9

Using DDL Statements to Create and Manage Tables

Objectives

After completing this lesson, you should be able to do the following:

- **Categorize the main database objects**
- **Review the table structure**
- **List the data types that are available for columns**
- **Create a simple table**
- **Explain how constraints are created at the time of table creation**
- **Describe how schema objects work**

ORACLE

Objectives

In this lesson, you are introduced to the data definition language (DDL) statements. You are taught the basics of how to create simple tables, alter them, and remove them. The data types available in DDL are shown, and schema concepts are introduced. Constraints are tied into this lesson. Exception messages that are generated from violating constraints during DML are shown and explained.

Database Objects

Object	Description
Table	Basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative names to objects

ORACLE

Database Objects

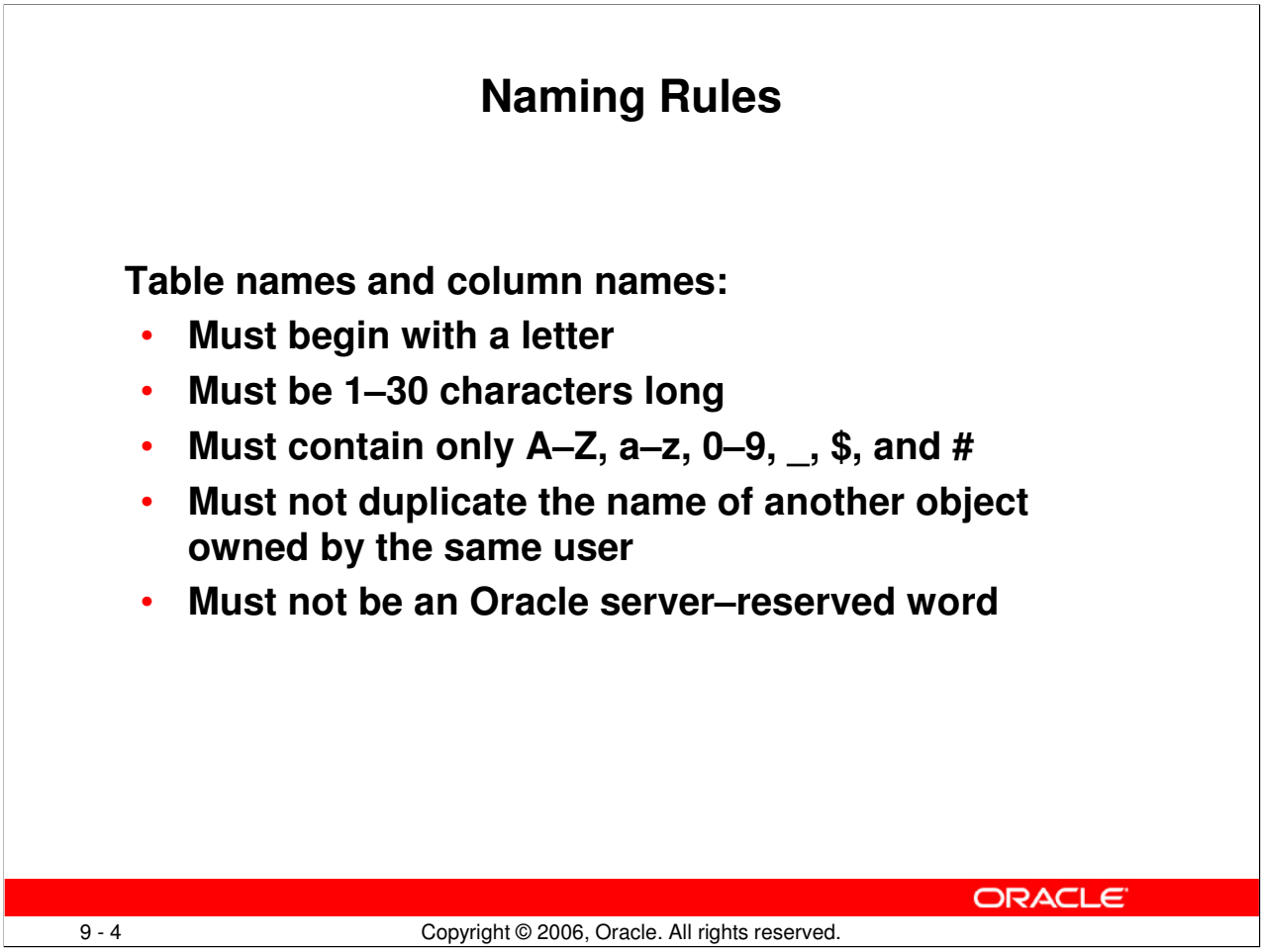
An Oracle database can contain multiple data structures. Each structure should be outlined in the database design so that it can be created during the build stage of database development.

