schema contains a set of schemas that are used to demonstrate Oracle Advanced Queuing capabilities.
- **SH:** The Sales History schema allows demonstrations with larger amounts of data. An extension to this schema provides support for advanced analytic processing.



Accessing Schema Objects

Database Instance: orcl.oracle.com

Home | Performance | Administration | Maintenance

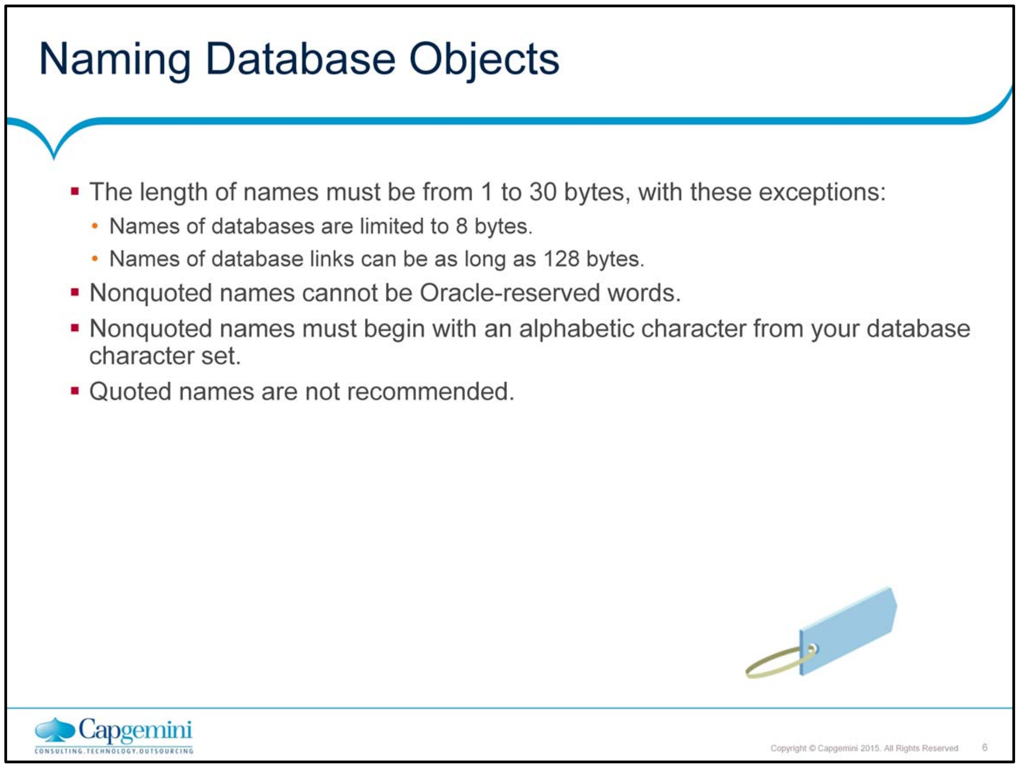
Schema

Database Objects	Programs	XML Database
Tables	Packages	Configuration
Indexes	Package Bodies	Resources
Views	Procedures	Access Control Lists
Synonyms	Functions	XML Schemas
Sequences	Triggers	XMLType Tables
Database Links	Java Classes	XMLType Views
Directory Objects	Java Sources	
Reorganize Objects		
Users & Privileges	**Materialized Views**	**BI & OLAP**
Users	Materialized Views	Dimensions
Roles	Materialized View Logs	Cubes
Profiles	Refresh Groups	OLAP Dimensions
Audit Settings		Measure Folders

Accessing Schema Objects

You can quickly access many types of schema objects from the Schema region of the Database Administration page.

After clicking one of the links, the Results page is displayed. In the Search region of the page, you can enter a schema name and object name to search for a specific object. In addition, you can search for other types of objects from the Search region by selecting the object type from the drop-down list. The drop-down list includes additional object types that are not shown as links on the Database Administration page.



Naming Database Objects

- The length of names must be from 1 to 30 bytes, with these exceptions:
  - Names of databases are limited to 8 bytes.
  - Names of database links can be as long as 128 bytes.
- Nonquoted names cannot be Oracle-reserved words.
- Nonquoted names must begin with an alphabetic character from your database character set.
- Quoted names are not recommended.

Naming Database Objects

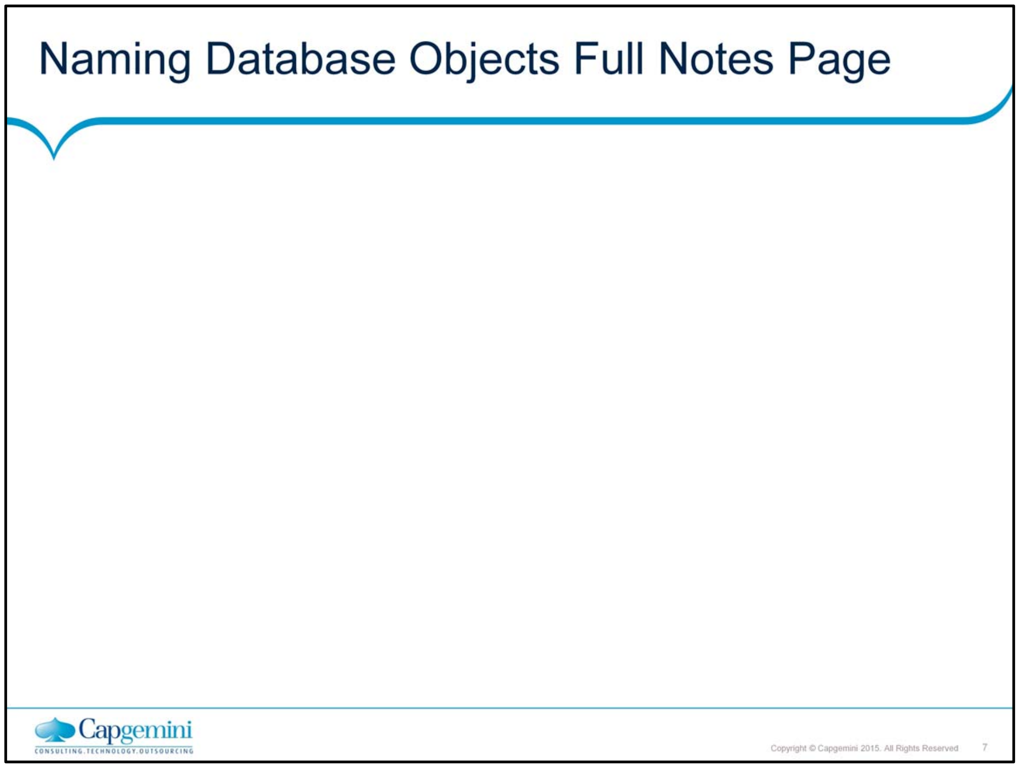
When you name an object in the database, you can enclose the name in double quotation marks (“”). If you do this, you can break several of the naming rules mentioned in the slide. However, this is not recommended because if you name an object this way, you must always refer to it with the quotation marks around the name. For example, if you name a table “Local Temp,” you must do the following:

```
SQL>  select * from "Local Temp";
TEMP_DATE    LO_TEMP    HI_TEMP
--------- ---------- ----------
01-DEC-03         30         41
```

If you enter the name in the wrong case, then you get an error:

```
SQL> select * from "local temp";
select * from "local temp"
              *
ERROR at line 1:
ORA-00942: table or view does not exist
```

Nonquoted names are stored in uppercase and are not case sensitive. When a SQL statement is processed, nonquoted names are converted to all uppercase.



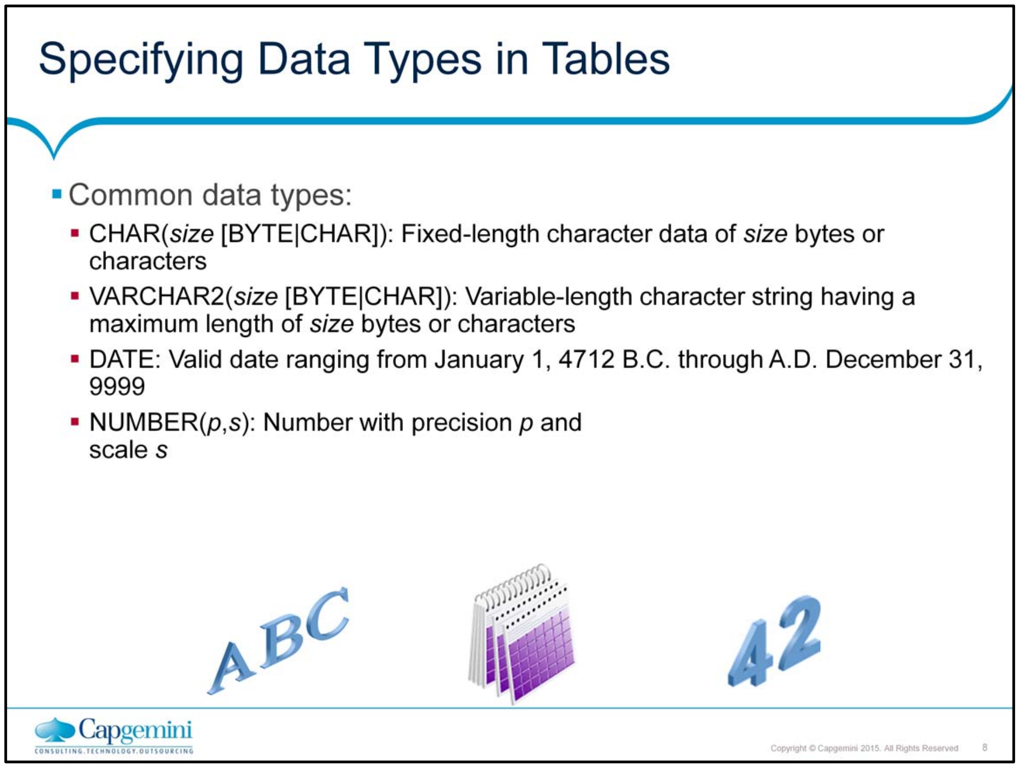
Naming Database Objects Full Notes Page

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Naming Database Objects (continued)

Nonquoted identifiers can contain only alphanumeric characters from your database character set and the underscore (_), the dollar sign ($), and the pound sign (#). Database links can also contain periods (.) and the “at” sign (@). You are strongly discouraged from using $ and # in nonquoted identifiers.

Quoted identifiers can contain any characters and punctuation marks as well as spaces. However, neither quoted nor nonquoted identifiers can contain double quotation marks.



Specifying Data Types in Tables

- Common data types:
  - CHAR(*size* [BYTE|CHAR]): Fixed-length character data of *size* bytes or characters
  - VARCHAR2(*size* [BYTE|CHAR]): Variable-length character string having a maximum length of *size* bytes or characters
  - DATE: Valid date ranging from January 1, 4712 B.C. through A.D. December 31, 9999
  - NUMBER(*p*,*s*): Number with precision *p* and scale *s*

Specifying Data Types in Tables

When you create a table, you must specify a data type for each of its columns. When you create a procedure or function, you must specify a data type for each of its arguments. These data types define the domain of values that each column can contain or each argument can have.

Built-in data types in the Oracle database include the following:

CHAR: Fixed-length character data of size bytes or characters. The maximum size is 2,000 bytes or characters; the default and minimum size is 1 byte.

BYTE indicates that the column has byte length semantics.

CHAR indicates that the column has character semantics.

VARCHAR2: Variable-length character string having maximum length size bytes or characters. The maximum size is 4,000 bytes. You must specify the size for VARCHAR2.

DATE: Valid date ranging from January 1, 4712 B.C. through A.D. December 31, 9999. It also stores the time: hours, minutes, and seconds.

NUMBER: Number with precision p and scale s. The precision can range from 1 through 38. The scale can range from –84 through 127.

Specifying Data Types in Tables (continued)

BINARY_FLOAT: This is a 32-bit floating-point number. This data type requires 5 bytes, including the length byte.
BINARY_DOUBLE: This is a 64-bit floating-point number. This data type requires 9 bytes.
FLOAT(*p*): This is an American National Standards Institute (ANSI) data type. The FLOAT data type is a floating-point number with a binary precision *p*. The default precision for this data type is 126 binary or 38 decimal.
INTEGER: This is equivalent to NUMBER(p,0).
NCHAR(*length*): The NCHAR data type is a Unicode-only data type. When you create a table with an NCHAR column, you define the column length in characters. You define the national character set when you create your database. The maximum length of a column is determined by the national character set definition. The width specifications of the NCHAR data type refer to the number of characters. The maximum column size allowed is 2,000 bytes. If you insert a value that is less than the column length, the Oracle database pads the value with blanks for full column length. You cannot insert a CHAR value into an NCHAR column, nor can you insert an NCHAR value into a CHAR column.

NVARCHAR2(size [BYTE|CHAR]): The NVARCHAR2 data type is a Unicode-only data type. It is like NCHAR except that its maximum length is 4,000 bytes and it is not blank-padded.
LONG: This is a character data of variable length of up to 2 gigabytes or 2^{31} –1 bytes. The LONG data type is deprecated; use the large object (LOB) data type instead.
LONG RAW: This is raw binary data of variable length of up to 2 gigabytes.
RAW(*size*): This is raw binary data of length *size* bytes. The maximum size is 2,000 bytes. You must specify the size for a RAW value.
ROWID: This is a base 64 string representing the unique address of a row in its table. This data type is primarily for values returned by the ROWID pseudocolumn.
UROWID: This is a base 64 string representing the logical address of a row of an index-organized table. The optional size is the size of a column of the UROWID type. The maximum size and default is 4,000 bytes.
BLOB: This is a binary large object.
CLOB: This is a character large object containing single-byte or multibyte characters. Both fixed-width and variable-width character sets are supported, and both use the CHAR database character set.

Specifying Data Types in Tables (continued)

NCLOB: This is a character large object containing Unicode characters. Both fixed-width and variable-width character sets are supported, and both use the NCHAR database character set. It stores national character set data.

Note: The maximum size for all LOB data types (BLOB, CLOB, and NCLOB) is:

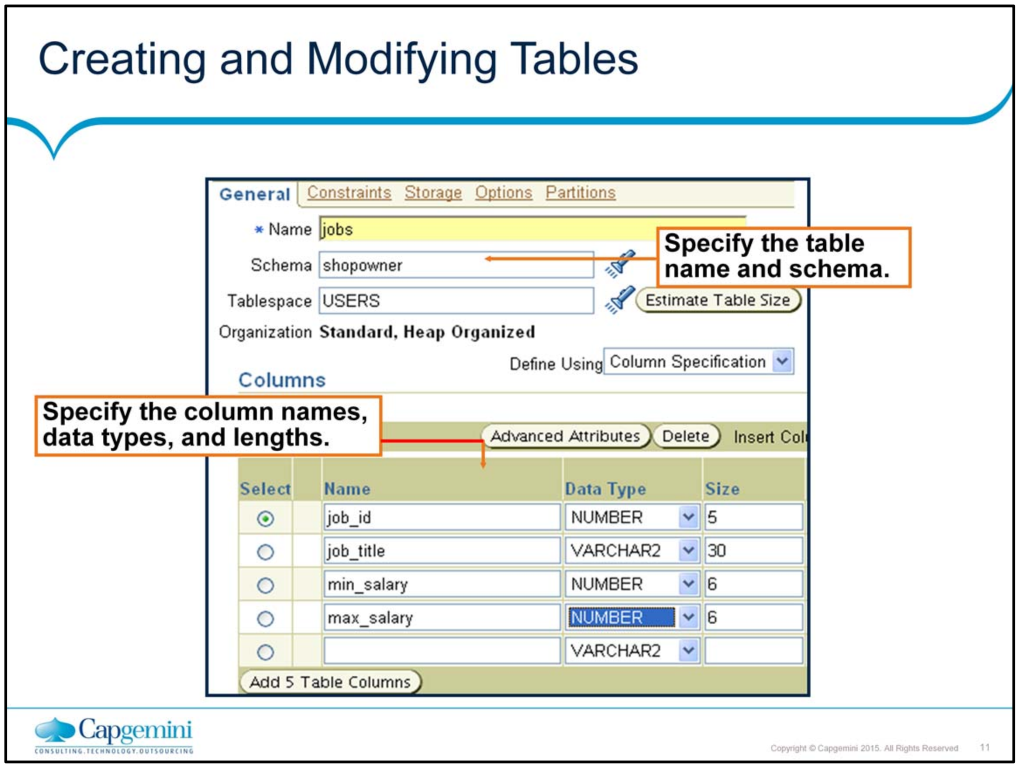
(4 gigabytes – 1) * (the value of CHUNK).

CHUNK is an optional attribute that you can set when defining a LOB. CHUNK specifies the number of bytes to be allocated for LOB manipulation. If the size is not a multiple of the database block size, then the database rounds up the size in bytes to the next multiple. For example, if the database block size is 2,048 and the CHUNK size is 2,050, then the database allocates 4,096 bytes (2 blocks). The maximum value is 32,768 (32 KB), which is the largest Oracle database block size allowed. The default CHUNK size is one Oracle database block.

BFILE: The BFILE data type contains a locator to a large binary file stored outside the database. It enables byte stream I/O access to external LOBs residing on the database server. The maximum size is 4 gigabytes.

TIMESTAMP(*fractional_seconds_precision*): With this data type, you can specify the year, month, and day values of date, as well as hour, minute, and second values of time, where *fractional_seconds_precision* is the number of digits in the fractional part of a second. The accepted values are 0 to 9. The default is 6.

For a complete list of built-in data types and user-defined types, refer to *Oracle Database SQL Reference*.



Creating and Modifying Tables

* Name	jobs
Schema	shopowner
Tablespace	USERS
Organization	**Standard, Heap Organized**
Define Using	Column Specification

Columns

Select	Name	Data Type	Size
●	job_id	NUMBER	5
○	job_title	VARCHAR2	30
○	min_salary	NUMBER	6
○	max_salary	NUMBER	6
○		VARCHAR2	

Specify the table name and schema.

Specify the column names, data types, and lengths.

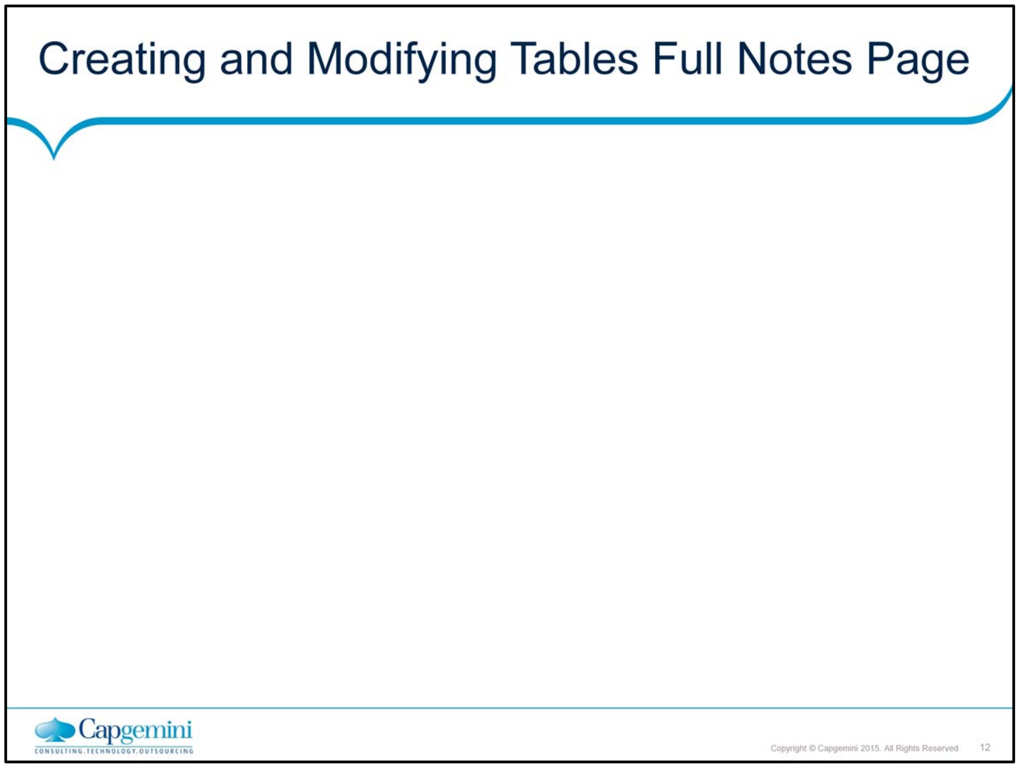
Creating and Modifying Tables

Tables are the basic units of data storage in an Oracle database. They hold all user-accessible data. Each table has columns and rows.

Creating a Table

To create a table by using Enterprise Manager, perform the following steps:

1. Click Tables in the Schema region of the Administration page. The Tables page appears.
2. If you know the schema name, enter all or part of it in the Schema field in the Search region. If you do not know the schema name, click the flashlight icon next to the Schema field. The Search and Select: Schema window appears. You can browse through the schema names and select the one that you are looking for.
3. Click Create. The Create Table: Table Organization page appears.
4. Accept the default of Standard, Heap Organized by clicking Continue. The Create Table page appears.
5. Enter the table name in the Name field.
6. Enter the schema name in the Schema field, or click the flashlight icon to invoke the search function.

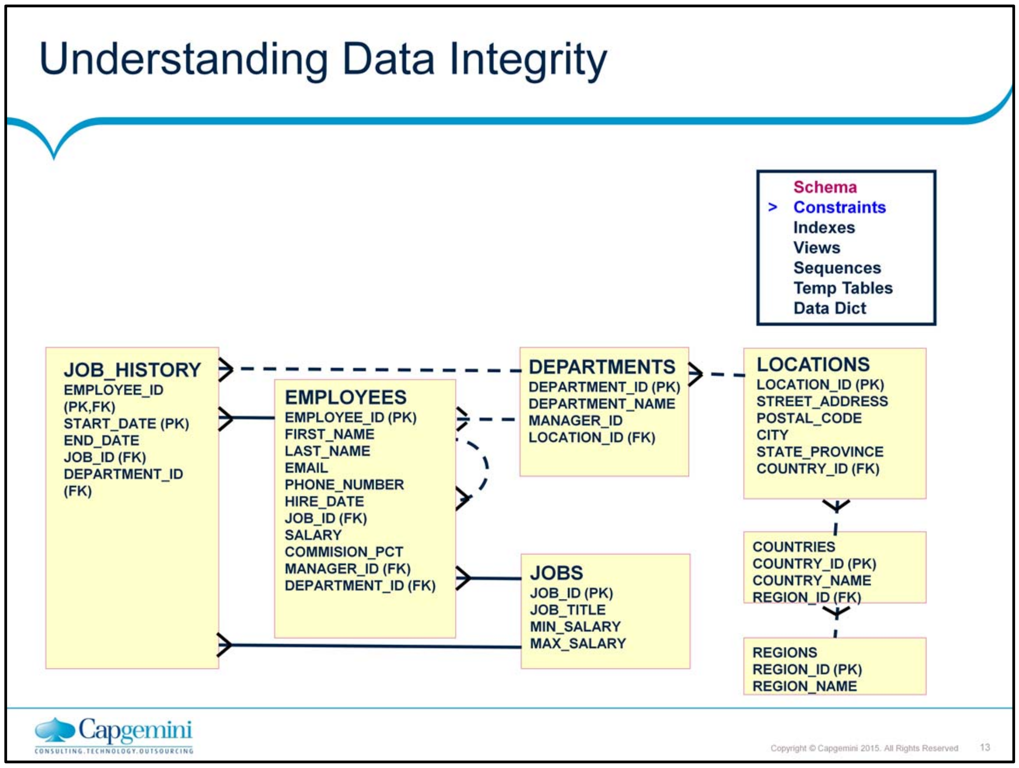


Creating and Modifying Tables Full Notes Page

Creating and Modifying Tables (continued)

7. Enter the tablespace name in the Tablespace field, or click the flashlight icon to invoke the search function.
8. In the Columns region, enter the column name and data types.
9. Click OK. An update message appears indicating that the table has been successfully created.

Modifying a Table
You can modify a table by using Enterprise Manager as described in the following steps. In this example, a column is added to a table.
On the Tables page, select the table in the results list and click Edit.
On the Edit Table page, click the Add 5 Table Columns button. An editable columns list appears.
Enter the new column name, data type, and size.
Click Apply. An update message appears indicating that the table has been modified successfully.



Understanding Data Integrity

Understanding Data Integrity

You can use the following integrity constraints to impose restrictions on the input of column values:

NOT NULL: By default, all columns in a table allow null values. Null means the absence of a value. A NOT NULL constraint requires that a column of a table must contain no null values. For example, you can define a NOT NULL constraint to require that a value be input in the LAST_NAME column for every row of the EMPLOYEES table.

UNIQUE Key: A UNIQUE key integrity constraint requires that every value in a column or set of columns (key) be unique—that is, no two rows of a table have duplicate values in a specified column or set of columns. For example, a UNIQUE key constraint is defined on the DEPARTMENT_NAME column of the DEPARTMENTS table to disallow rows with duplicate department names. Except for special circumstances, this is enforced with a unique index.

PRIMARY KEY: Each table in the database can have at most one PRIMARY KEY constraint. The values in the group of one or more columns subject to this constraint constitute the unique identifier of the row. In effect, each row is named by its primary key values.

Understanding Data Integrity (continued)

The Oracle server's implementation of the PRIMARY KEY integrity constraint guarantees that both the following are true:
No two rows of a table have duplicate values in the specified column or set of columns.
The primary key columns do not allow nulls. That is, a value must exist for the primary key columns in each row.

Under normal circumstances, the database enforces the PRIMARY KEY constraints by using indexes. The primary key constraint created for the DEPARTMENT_ID column in the DEPARTMENTS table is enforced by the implicit creation of:

A unique index on that column
A NOT NULL constraint for that column

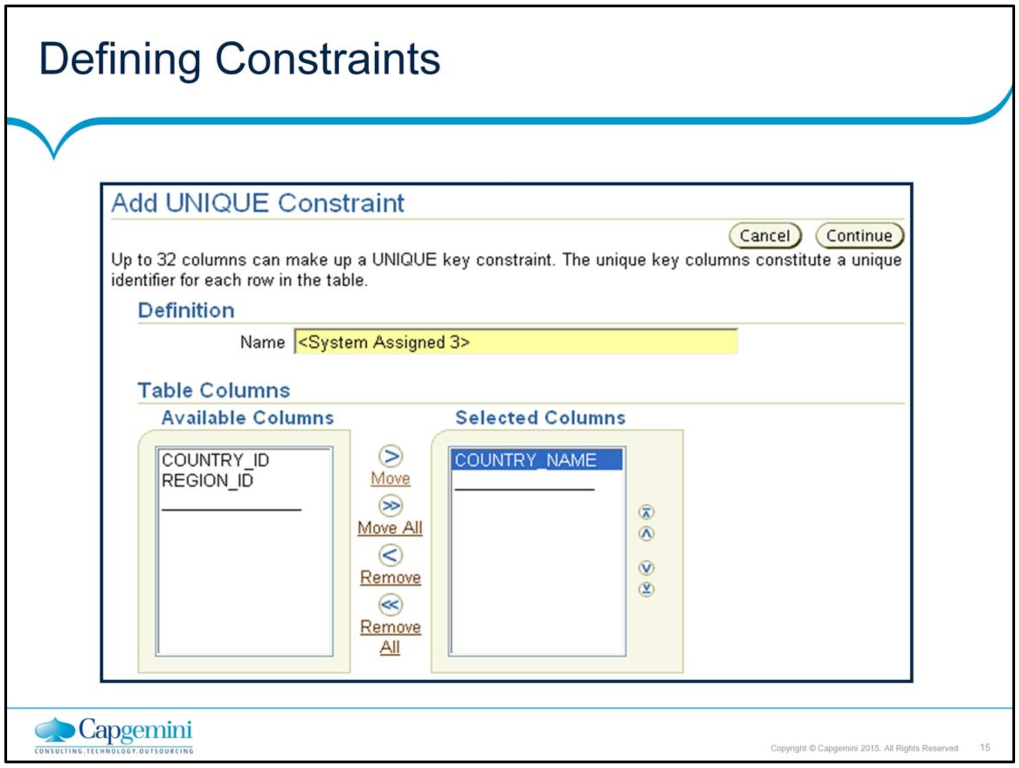
Referential integrity constraints: Different tables in a relational database can be related by common columns, and the rules that govern the relationship of the columns must be maintained. Referential integrity rules guarantee that these relationships are preserved.

A referential integrity constraint requires that for each row of a table, the value in the foreign key must match a value in a parent key.

As an example, a foreign key is defined on the DEPARTMENT_ID column of the EMPLOYEES table. It guarantees that every value in this column must match a value in the primary key of the DEPARTMENTS table. Therefore, no erroneous department numbers can exist in the DEPARTMENT_ID column of the DEPARTMENTS table.

Another type of referential integrity constraint is called a self-referential integrity constraint. This type of foreign key references a parent key in the same table

Check constraints: A CHECK integrity constraint on a column or set of columns requires that a specified condition be true or unknown for every row of the table. If a data manipulation language (DML) statement results in the condition of the CHECK constraint evaluating to false, then the statement is rolled back.



Defining Constraints

Add UNIQUE Constraint

Up to 32 columns can make up a UNIQUE key constraint. The unique key columns constitute a unique identifier for each row in the table.

Definition

Name <System Assigned 3>

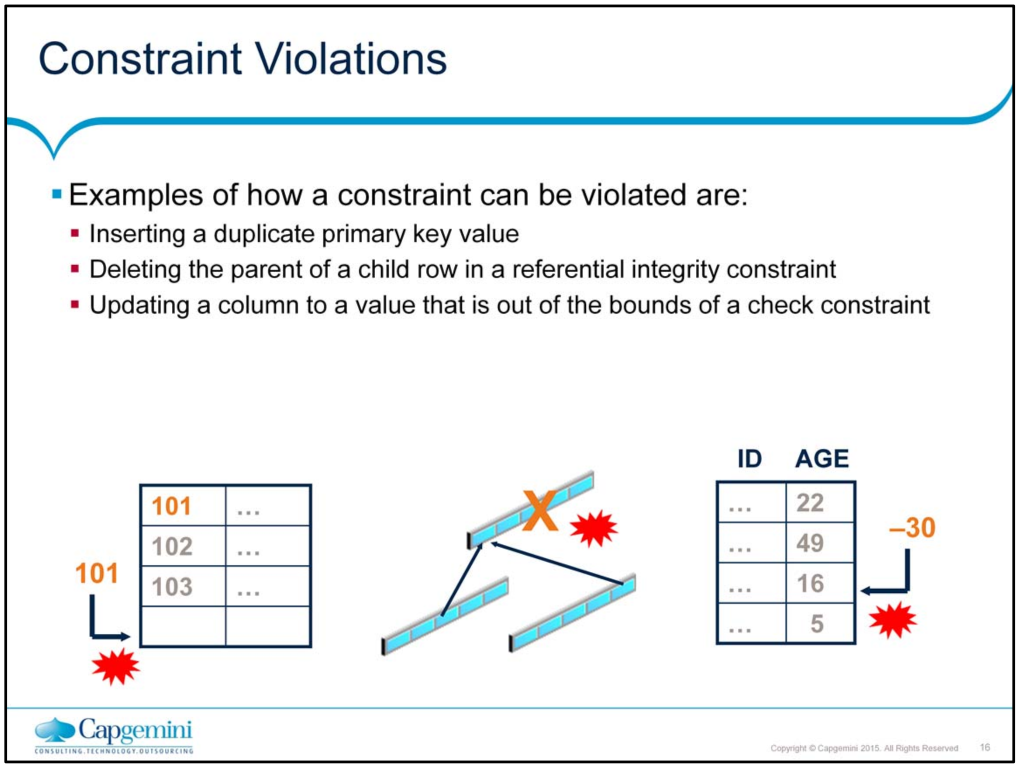
Table Columns

Available Columns	Selected Columns
COUNTRY_ID	COUNTRY_NAME
REGION_ID	

Defining Constraints

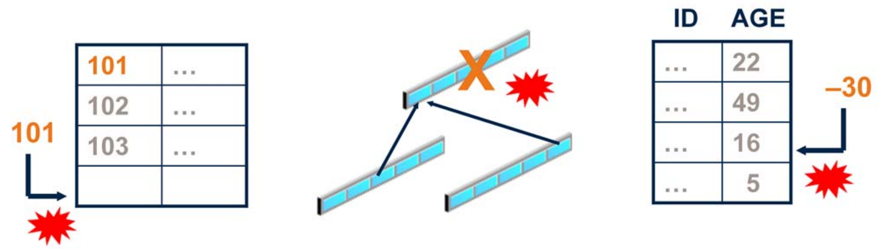
To add a constraint to a table by using Enterprise Manager, perform the following steps:

1. Select the table on the Tables page, and click Edit.
2. Click Constraints. The Constraints page is displayed showing all constraints that have been defined on the table.
3. Select the type of constraint that you want to add from the drop-down list, and click Add.
4. Enter the appropriate information for the type of constraint that you are defining. Click OK.