- **Table:** Stores data
- **View:** Subset of data from one or more tables
- **Sequence:** Generates numeric values
- **Index:** Improves the performance of some queries
- **Synonym:** Gives alternative names to objects

Oracle Table Structures

- Tables can be created at any time, even while users are using the database.
- You do not need to specify the size of a table. The size is ultimately defined by the amount of space allocated to the database as a whole. It is important, however, to estimate how much space a table will use over time.
- Table structure can be modified online.

Note: More database objects are available but are not covered in this course.



Naming Rules

You name database tables and columns according to the standard rules for naming any Oracle database object:

- Table names and column names must begin with a letter and be 1–30 characters long.
- Names must contain only the characters A–Z, a–z, 0–9, _ (underscore), $, and # (legal characters, but their use is discouraged).
- Names must not duplicate the name of another object owned by the same Oracle server user.
- Names must not be an Oracle server–reserved word.

Naming Guidelines

Use descriptive names for tables and other database objects.

Note: Names are case-insensitive. For example, `EMPLOYEES` is treated as the same name as `eMPloyees` or `eMpLOYEES`.

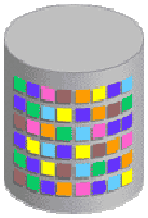
For more information, see “Object Names and Qualifiers” in the *Oracle Database SQL Reference.*

CREATE TABLE Statement

- **You must have:**
  - **CREATE TABLE privilege**
  - **A storage area**

```
CREATE TABLE [schema.]table
        (column datatype [DEFAULT expr][, ...]);
```

- **You specify:**
  - **Table name**
  - **Column name, column data type, and column size**



ORACLE

CREATE TABLE Statement

You create tables to store data by executing the SQL CREATE TABLE statement. This statement is one of the DDL statements, which are a subset of SQL statements used to create, modify, or remove Oracle database structures. These statements have an immediate effect on the database, and they also record information in the data dictionary.

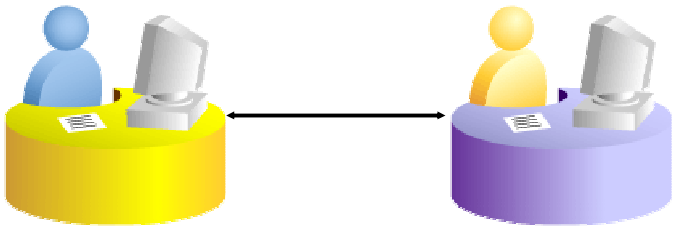
To create a table, a user must have the CREATE TABLE privilege and a storage area in which to create objects. The database administrator uses data control language statements to grant privileges to users (DCL statements are covered in a later lesson).

In the syntax:

schema	Is the same as the owner's name
table	is the name of the table
DEFAULT *expr*	Specifies a default value if a value is omitted in the INSERT statement
column	Is the name of the column
datatype	Is the column's data type and length

Referencing Another User's Tables

- **Tables belonging to other users are not in the user's schema.**
- **You should use the owner's name as a prefix to those tables.**



USERA

```
SELECT *
FROM userB.employees;
```

USERB

```
SELECT *
FROM userA.employees;
```

ORACLE

Referencing Another User's Tables

A *schema* is a collection of objects. Schema objects are the logical structures that directly refer to the data in a database. Schema objects include tables, views, synonyms, sequences, stored procedures, indexes, clusters, and database links.

If a table does not belong to the user, the owner's name must be prefixed to the table. For example, if there are schemas named `USERA` and `USERB`, and both have an `EMPLOYEES` table, then if `USERA` wants to access the `EMPLOYEES` table that belongs to `USERB`, he must prefix the table name with the schema name:

```
SELECT *
FROM   userb.employees;
```

If `USERB` wants to access the `EMPLOYEES` table that is owned by `USERA`, he must prefix the table name with the schema name:

```
SELECT *
FROM   usera.employees;
```

DEFAULT Option

- Specify a default value for a column during an insert.

```
... hire_date DATE DEFAULT SYSDATE, ...
```

- Literal values, expressions, or SQL functions are legal values.
- Another column's name or a pseudocolumn are illegal values.
- The default data type must match the column data type.

```
CREATE TABLE hire_dates
        (id          NUMBER(8),
         hire_date DATE DEFAULT SYSDATE);
Table created.
```

ORACLE

9 - 7

DEFAULT Option

When you define a table, you can specify that a column be given a default value by using the DEFAULT option. This option prevents null values from entering the columns if a row is inserted without a value for the column. The default value can be a literal, an expression, or a SQL function (such as SYSDATE or USER), but the value cannot be the name of another column or a pseudocolumn (such as NEXTVAL or CURRVAL). The default expression must match the data type of the column.