Constraint Violations

- Examples of how a constraint can be violated are:
  - Inserting a duplicate primary key value
  - Deleting the parent of a child row in a referential integrity constraint
  - Updating a column to a value that is out of the bounds of a check constraint



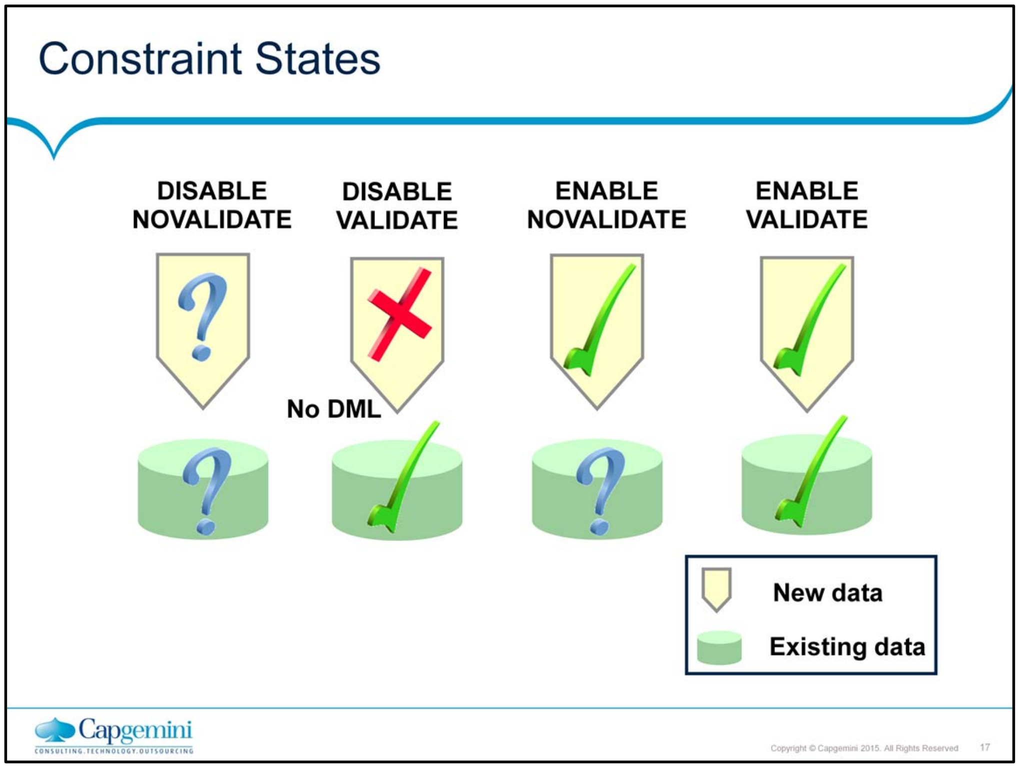
Constraint Violations

A constraint violation occurs when DML is submitted, which does not comply with the constraint. Constraint violations can come in many forms. Among these are the following:

Uniqueness: An attempt is made to have duplicate values in a column that has a unique constraint, such as in the case where a column is the primary key, or it is uniquely indexed.

Referential integrity: The rule of every child row having a parent row is violated.

Check: An attempt is made to store a value in a column that does not follow the rules defined for that column. For example, an AGE column could have a check constraint on it enforcing it to be a positive number.



Constraint States

Constraint States

To better deal with situations where data must be temporarily in violation of a constraint, you can designate a constraint to be in various states. An integrity constraint can be enabled (ENABLE) or disabled (DISABLE). If a constraint is enabled, the data is checked as it is entered or updated in the database. Data that does not conform to the constraint's rule is prevented from being entered. If a constraint is disabled, then the nonconforming data can be entered into the database. An integrity constraint can be in one of the following states:

DISABLE NOVALIDATE
DISABLE VALIDATE
ENABLE NOVALIDATE
ENABLE VALIDATE

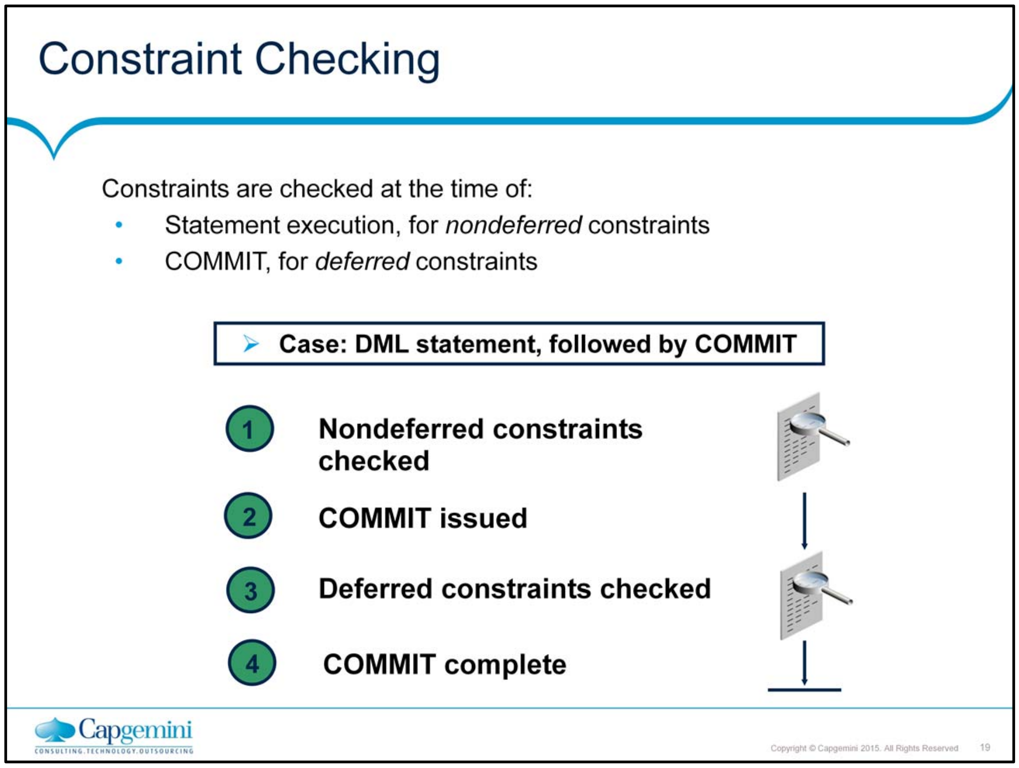
Constraint States (continued)

DISABLE NOVALIDATE: New as well as existing data may not conform to the constraint because it is not checked. This is often used when the data is from an already validated source and the table is read-only, so no new data is being entered into the table.

DISABLE VALIDATE: If a constraint is in this state, then any modification of the constrained columns is not allowed because it would be inconsistent to have validated the existing data and then allow unchecked data to enter the table. This is often used when the existing data must be validated but the data is not going to be modified and the index is not otherwise needed for performance.

ENABLE NOVALIDATE: New data conforms to the constraint but existing data is in an unknown state. This is frequently used so that existing constraint violations can be corrected, and at the same time, new violations are not allowed to enter the system.

ENABLE VALIDATE: Both new and existing data conform to the constraint. This is the typical and default state of a constraint.



Constraint Checking

Constraints are checked at the time of:
- Statement execution, for *nondeferred* constraints
- COMMIT, for *deferred* constraints

Case: DML statement, followed by COMMIT

1. **Nondeferred constraints checked**
2. **COMMIT issued**
3. **Deferred constraints checked**
4. **COMMIT complete**

Constraint Checking

You can defer checking constraints for validity until the end of the transaction.

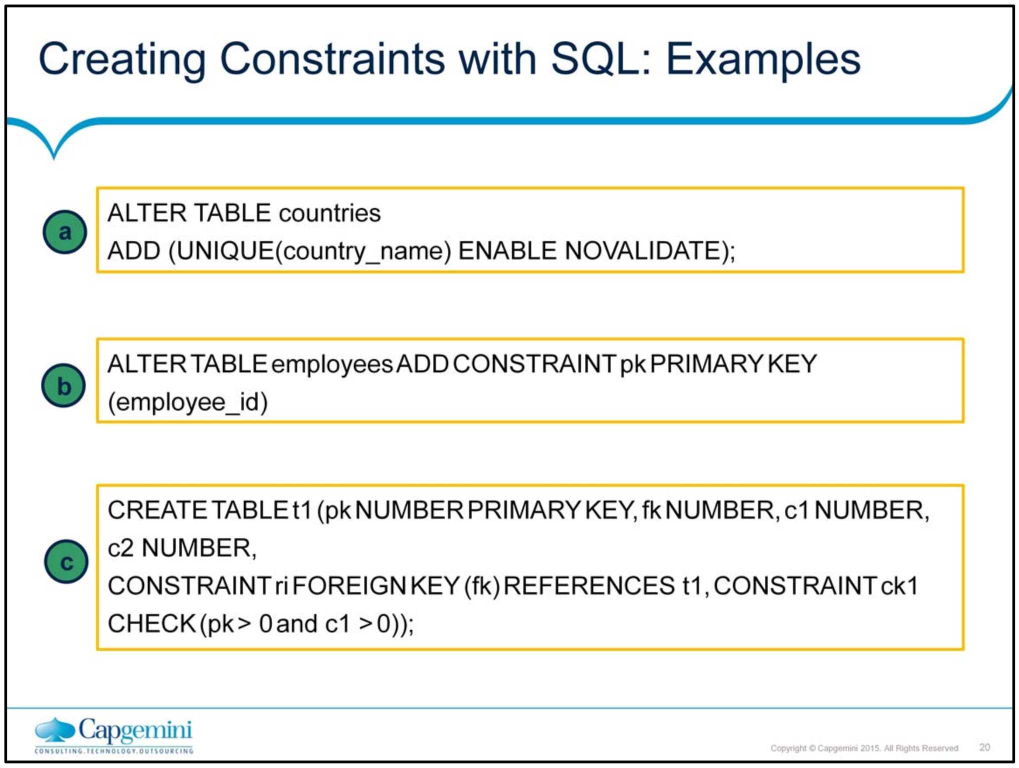
Nondeferred constraints, also known as immediate constraints, are enforced at the end of every DML statement. A constraint violation causes the statement to be rolled back. If a constraint causes an action such as delete cascade, the action is taken as part of the statement that caused it. A constraint that is defined as nondeferrable cannot be changed to a deferrable constraint.

Deferred constraints are constraints that are checked only when a transaction is committed. If any constraint violations are detected at commit time, then the entire transaction is rolled back. These constraints are most useful when both the parent and child rows in a foreign key relationship are entered at the same time, as in the case of an order entry system, where the order and the items in the order are entered at the same time.

A constraint that is defined as deferrable can be specified as one of the following:

Initially immediate specifies that by default it must function as an immediate constraint unless explicitly set otherwise.

Initially deferred specifies that by default the constraint must be enforced only at the end of the transaction.



Creating Constraints with SQL: Examples

a
```
ALTER TABLE countries
ADD (UNIQUE(country_name) ENABLE NOVALIDATE);
```

b
```
ALTER TABLE employees ADD CONSTRAINT pk PRIMARY KEY
(employee_id)
```

c
```
CREATE TABLE t1 (pk NUMBER PRIMARY KEY, fk NUMBER, c1 NUMBER,
c2 NUMBER,
CONSTRAINT ri FOREIGN KEY (fk) REFERENCES t1, CONSTRAINT ck1
CHECK (pk > 0 and c1 > 0));
```

Creating Constraints with SQL: Examples

Three examples of constraint creation are shown in the slide:

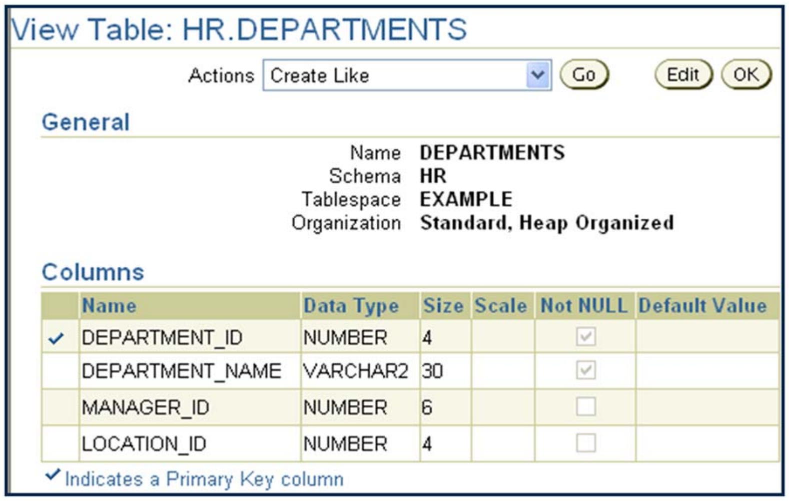
a. After this statement executes, any inserts or updates done on the COUNTRIES table are required to have a COUNTRY_NAME value that is unique. But it is possible that when this statement is issued, there already exist COUNTRY_NAME values in the table that are not unique. The NOVALIDATE keyword indicates that they should be ignored. Only new rows are constrained.

b. This statement adds a primary key to the employee table. The constraint name is PK and the primary key is the EMPLOYEE_ID column.

c. This statement defines constraints at the time the table is created, rather than using an ALTER TABLE statement later. The RI constraint enforces that the values in the FK column must be present in the primary key column of the T1 table. The CK1 constraint enforces that the PK and C1 columns are greater than zero.

Note: Each constraint has a name. If one is not supplied in the DDL statement, then a system-supplied name is assigned, which starts with SYS_.

Viewing the Columns in a Table



View Table: HR.DEPARTMENTS

Actions Create Like Go Edit OK

General

Name	**DEPARTMENTS**
Schema	**HR**
Tablespace	**EXAMPLE**
Organization	**Standard, Heap Organized**

Columns

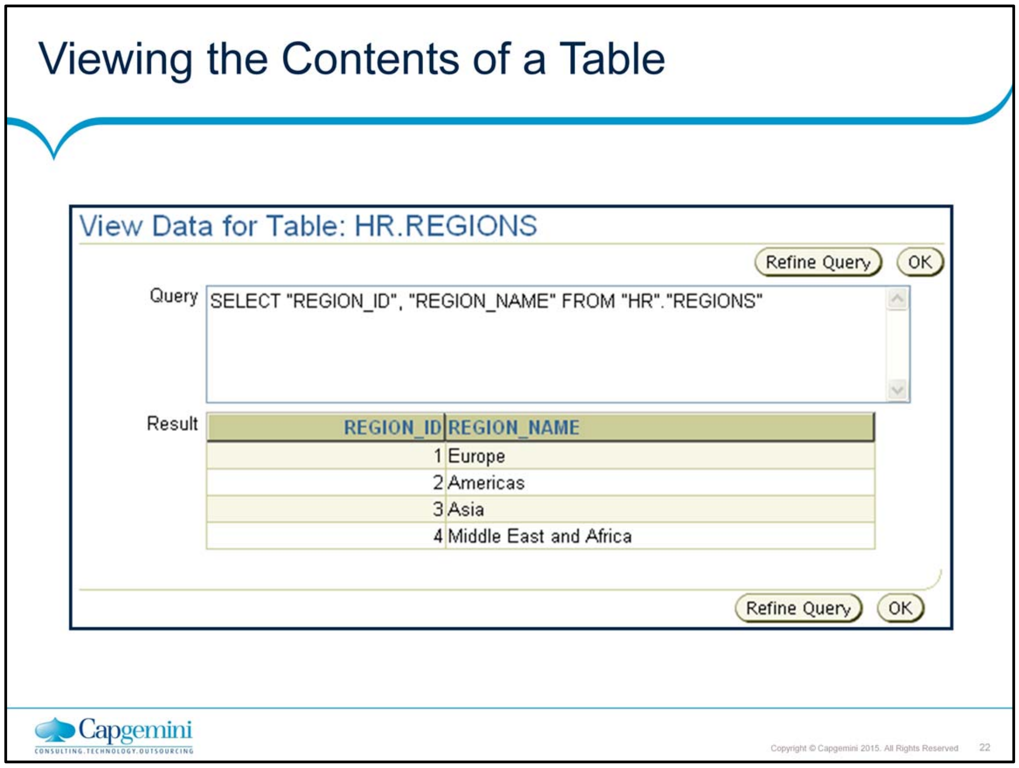
	Name	Data Type	Size	Scale	Not NULL	Default Value
✓	DEPARTMENT_ID	NUMBER	4		✓	
	DEPARTMENT_NAME	VARCHAR2	30		✓	
	MANAGER_ID	NUMBER	6		☐	
	LOCATION_ID	NUMBER	4		☐	

✓ Indicates a Primary Key column

Viewing the Columns in a Table

To view the attributes of a table by using Enterprise Manager, perform the following steps:

1. Click the Tables link in the Schema region of the Database Administration page.
2. Select a table from the Results list and click the View button to see the attributes of the table.



Viewing the Contents of a Table

View Data for Table: HR.REGIONS

Query: SELECT "REGION_ID", "REGION_NAME" FROM "HR"."REGIONS"

Result

REGION_ID	REGION_NAME
1	Europe
2	Americas
3	Asia
4	Middle East and Africa

Viewing the Contents of a Table

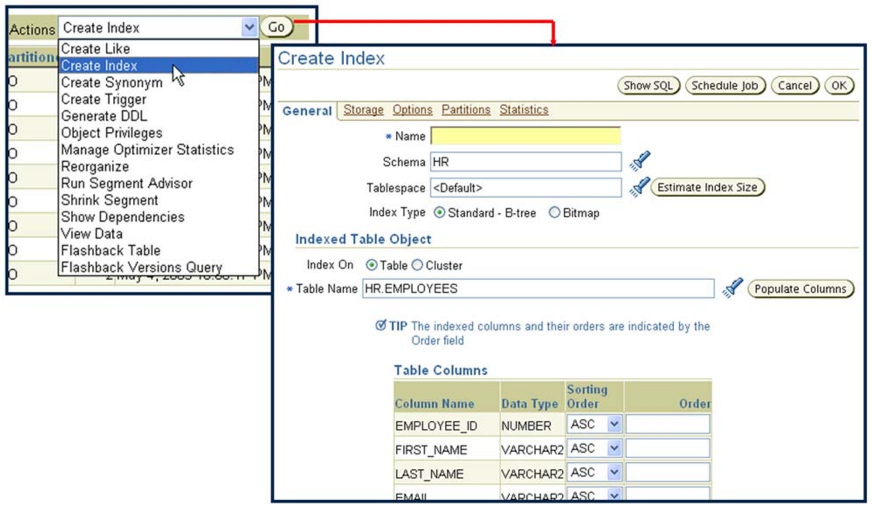
To view the rows in a table by using Enterprise Manager, perform the following steps:

1. Select the table on the Tables page.
2. Select View Data from the Actions menu and click Go.

The View Data for Table page appears. The row data for the table is shown in the Result region. The Query box displays the SQL query that is executed to produce the results. On this page, you can click any column name and sort the data in the column in either ascending or descending order. If you want to change the query, click the Refine Query button. On the Refine Query for Table page, you can select the columns that you want to display and specify a WHERE clause for the SQL statement to limit the results.

For more information about the WHERE clauses in SQL statements, refer to Oracle Database SQL Reference.

Actions with Tables



Create Index

* Name

Schema HR

Tablespace <Default>

Index Type [✓ Standard - B-tree] Bitmap

Indexed Table Object

Index On [✓ Table] Cluster

* Table Name HR.EMPLOYEES

TIP The indexed columns and their orders are indicated by the Order field

Table Columns

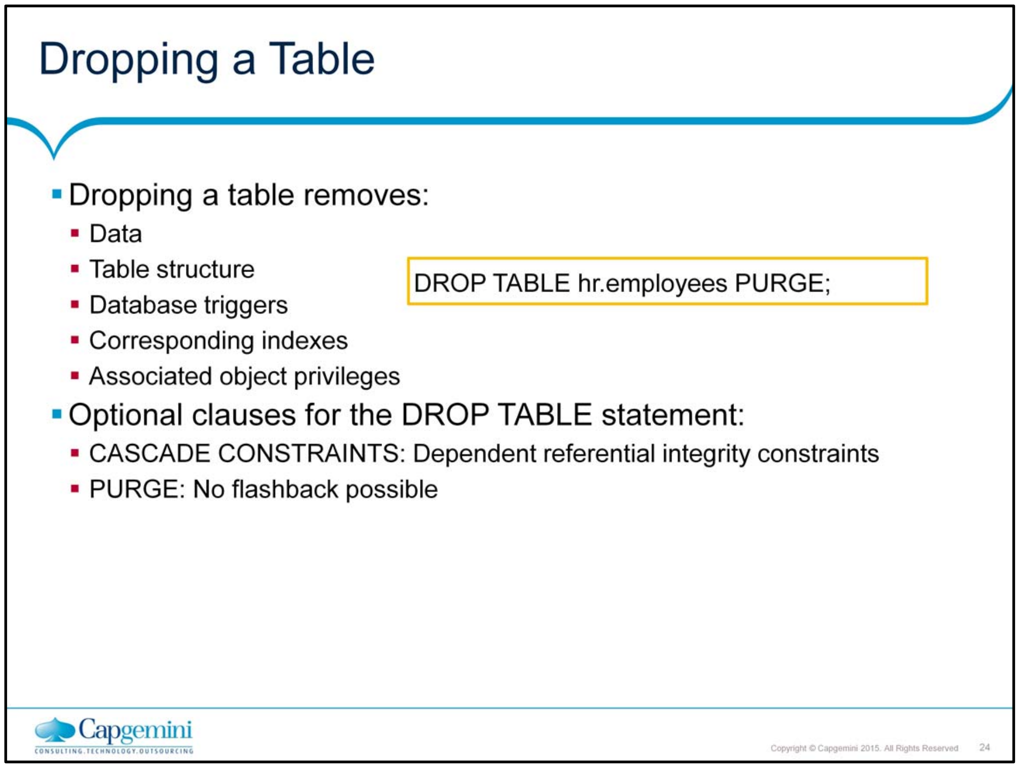
Column Name	Data Type	Sorting Order	Order
EMPLOYEE_ID	NUMBER	ASC	
FIRST_NAME	VARCHAR2	ASC	
LAST_NAME	VARCHAR2	ASC	
EMAIL	VARCHAR2	ASC	

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Actions with Tables

You can select a table and then perform actions on that table. Here are some of those actions:

- Create Like: With this action, you can create a table that has the same structure as the selected table. You must change the constraint names. You can add or delete columns and make other changes to the table structure before it is created.
- Create Index: Use this option to create indexes on a table.
- Generate DDL: This generates the DDL that represents the table as it already exists. This can then be copied to a text file for use as a script or for documentation purposes.
- Grant Privileges: By default, when a table is created, only the owner can do anything with it. The owner must grant privileges to other users in order for them to perform DML or possibly DDL on the table.
- Show Dependencies: This shows objects that this table depends on or objects that depend on this table.
- View Data: This selects and displays data from the table in a read-only manner.



Dropping a Table

- Dropping a table removes:
  - Data
  - Table structure
  - Database triggers
  - Corresponding indexes
  - Associated object privileges
- Optional clauses for the DROP TABLE statement:
  - CASCADE CONSTRAINTS: Dependent referential integrity constraints
  - PURGE: No flashback possible

```
DROP TABLE hr.employees PURGE;
```

Dropping a Table

Syntax:

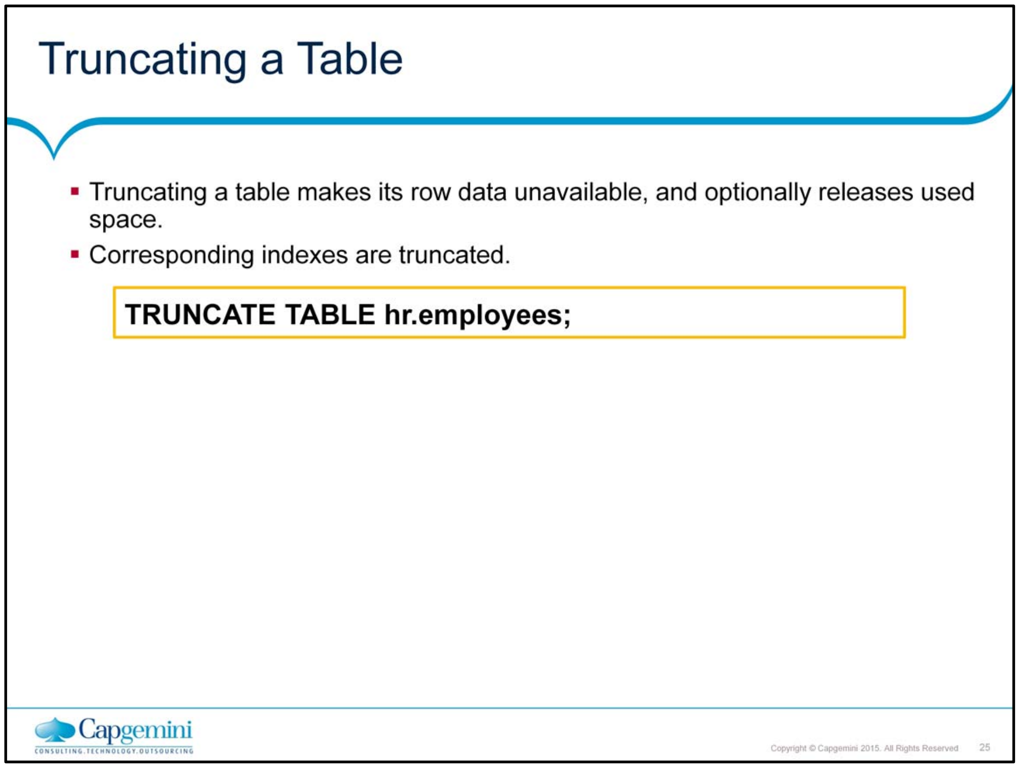
DROP TABLE [schema.] table [CASCADE CONSTRAINTS] [PURGE]

The DROP TABLE command removes data, the table structure, and associated object privileges. Some DROP TABLE considerations are as follows:

Without the PURGE clause, the table definition, associated indexes, and triggers are placed in a recycle bin. The table data still exists, but is inaccessible without the table definition. If you drop a table through Enterprise Manager, the PURGE clause is not used. Use the FLASHBACK TABLE command to recover schema objects from the recycle bin. The PURGE RECYCLEBIN command empties the recycle bin.

The CASCADE CONSTRAINTS option is required to remove all dependent referential integrity constraints.

Note: If you do not use the PURGE option, the space taken up by the table and its indexes still counts against the user’s allowed quota for the tablespaces involved. That is, the space is still considered as being used.



Truncating a Table

- Truncating a table makes its row data unavailable, and optionally releases used space.
- Corresponding indexes are truncated.

```
TRUNCATE TABLE hr.employees;
```

Truncating a Table

Syntax:

TRUNCATE TABLE [schema.] table [{DROP | REUSE} STORAGE]

The effects of using this command are as follows:

The table is marked as empty by setting the high-water mark (HWM) to the beginning of the table, making its rows unavailable.

No undo data is generated and the command commits implicitly because TRUNCATE TABLE is a DDL command.

Corresponding indexes are also truncated.

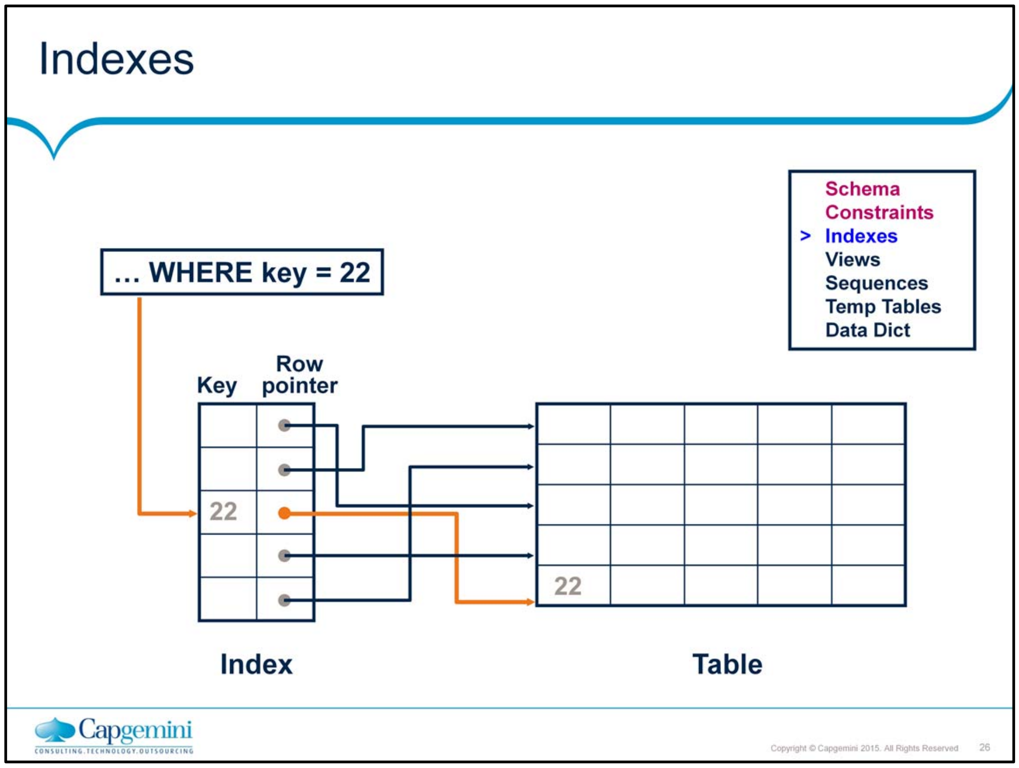
A table that is being referenced by a foreign key cannot be truncated.

The delete triggers do not fire when this command is used.

This is usually many times faster than issuing a DELETE statement to delete all the rows of the table due to the following reasons:

The Oracle database resets the table’s HWM instead of processing each row as a DELETE operation.

No undo data is generated.

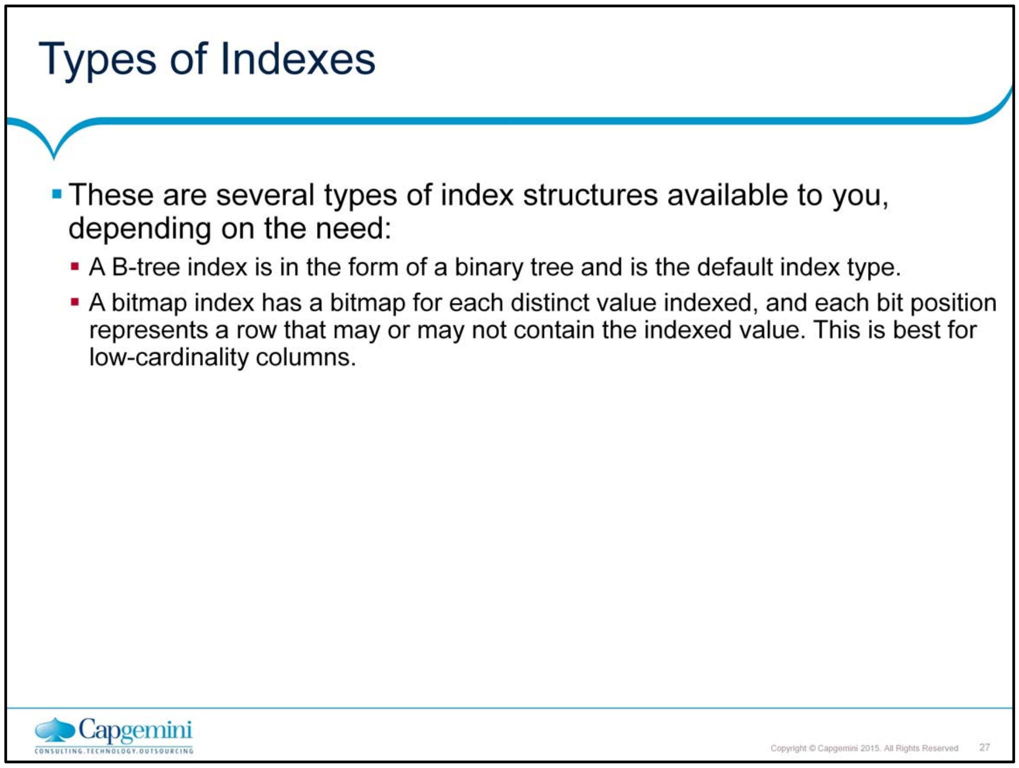


Indexes

Indexes

Indexes are optional structures associated with tables. They can be created to improve the performance of data update and retrieval. An Oracle index provides a direct access path to a row of data.