Note: CURRVAL and NEXTVAL are explained later in this lesson.

Creating Tables

- **Create the table.**

```
CREATE TABLE dept
        (deptno      NUMBER(2),
         dname       VARCHAR2(14),
         loc         VARCHAR2(13),
         create_date DATE DEFAULT SYSDATE);
Table created.
```

- **Confirm table creation.**

```
DESCRIBE dept
```

Name	Null?	Type
DEPTNO		NUMBER(2)
DNAME		VARCHAR2(14)
LOC		VARCHAR2(13)
CREATE_DATE		DATE

ORACLE

Creating Tables

The example in the slide creates the `DEPT` table, with four columns: `DEPTNO`, `DNAME`, `LOC`, and `CREATE_DATE`. The `CREATE_DATE` column has a default value. If a value is not provided for an `INSERT` statement, the system date is automatically inserted.

It further confirms the creation of the table by issuing the `DESCRIBE` command.

Because creating a table is a DDL statement, an automatic commit takes place when this statement is executed.

Data Types

Data Type	Description
VARCHAR2(*size*)	Variable-length character data
CHAR(*size*)	Fixed-length character data
NUMBER(*p*,*s*)	Variable-length numeric data
DATE	Date and time values
LONG	Variable-length character data (up to 2 GB)
CLOB	Character data (up to 4 GB)
RAW and LONG RAW	Raw binary data
BLOB	Binary data (up to 4 GB)
BFILE	Binary data stored in an external file (up to 4 GB)
ROWID	A base-64 number system representing the unique address of a row in its table

ORACLE

Data Types

When you identify a column for a table, you need to provide a data type for the column. There are several data types available:

Data Type	Description
VARCHAR2(*size*)	Variable-length character data (A maximum *size* must be specified: minimum *size* is 1; maximum *size* is 4,000.)
CHAR [(*size*)]	Fixed-length character data of length *size* bytes (Default and minimum *size* is 1; maximum *size* is 2,000.)
NUMBER [(*p*,*s*)]	Number having precision *p* and scale *s* (The precision is the total number of decimal digits, and the scale is the number of digits to the right of the decimal point; the precision can range from 1 to 38, and the scale can range from –84 to 127.)
DATE	Date and time values to the nearest second between January 1, 4712 B.C., and December 31, 9999 A.D.
LONG	Variable-length character data (up to 2 GB)
CLOB	Character data (up to 4 GB)

Data Types (continued)

Data Type	Description
RAW(*size*)	Raw binary data of length *size* (A maximum *size* must be specified: maximum *size* is 2,000.)
LONG RAW	Raw binary data of variable length (up to 2 GB)
BLOB	Binary data (up to 4 GB)
BFILE	Binary data stored in an external file (up to 4 GB)
ROWID	A base-64 number system representing the unique address of a row in its table

Guidelines

- A LONG column is not copied when a table is created using a subquery.
- A LONG column cannot be included in a GROUP BY or an ORDER BY clause.
- Only one LONG column can be used per table.
- No constraints can be defined on a LONG column.
- You might want to use a CLOB column rather than a LONG column.

Datetime Data Types

You can use several datetime data types:

Data Type	Description
TIMESTAMP	Date with fractional seconds
INTERVAL YEAR TO MONTH	Stored as an interval of years and months
INTERVAL DAY TO SECOND	Stored as an interval of days, hours, minutes, and seconds

Other Datetime Data Types

Data Type	Description
TIMESTAMP	Enables the time to be stored as a date with fractional seconds. There are several variations of this data type.
INTERVAL YEAR TO MONTH	Enables time to be stored as an interval of years and months. Used to represent the difference between two datetime values in which the only significant portions are the year and month.
INTERVAL DAY TO SECOND	Enables time to be stored as an interval of days, hours, minutes, and seconds. Used to represent the precise difference between two datetime values.

Note: These datetime data types are available with Oracle9*i* and later releases. For detailed information about the datetime data types, see the topics "TIMESTAMP Datatype," "INTERVAL YEAR TO MONTH Datatype," and "INTERVAL DAY TO SECOND Datatype" in the *Oracle SQL Reference*.

Datetime Data Types

- The **TIMESTAMP** data type is an extension of the **DATE** data type.
- It stores the year, month, and day of the **DATE** data type plus hour, minute, and second values as well as the fractional second value.
- You can optionally specify the time zone.

```
TIMESTAMP[(fractional_seconds_precision)]
```

```
TIMESTAMP[(fractional_seconds_precision)]
WITH TIME ZONE
```

```
TIMESTAMP[(fractional_seconds_precision)]
WITH LOCAL TIME ZONE
```

ORACLE

TIMESTAMP Data Type

The TIMESTAMP data type is an extension of the DATE data type. It stores the year, month, and day of the DATE data type plus hour, minute, and second values. This data type is used for storing precise time values.