Indexes can be created on one or more columns of a table. After an index is created, it is automatically maintained and used by the Oracle server. Updates to a table's data, such as adding new rows, updating rows, or deleting rows, are automatically propagated to all relevant indexes with complete transparency to users.



Types of Indexes

- These are several types of index structures available to you, depending on the need:
  - A B-tree index is in the form of a binary tree and is the default index type.
  - A bitmap index has a bitmap for each distinct value indexed, and each bit position represents a row that may or may not contain the indexed value. This is best for low-cardinality columns.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Types of Indexes

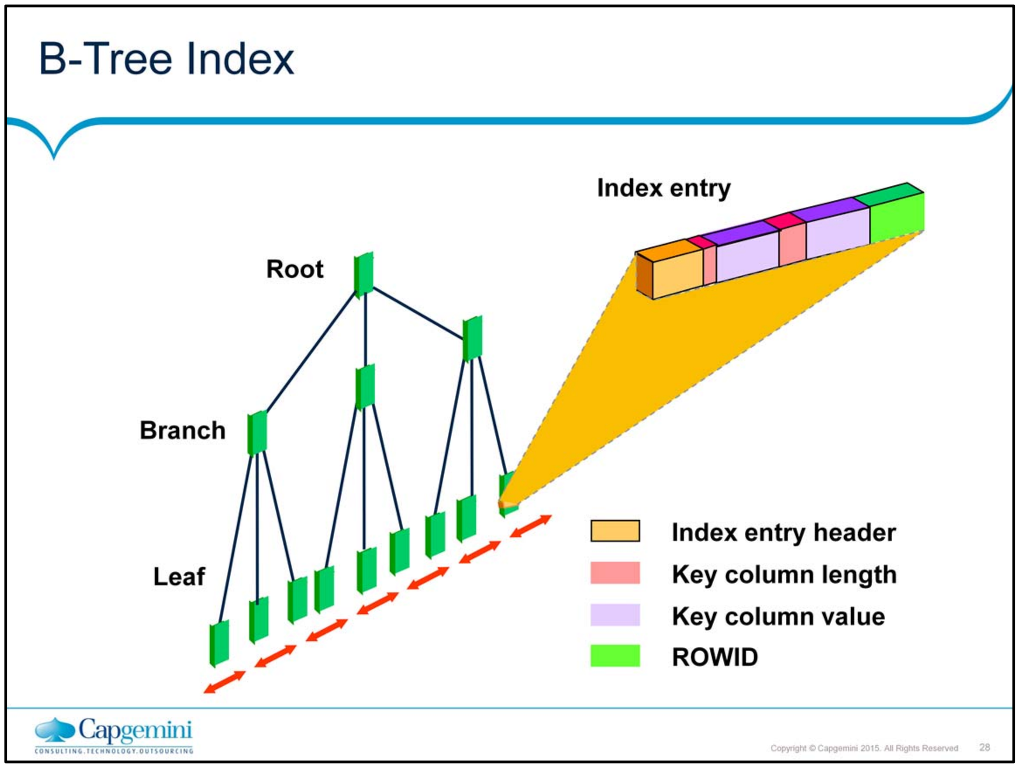
The following are the most common forms of indexes:

B-tree

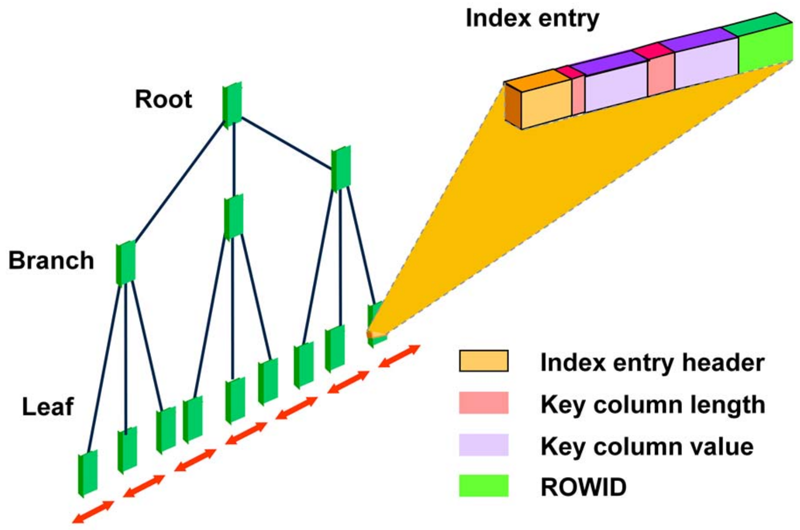
Bitmap

A B-tree index has its key values stored in a balanced tree (B-tree), allowing for fast binary searches.

A bitmap index has a bitmap for each distinct key value being indexed. Within each bitmap, there is a bit set aside for each row in the table being indexed. This allows for fast lookups when there are few distinct values; that is, the indexed column has low cardinality. An example of this is a gender indicator. It can have values of “M” and “F” only. So, there are only two bitmaps to search. For example, if a bitmap index were used for a phone_number column, there would be so many bitmaps to manage and search that it would be very inefficient. Use bitmap indexes for low-cardinality columns.



B-Tree Index



🟧 Index entry header
🟥 Key column length
🟪 Key column value
🟩 ROWID

B-Tree Index

Structure of a B-tree index

At the top of the index is the root, which contains entries that point to the next level in the index. At the next level are branch blocks, which in turn point to blocks at the next level in the index. At the lowest level are the leaf nodes, which contain the index entries that point to rows in the table. The leaf blocks are doubly linked to facilitate the scanning of the index in an ascending as well as descending order of key values.

Format of index leaf entries

An index entry is made up of the following components:

- An entry header, which stores the number of columns and locking information
- Key column length-value pairs, which define the size of a column in the key followed by the value for the column (The number of such pairs is a maximum of the number of columns in the index.)
- ROWID of a row that contains the key values

B-Tree Index (continued)

Index leaf entry characteristics

In a B-tree index on a nonpartitioned table:

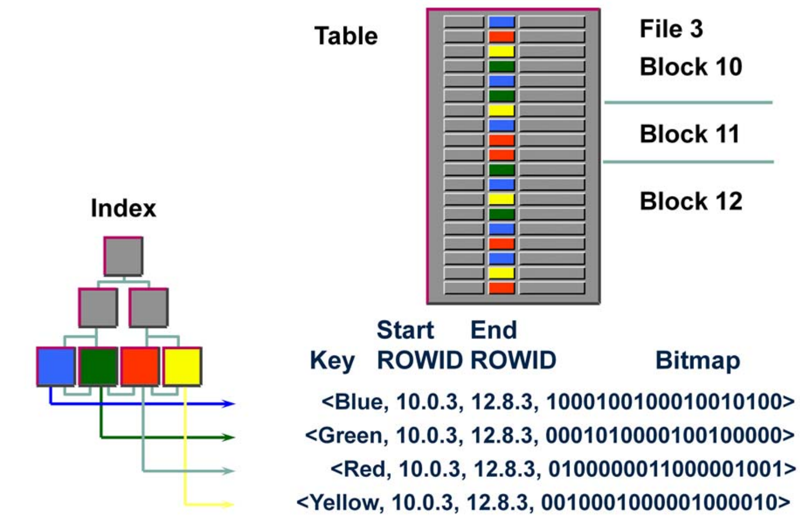
- Key values are repeated if there are multiple rows that have the same key value unless the index is compressed
- There is no index entry corresponding to a row that has all key columns that are NULL. Therefore, a WHERE clause specifying NULL will always result in a full table scan.
- Restricted ROWID is used to point to the rows of the table because all rows belong to the same segment

Effect of DML operations on an index

The Oracle server maintains all the indexes when DML operations are carried out on the table. Here is an explanation of the effect of a DML command on an index:

- Insert operations result in the insertion of an index entry in the appropriate block.
- Deleting a row results only in a logical deletion of the index entry. The space used by the deleted row is not available for new entries until all the entries in the block are deleted.
- Updates to the key columns result in a logical delete and an insert to the index. The PCTFREE setting has no effect on the index except at the time of creation. A new entry may be added to an index block even if it has less space than that specified by PCTFREE.

Bitmap Indexes



Key	Start ROWID	End ROWID	Bitmap
<Blue,	10.0.3,	12.8.3,	1000100100010010100>
<Green,	10.0.3,	12.8.3,	0001010000100100000>
<Red,	10.0.3,	12.8.3,	0100000011000001001>
<Yellow,	10.0.3,	12.8.3,	0010001000001000010>

Bitmap Indexes

Bitmap indexes are more advantageous than B-tree indexes in certain situations:

When a table has millions of rows and the key columns have low cardinality—that is, there are very few distinct values for the column. For example, bitmap indexes may be preferable to B-tree indexes for the gender and marital status columns of a table containing passport records.

When queries often use a combination of multiple WHERE conditions involving the OR operator

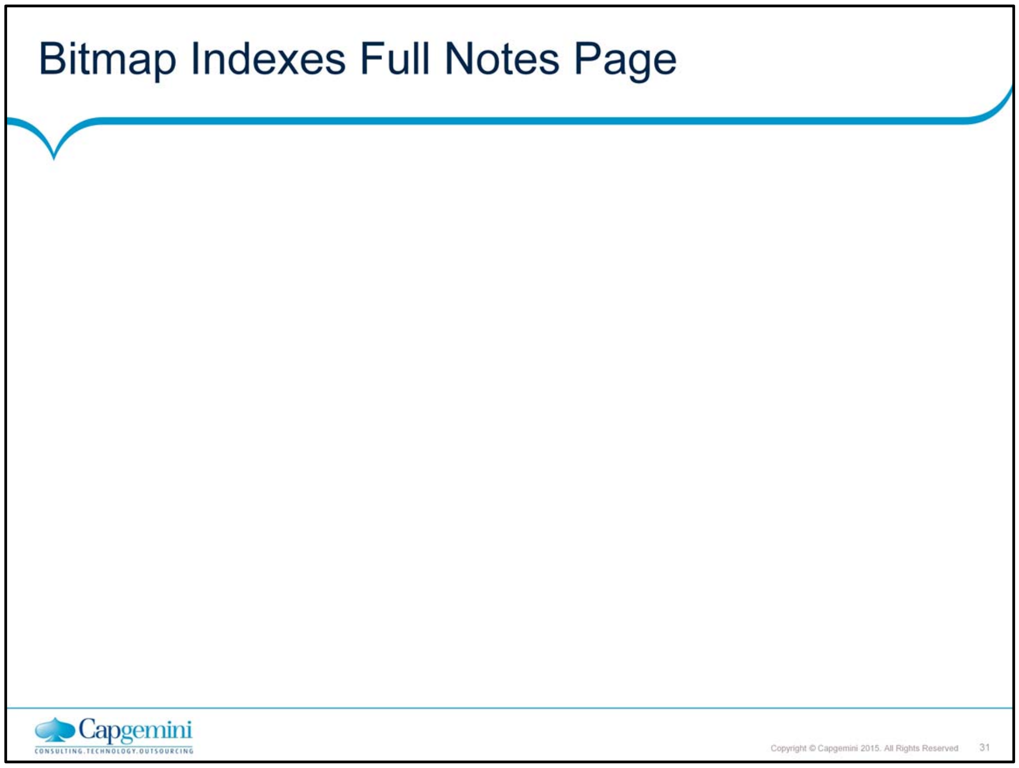
When there is read-only or low update activity on the key columns

Structure of a bitmap index

A bitmap index is also organized as a B-tree, but the leaf node stores a bitmap for each key value instead of a list of ROWIDs. Each bit in the bitmap corresponds to a possible ROWID, and if the bit is set, it means that the row with the corresponding ROWID contains the key value.

As shown in the diagram, the leaf node of a bitmap index contains the following:

An entry header that contains the number of columns and lock information



Bitmap Indexes Full Notes Page

Bitmap Indexes (continued)

Structure of a bitmap index (continued)

Key values consisting of length and value pairs for each key column. In the example, the key consists of only one column, and the first entry has a key value of Blue.

Start ROWID, which in the example specifies block number ten, row number zero, and file number three

End ROWID, which in the example specifies block number twelve, row
number eight, and file number three

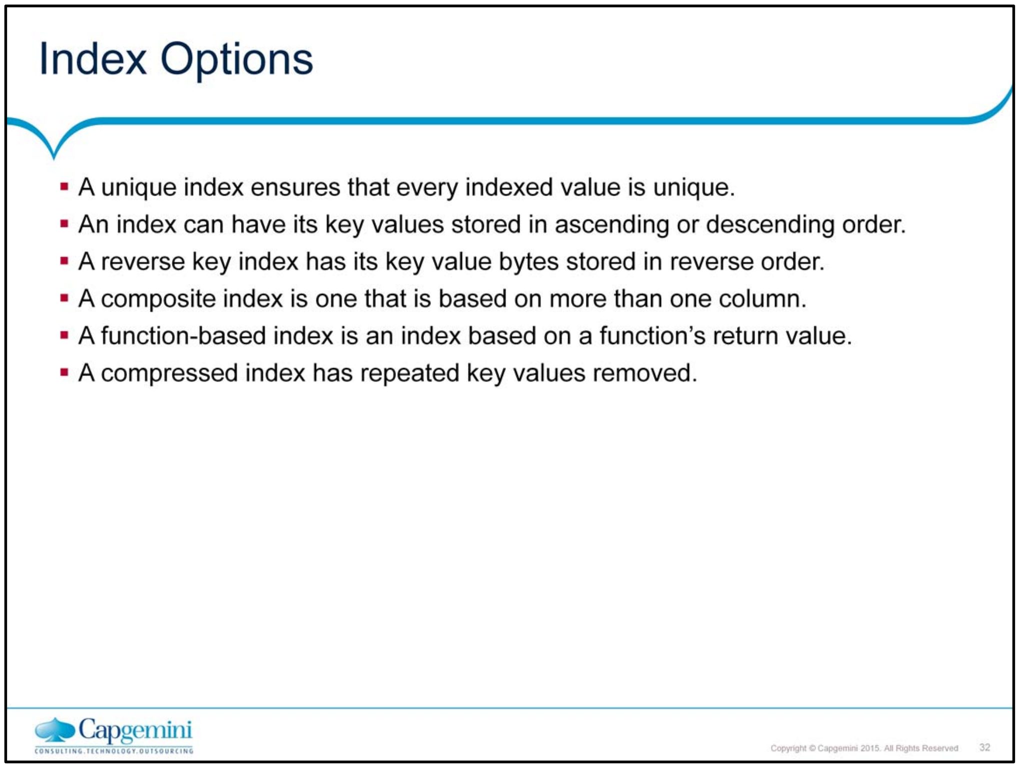
A bitmap segment consisting of a string of bits. (The bit is set when the corresponding row contains the key value and is unset when the row does not contain the key value. The Oracle server uses a patented compression technique to store bitmap segments.)

The start ROWID is the ROWID of the first row pointed to by the bitmap segment of the bitmap—that is, the first bit of the bitmap corresponds to that ROWID, the second bit of the bitmap corresponds to the next row in the block, and the end ROWID is a pointer to the last row in the table covered by the bitmap segment. Bitmap indexes use restricted ROWIDs.

Using a bitmap index

The B-tree is used to locate the leaf nodes that contain bitmap segments for a given value of the key. Start ROWID and the bitmap segments are used to locate the rows that contain the key value.

When changes are made to the key column in the table, bitmaps must be modified. This results in the locking of the relevant bitmap segments. Because locks are acquired on the whole bitmap segment, a row that is covered by the bitmap cannot be updated by other transactions until the first transaction ends.



Index Options

- A unique index ensures that every indexed value is unique.
- An index can have its key values stored in ascending or descending order.
- A reverse key index has its key value bytes stored in reverse order.
- A composite index is one that is based on more than one column.
- A function-based index is an index based on a function’s return value.
- A compressed index has repeated key values removed.

Index Options

For efficiency of retrieval, it may be advantageous to have an index store the keys in descending order. This decision is made on the basis of how the data is accessed most frequently.

A reverse key index has the bytes of the indexed value stored in reverse order. This can reduce activity in a particular hot spot in the index. If many users are processing data in the same order, then the prefix portions of the key values (that are currently being processed) are close in value at any given instant. Consequently, there is a lot of activity in that area of the index structure. A reverse key index spreads that activity out across the index structure by indexing a reversed-byte version of the key values.

An index created by the combination of more than one column is called a composite index. For example, you can create an index based on a person’s last name and first name:

```
CREATE INDEX name_ix ON employees
(last_name, first_name);
```

Index Options (continued)

A function-based index indexes a function's return value. This function can be a built-in SQL function, a supplied PL/SQL function, or a user-written function. This relieves the server from having to invoke the function for every key value as it performs a search on the indexed expression. The following example indexes the returned tree volume that is computed by the function, based on each tree's species, height, and volume (which are columns in the TREES table):

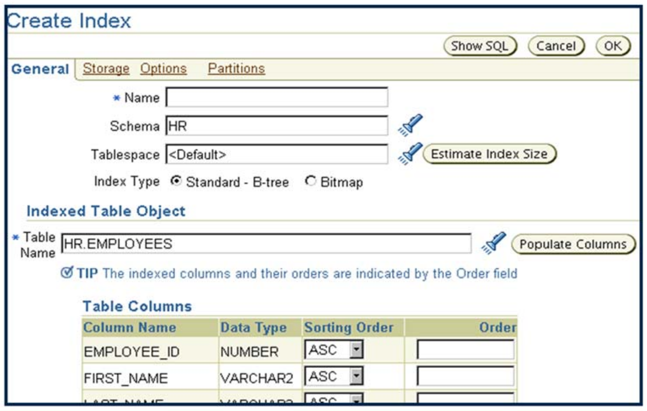
```
CREATE INDEX tree_vol_ix ON
TREES(volume(species,height,circumferenc
e));
```

Then, any query that contains the expression volume(species,height,circumference) in the WHERE clause may be able to take advantage of this index, and execute much more quickly because the volume computation is already done for each tree.

Function-based indexes are maintained automatically, as are normal indexes.

You can use a compressed index to reduce disk consumption at execution time. Because repeated key values are removed, more index entries can fit in a given amount of disk space, resulting in the ability to read more entries from the disk in the same amount of time. Compression and decompression must be performed for the writing and reading of the index, respectively.

Creating Indexes


Create Index

* Name

Schema HR

Tablespace <Default>

Index Type [✓ Standard - B-tree] Bitmap

Indexed Table Object

* Table Name HR.EMPLOYEES

TIP The indexed columns and their orders are indicated by the Order field

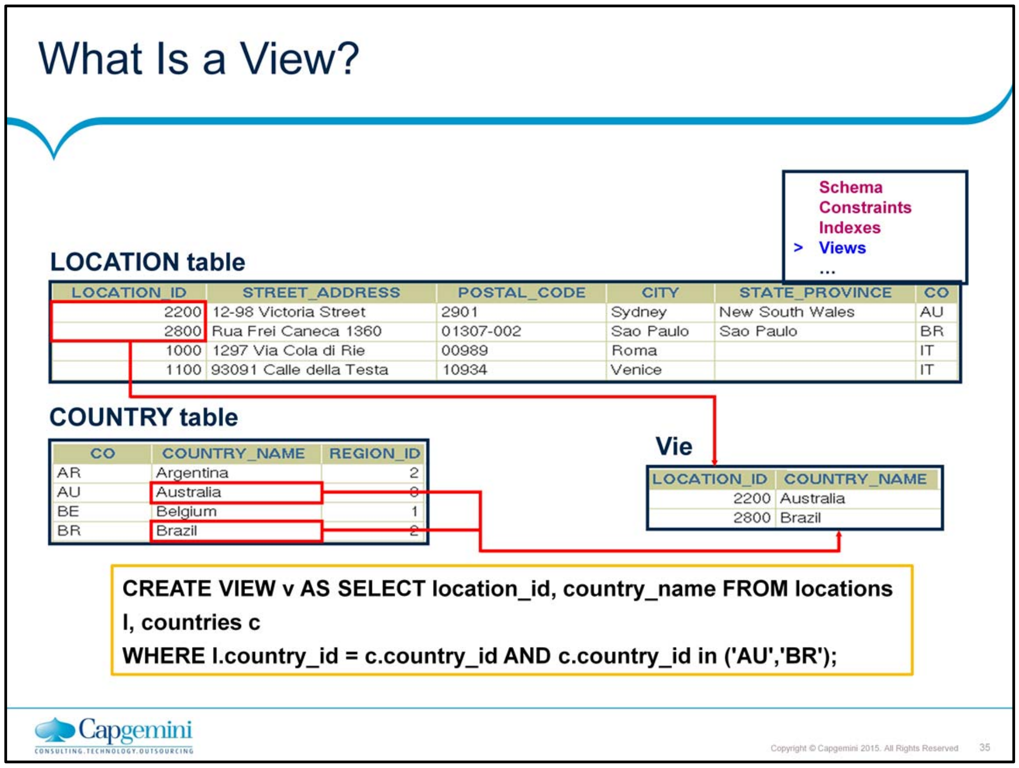
Table Columns

Column Name	Data Type	Sorting Order	Order
EMPLOYEE_ID	NUMBER	ASC	
FIRST_NAME	VARCHAR2	ASC	

```
CREATE INDEX my_index ON
employees(last_name, first_name);
```

Creating Indexes

You can click the Indexes link under the Schema heading of the Administration page to view the Indexes page. You can view index attributes or use the Actions menu to view dependencies for an index. Indexes can be created explicitly, or implicitly through constraints that are placed on a table. An example of an implicitly created index is the definition of a primary key, in which case a unique index would be automatically created to enforce uniqueness on the column.



What Is a View?

Schema
Constraints
Indexes
> Views
...

LOCATION table

LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE_PROVINCE	CO
2200	12-98 Victoria Street	2901	Sydney	New South Wales	AU
2800	Rua Frei Caneca 1360	01307-002	Sao Paulo	Sao Paulo	BR
1000	1297 Via Cola di Rie	00989	Roma		IT
1100	93091 Calle della Testa	10934	Venice		IT

COUNTRY table

CO	COUNTRY_NAME	REGION_ID
AR	Argentina	2
AU	Australia	3
BE	Belgium	1
BR	Brazil	2

Vie

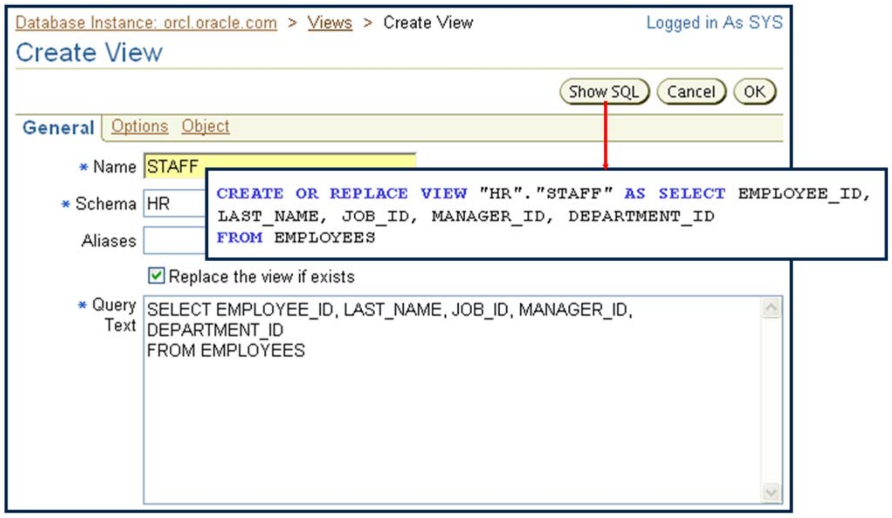
LOCATION_ID	COUNTRY_NAME
2200	Australia
2800	Brazil

```
CREATE VIEW v AS SELECT location_id, country_name FROM locations
l, countries c
WHERE l.country_id = c.country_id AND c.country_id in ('AU','BR');
```

What Is a View?

Views are customized representations of data in one or more tables or other views. They can be thought of as stored queries because they can hide very complex conditions, joins, and other complex expressions and SQL constructs. Views do not actually contain data; instead, they derive their data from the tables on which they are based. These tables are referred to as the base tables of the view.

Creating Views



Name: STAFF
Schema: HR
Aliases:
[✓] Replace the view if exists
Query Text:

```
SELECT EMPLOYEE_ID, LAST_NAME, JOB_ID, MANAGER_ID,
DEPARTMENT_ID
FROM EMPLOYEES
```

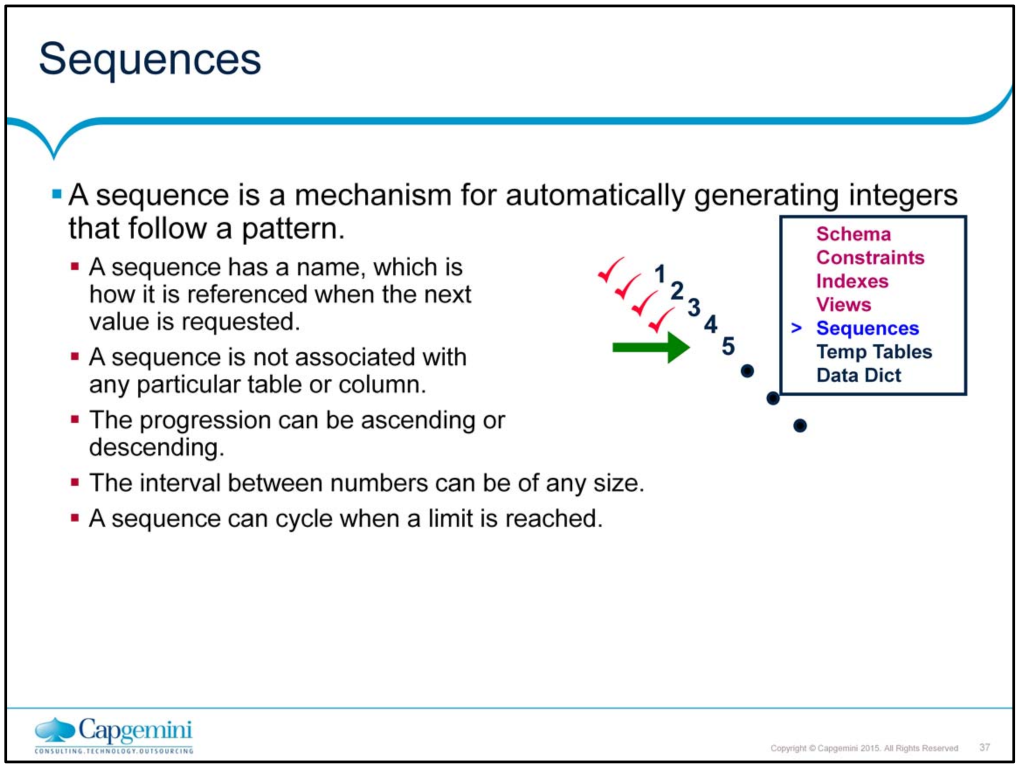
```
CREATE OR REPLACE VIEW "HR"."STAFF" AS SELECT EMPLOYEE_ID,
LAST_NAME, JOB_ID, MANAGER_ID, DEPARTMENT_ID
FROM EMPLOYEES
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

Creating Views

Like tables, views can be queried, updated, inserted into, and deleted from, with some restrictions. All operations performed on a view actually affect the base tables of the view. Views provide an additional level of security by restricting access to a predetermined set of rows and columns of a table. They also hide data complexity and store complex queries.

To see the views defined in the database, click the Views link under the Schema heading on the Administration page.



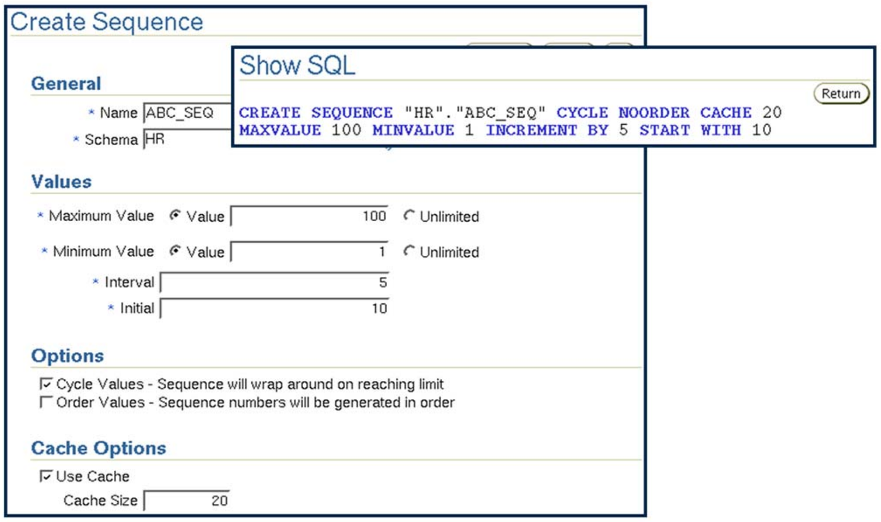
Sequences

- A sequence is a mechanism for automatically generating integers that follow a pattern.
  - A sequence has a name, which is how it is referenced when the next value is requested.
  - A sequence is not associated with any particular table or column.
  - The progression can be ascending or descending.
  - The interval between numbers can be of any size.
  - A sequence can cycle when a limit is reached.

Sequences

To retrieve the next value from a sequence, you reference it by its name; there is no association of a sequence to a table or a column.
After a given number is issued, it will not be issued again, unless the sequence is defined as cyclical. Sometimes an application requests a value it never ends up using or storing in the database. This may result in gaps in the numbers that reside in the table that they are being stored into.
Caching of sequence numbers improves performance because a set of numbers is preallocated in memory for faster access. If there is an instance failure, any cached sequence numbers are not used, which results in gaps.
Note: If an application requires that there be no gaps, then the application should implement a custom number generator. However, this method can result in very poor performance. If you use a table to store a value, and increment that value and update the table for each request, that process would be a systemwide bottleneck. This is because every session would have to wait for that mechanism, which, to guarantee no duplicates or gaps, can handle only a single request at a time.

Creating a Sequence



Create Sequence

General

* Name	ABC_SEQ
* Schema	HR

Values

* Maximum Value	[✓ Value]	100	Unlimited
* Minimum Value	[✓ Value]	1	Unlimited
* Interval		5	
* Initial		10	

Options

[✓] Cycle Values - Sequence will wrap around on reaching limit
[ ] Order Values - Sequence numbers will be generated in order

Cache Options

[✓] Use Cache
Cache Size 20

Show SQL

```
CREATE SEQUENCE "HR"."ABC_SEQ" CYCLE NOORDER CACHE 20
MAXVALUE 100 MINVALUE 1 INCREMENT BY 5 START WITH 10
```

Creating a Sequence

You can view and create sequences with Enterprise Manager by clicking the Sequences link under the Schema heading of the Administration page. Here is a summary of the sequence creation options:

Name: This is the name of the sequence, which is how it is referenced.

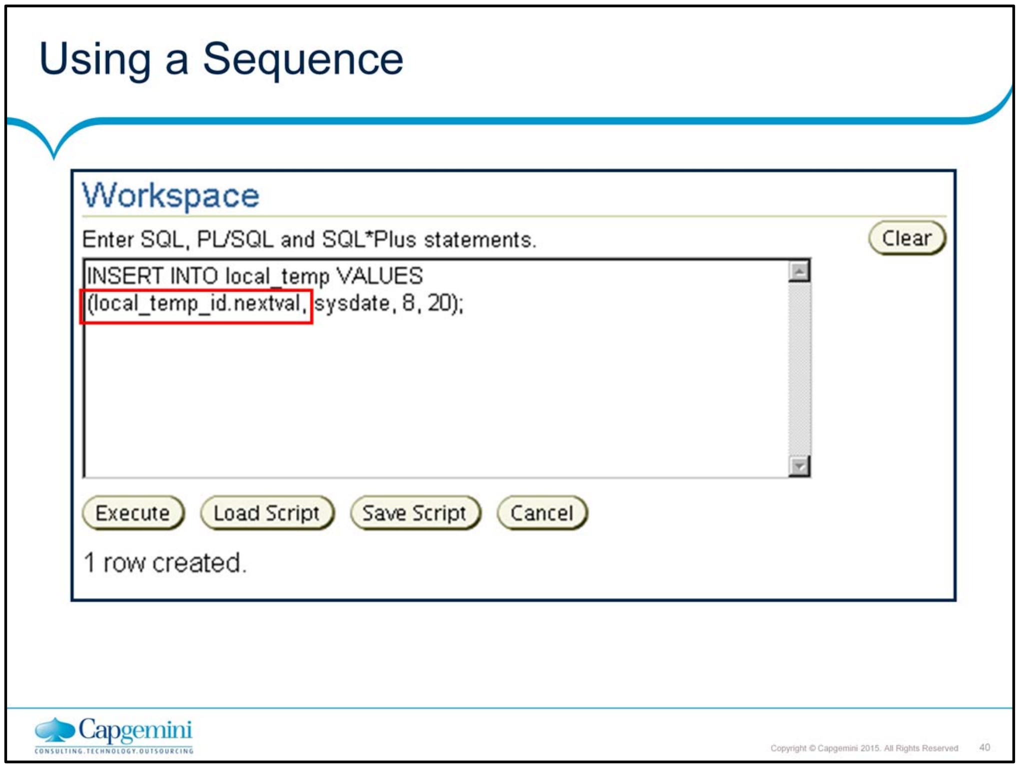
Schema: This is the owner of the sequence.

Maximum Value: Specify the maximum value that the sequence can generate. This integer value can have 28 or fewer digits. It must be greater than Minimum Value and Initial. Using Unlimited indicates the maximum value of 1027 for an ascending sequence or –1 for a descending sequence. The default is Unlimited.

Minimum Value: Specify the minimum value of the sequence. This integer value can have 28 or fewer digits. It must be less than or equal to Initial and less than Maximum Value. Using Unlimited indicates the minimum value of 1 for an ascending sequence or –1026 for a descending sequence. The default is Unlimited.

Creating a Sequence (continued)

Interval: Specify the interval between sequence numbers. This integer value can be any positive or negative integer, but it cannot be zero. It can have 28 or fewer digits. The default value is one.
Initial: Specify the first sequence number to be generated. Use this clause to start an ascending sequence at a value greater than its minimum or to start a descending sequence at a value less than its maximum.
Cycle Values: After an ascending sequence reaches its maximum value, it generates its minimum value. After a descending sequence reaches its minimum, it generates its maximum value. If you do not choose this option, an error is returned when you attempt to retrieve a value after the sequence has been exhausted.
Order Values: This guarantees that sequence numbers are generated in the order of request. This clause is useful if you are using sequence numbers as timestamps. Guaranteeing order is usually not important for sequences that are used to generate primary keys. This option is necessary only to guarantee ordered generation if you are using the Oracle database with Real Application Clusters.
Cache Options: Specify how many values of the sequence the Oracle database preallocates and keeps in memory for faster access. This integer value can have 28 or fewer digits. The minimum value for this parameter is 2. For sequences that cycle, this value must be less than the number of values in the cycle. You cannot cache more values than what would fit in a given cycle of sequence numbers.



Using a Sequence

```
INSERT INTO local_temp VALUES
(local_temp_id.nextval, sysdate, 8, 20);
```

```
1 row created.
```

Using a Sequence

Refer to sequence values in SQL statements with the following pseudocolumns:

CURRVAL: Returns the current value of a sequence

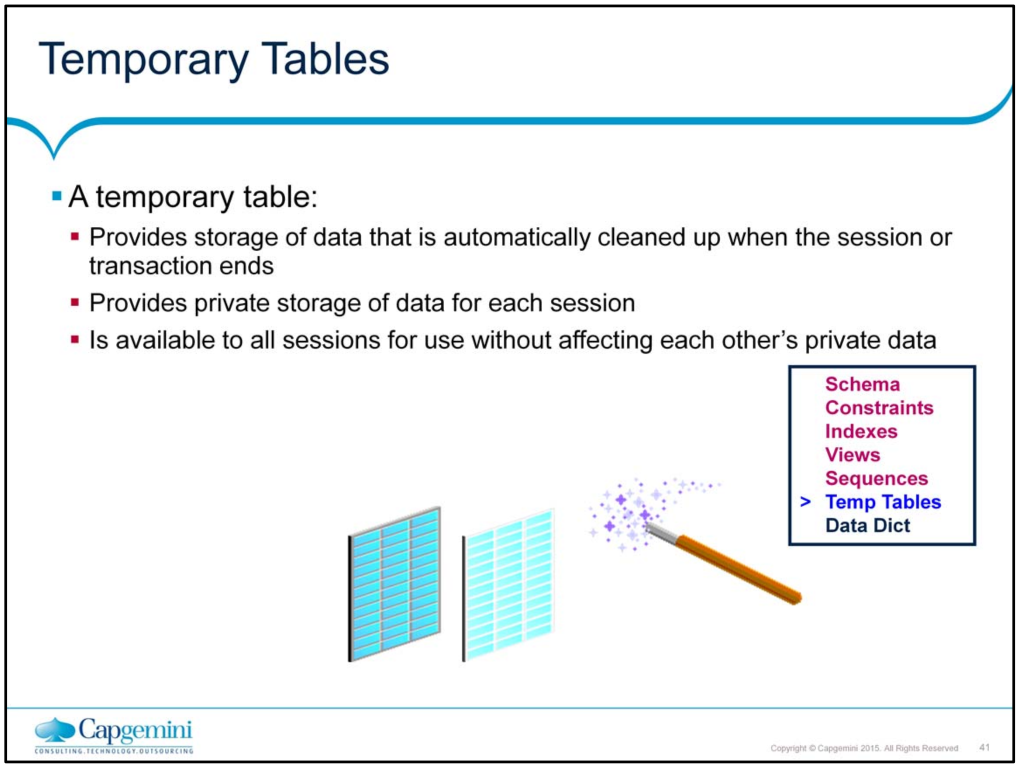
NEXTVAL: Increments the sequence and returns the next value

You must qualify CURRVAL and NEXTVAL with the name of the sequence:

sequence.CURRVAL

sequence.NEXTVAL

The first reference to NEXTVAL returns the initial value of the sequence. Subsequent references to NEXTVAL increment the sequence value by the defined increment and return the new value. Any reference to CURRVAL always returns the current value of the sequence, which is the value returned by the last reference to NEXTVAL.



Temporary Tables

- A temporary table:
  - Provides storage of data that is automatically cleaned up when the session or transaction ends
  - Provides private storage of data for each session
  - Is available to all sessions for use without affecting each other's private data

Schema
Constraints
Indexes
Views
Sequences
> Temp Tables
Data Dict
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Temporary Tables

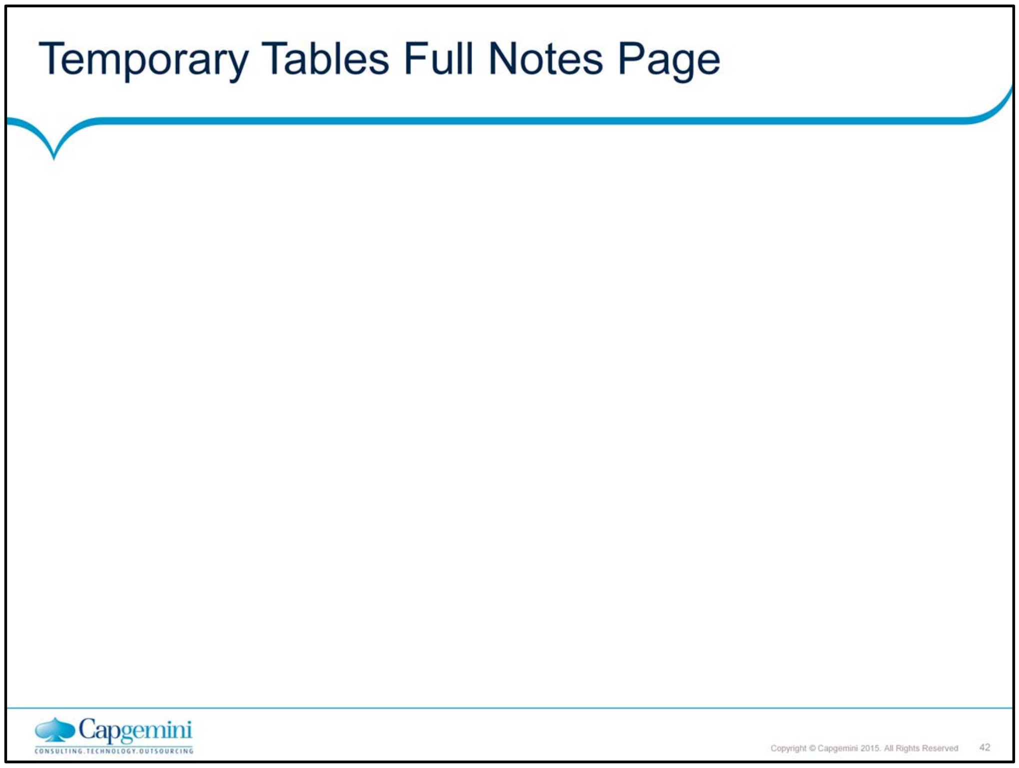
You can take advantage of temporary tables when you need to privately store data for the purpose of performing a task, and you want the data to be cleaned up when that task is performed, at the end of either a transaction or a session. Temporary tables provide this functionality while relieving you of the responsibilities of hiding your data from other sessions, and removing the generated data when you have finished. The only temporary table data visible to a session is the data that the session has inserted.

A temporary table can be transaction specific or session specific. For transaction-specific temporary tables, data exists for the duration of the transaction whereas for session-specific temporary tables, data exists for the duration of the session. In both cases, the data inserted by a session is private to the session. Each session can view and modify only its own data. As a result, DML locks are never acquired on the data of temporary tables.

The following clauses control the lifetime of the rows:

ON COMMIT DELETE ROWS: To specify that the lifetime of the inserted rows is for the duration of the transaction only

ON COMMIT PRESERVE ROWS: To specify that the lifetime of the inserted rows is for the duration of the session



Temporary Tables Full Notes Page

Temporary Tables (continued)

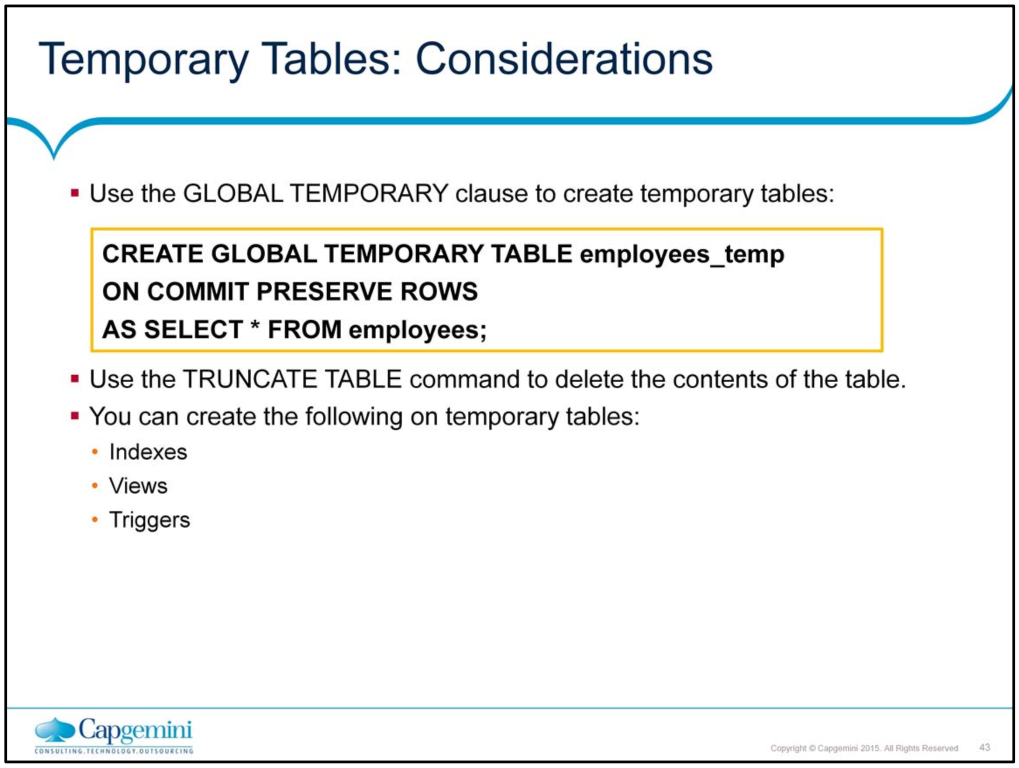
The CREATE GLOBAL TEMPORARY TABLE statement creates a temporary table. You can create indexes, views, and triggers on temporary tables, and you can also use Export and Import or Data Pump to export and import the definition of a temporary table. However, no data is exported, even if you use the ROWS option.

In addition to the already mentioned events that cause the data to be deleted, you can force the data to be removed efficiently with the TRUNCATE TABLE command. This removes all the data that you have inserted. It is more efficient than using the DELETE command.

You can create indexes, views, and triggers on temporary tables.

Temporary tables can be created using Enterprise Manager by clicking the Temporary option on the Create Table: Table Organization page. Click Continue, and the next page enables you to specify whether the temporary table is session specific or transaction specific. The Tablespace field is disabled because a temporary table is always created in the user's temporary tablespace; no other tablespace can be specified.

Note: The GLOBAL keyword is based on the terminology specified in the International Organization for Standardization (ISO) standard for SQL.



Temporary Tables: Considerations

- Use the GLOBAL TEMPORARY clause to create temporary tables:

```
CREATE GLOBAL TEMPORARY TABLE employees_temp
ON COMMIT PRESERVE ROWS
AS SELECT * FROM employees;
```

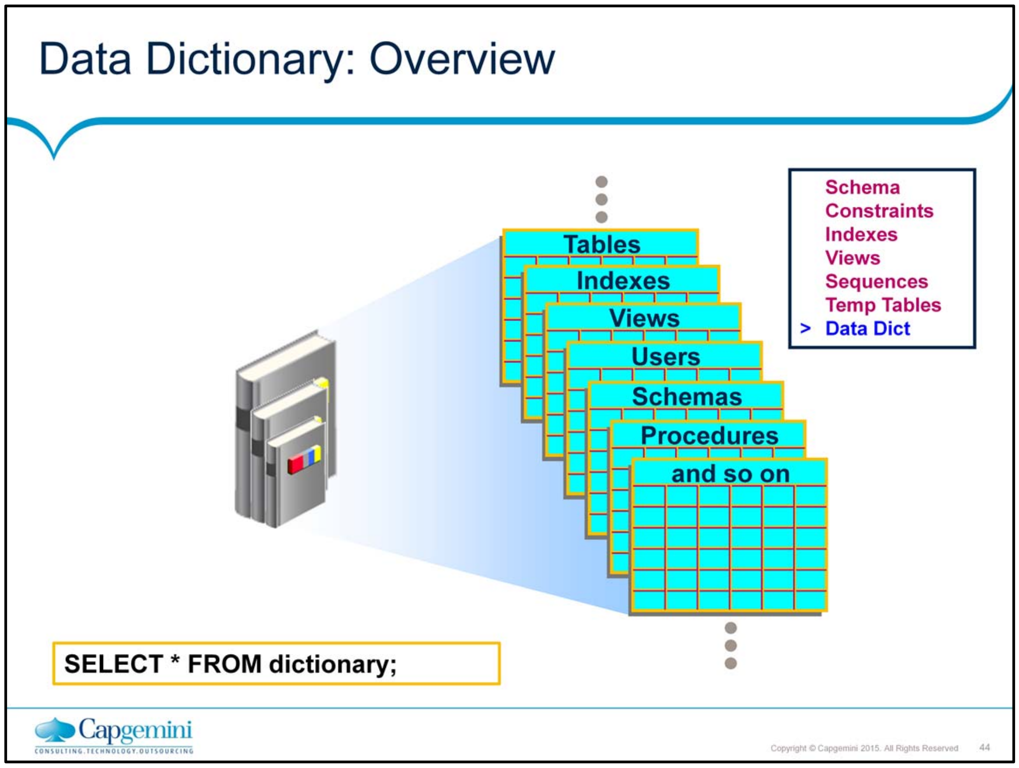
- Use the TRUNCATE TABLE command to delete the contents of the table.
- You can create the following on temporary tables:
  - Indexes
  - Views
  - Triggers

Temporary Tables: Considerations

The CREATE GLOBAL TEMPORARY TABLE statement creates a temporary table. You can create indexes, views, and triggers on temporary tables, and you can also use Export and Import or Data Pump to export and import the definition of a temporary table. However, no data is exported even if you use the ROWS option.

In addition to the already mentioned events that cause the data to be deleted, you can force the data to be removed efficiently with the TRUNCATE TABLE command. This removes all the data that you have inserted. It is more efficient than using the DELETE command.

Note: The GLOBAL keyword is based on the terminology specified in the International Organization for Standardization (ISO) standard for SQL.



Data Dictionary: Overview

```
SELECT * FROM dictionary;
```

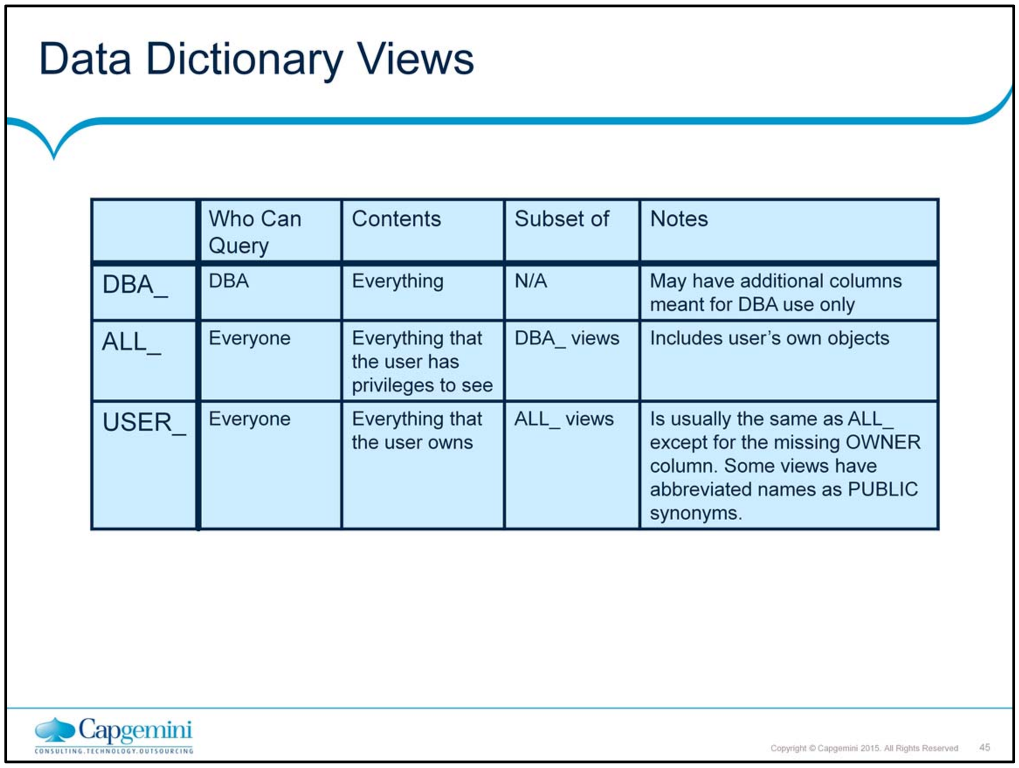
Data Dictionary: Overview

Oracle's data dictionary is the description of a database. It contains the names and attributes of all objects in the database. The creation or modification of any object causes an update to the data dictionary that reflects those changes. This information is stored in the base tables that are maintained by the Oracle database, but you access these tables by using predefined views rather than reading the tables directly.

The data dictionary:

- Is used by the Oracle database server to find information about users, objects, constraints, and storage
- Is maintained by the Oracle database server as object structures or definitions are modified
- Is available for use by any user to query information about the database
- Is owned by the SYS user
- Should never be modified directly using SQL

Note: The DICTIONARY data dictionary view, or the DICT synonym for this, contains the names and descriptions of everything in the data dictionary. Use the DICT_COLUMNS view to see the view columns and their definitions. For complete definitions of each view, see the Oracle Database Reference documentation.

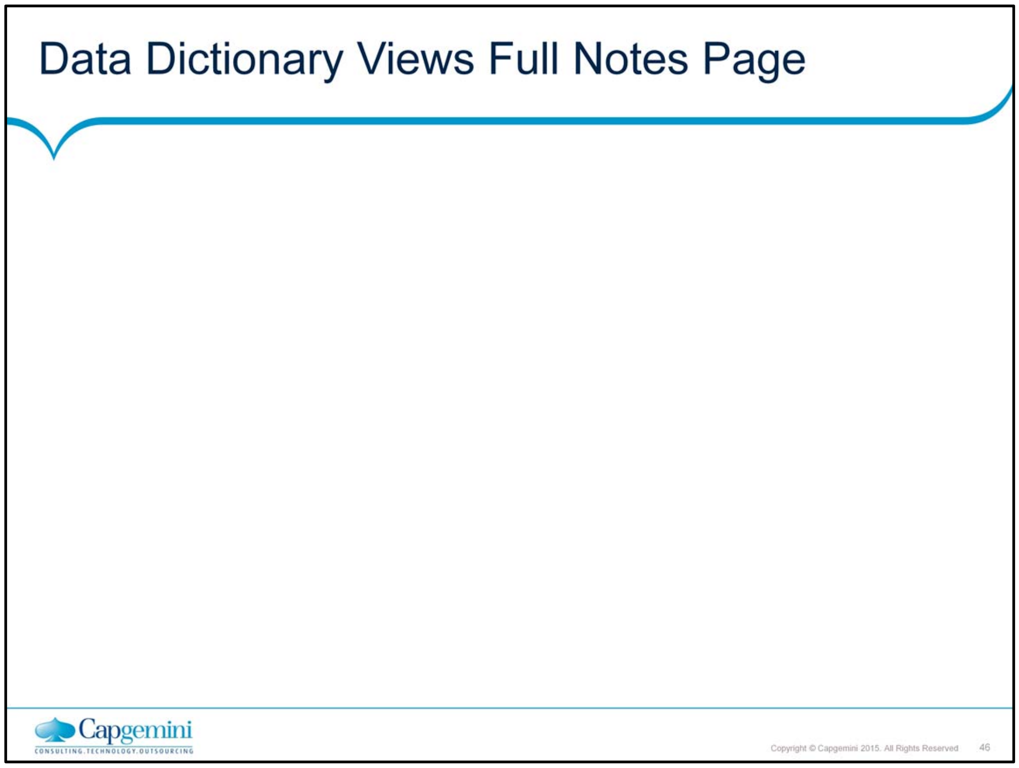


Data Dictionary Views

	Who Can Query	Contents	Subset of	Notes
DBA_	DBA	Everything	N/A	May have additional columns meant for DBA use only
ALL_	Everyone	Everything that the user has privileges to see	DBA_ views	Includes user's own objects
USER_	Everyone	Everything that the user owns	ALL_ views	Is usually the same as ALL_ except for the missing OWNER column. Some views have abbreviated names as PUBLIC synonyms.

Data Dictionary Views

The view prefixes indicate what or how much data a given user can see. The global view of everything is accessed only by users with DBA privileges, using the DBA_ prefix. The next level of privilege is at the ALL_ prefix level, which represents all objects that the querying user is privileged to see, whether he or she owns them or not. For example, if USER_A has been granted access to a table owned by USER_B, then USER_A sees that table listed in any ALL_ view dealing with table names. The USER_ prefix represents the smallest scope of visibility. This shows only those objects that the querying user owns; that is, those that are present in his or her own schema.

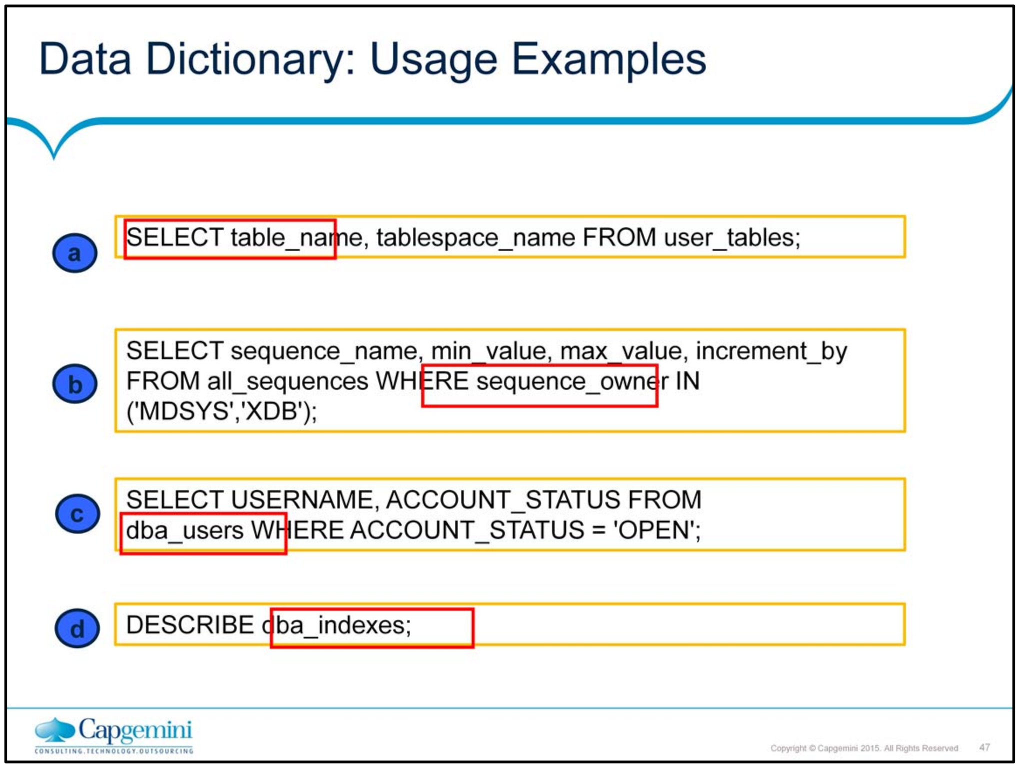


Data Dictionary Views Full Notes Page

Data Dictionary Views (continued)

Generally, each view set is a subset of the higher privileged view set, row-wise and columnwise. Not all views in a given view set have a corresponding view in the other view sets. This is dependent on the nature of the information in the view. For example, there is a DBA_LOCK view, but there is no ALL_LOCK view. This is because only a DBA would have interest in data about locks. You should be certain to choose the appropriate view set to meet the need that you have. If you have the privilege to access the DBA views, you still may want to query only the USER version of the view because you know that it is something that you own and you do not want other objects to be added to your result set.

The DBA_ views can be queried by users with the SYSDBA or SELECT ANY DICTIONARY privilege.



Data Dictionary: Usage Examples

a
```
SELECT table_name, tablespace_name FROM user_tables;
```

b
```
SELECT sequence_name, min_value, max_value, increment_by
FROM all_sequences WHERE sequence_owner IN
('MDSYS','XDB');
```

c
```
SELECT USERNAME, ACCOUNT_STATUS FROM
dba_users WHERE ACCOUNT_STATUS = 'OPEN';
```

d
```
DESCRIBE dba_indexes;
```

Static Data Dictionary: Usage Examples

The examples in the slide show queries that answer these questions:

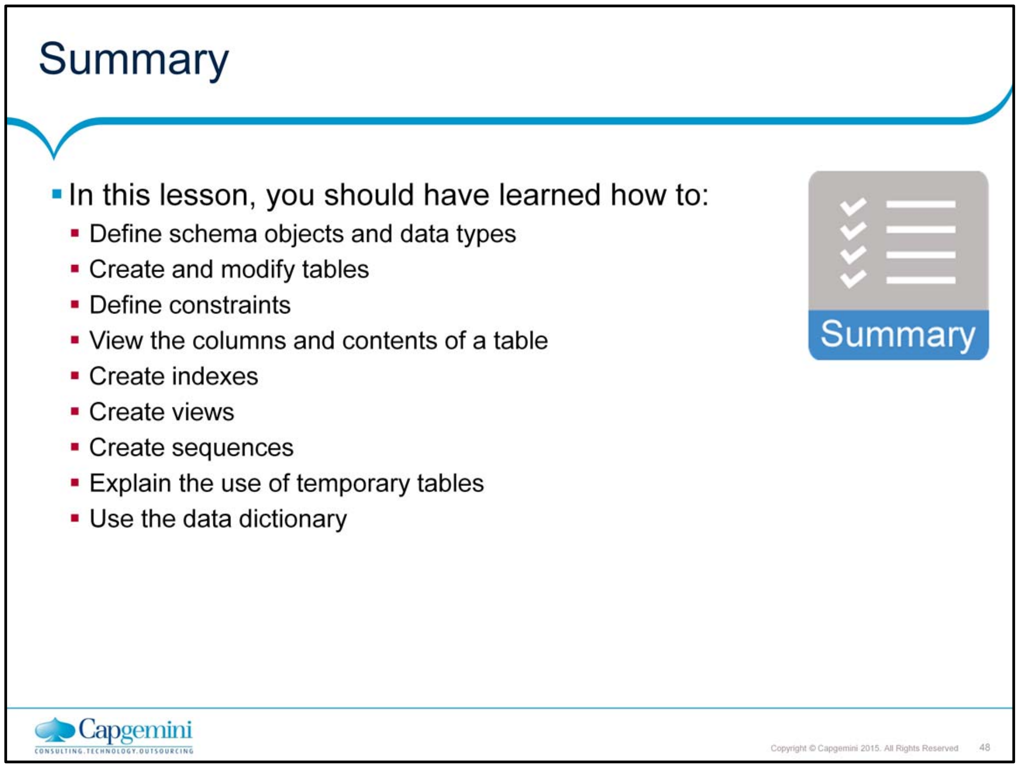
a. What are the names of the tables (along with the name of the tablespace where they reside) that have been created in your schema?

b. What is the significant information about any sequences in the database that you have access to?

c. What users in this database are currently able to log in?

d. What are the columns of the DBA_INDEXES view? This shows you what information you can view about all the indexes in the database. The following is a partial output of this command:

```
SQL> DESCRIBE dba_indexes;
 Name           Null?    Type
 --------------- -------- -------------
 OWNER          NOT NULL VARCHAR2(30)
 INDEX_NAME     NOT NULL VARCHAR2(30)
 INDEX_TYPE              VARCHAR2(27)
 TABLE_OWNER    NOT NULL VARCHAR2(30)
 TABLE_NAME     NOT NULL VARCHAR2(30)
```



Summary

- In this lesson, you should have learned how to:
  - Define schema objects and data types
  - Create and modify tables
  - Define constraints
  - View the columns and contents of a table
  - Create indexes
  - Create views
  - Create sequences
  - Explain the use of temporary tables
  - Use the data dictionary