The fractional_seconds_precision optionally specifies the number of digits in the fractional part of the SECOND datetime field and can be a number in the range 0 to 9. The default is 6.

Example

In this example, a table is created named NEW_EMPLOYEES, with a column START_DATE that has a data type of TIMESTAMP:

```
CREATE TABLE new_employees
  (employee_id NUMBER,
   first_name VARCHAR2(15),
   last_name VARCHAR2(15),
   ...
   start_date TIMESTAMP(7),
   ...);
```

Suppose that two rows are inserted in the NEW_EMPLOYEES table. The displayed output shows the differences. (A DATE data type defaults to display the DD-MON-RR format.):

TIMESTAMP Data Type (continued)

```
SELECT start_date
FROM   new_employees;

17-JUN-03 12.00.00.000000 AM
21-SEP-03 12.00.00.000000 AM
```

TIMESTAMP WITH TIME ZONE Data Type

TIMESTAMP WITH TIME ZONE is a variant of TIMESTAMP that includes a time-zone displacement in its value. The time-zone displacement is the difference (in hours and minutes) between local time and UTC (Universal Time Coordinate, formerly known as Greenwich Mean Time). This data type is used for collecting and evaluating date information across geographic regions.

For example,

```
TIMESTAMP '2003-04-15 8:00:00 -8:00'
```

is the same as

```
TIMESTAMP '2003-04-15 11:00:00 -5:00'
```

That is, 8:00 a.m. Pacific Standard Time is the same as 11:00 a.m. Eastern Standard Time.

This can also be specified as follows:

```
TIMESTAMP '2003-04-15 8:00:00 US/Pacific'
```

TIMESTAMP WITH LOCAL TIME ZONE Data Type

TIMESTAMP WITH LOCAL TIME ZONE is another variant of TIMESTAMP that includes a time-zone displacement in its value. It differs from TIMESTAMP WITH TIME ZONE in that data stored in the database is normalized to the database time zone, and the time-zone displacement is not stored as part of the column data. When users retrieve the data, it is returned in the users' local session time zone. The time-zone displacement is the difference (in hours and minutes) between local time and UTC.

Unlike TIMESTAMP WITH TIME ZONE, you can specify columns of type TIMESTAMP WITH LOCAL TIME ZONE as part of a primary or unique key, as in the following example:

```
CREATE TABLE time_example
    (order_date TIMESTAMP WITH LOCAL TIME ZONE);

INSERT INTO time_example VALUES('15-JAN-04 09:34:28 AM');

SELECT *
FROM   time_example;

ORDER_DATE
---------------------------
15-JAN-04 09.34.28.000000 AM
```

The TIMESTAMP WITH LOCAL TIME ZONE type is appropriate for two-tier applications in which you want to display dates and times using the time zone of the client system.

Datetime Data Types

- **The `INTERVAL YEAR TO MONTH` data type stores a period of time using the `YEAR` and `MONTH` datetime fields:**

```
INTERVAL YEAR [(year_precision)] TO MONTH
```

- **The `INTERVAL DAY TO SECOND` data type stores a period of time in terms of days, hours, minutes, and seconds:**

```
INTERVAL DAY [(day_precision)]
   TO SECOND [(fractional_seconds_precision)]
```

ORACLE

`INTERVAL YEAR TO MONTH` Data Type

`INTERVAL YEAR TO MONTH` stores a period of time using the `YEAR` and `MONTH` datetime fields.

Use `INTERVAL YEAR TO MONTH` to represent the difference between two datetime values, where the only significant portions are the year and month. For example, you might use this value to set a reminder for a date that is 120 months in the future, or check whether 6 months have elapsed since a particular date.

In the syntax:

`year_precision`	is the number of digits in the `YEAR` datetime field. The default value of `year_precision` is 2.

Examples

- `INTERVAL '123-2' YEAR(3) TO MONTH`
  Indicates an interval of 123 years, 2 months
- `INTERVAL '123' YEAR(3)`
  Indicates an interval of 123 years 0 months
- `INTERVAL '300' MONTH(3)`
  Indicates an interval of 300 months
- `INTERVAL '123' YEAR`
  Returns an error because the default precision is 2, and `123` has 3 digits

INTERVAL YEAR TO MONTH Data Type (continued)

```
CREATE TABLE time_example2
(loan_duration INTERVAL YEAR (3) TO MONTH);

INSERT INTO time_example2 (loan_duration)
  VALUES (INTERVAL '120' MONTH(3));

SELECT TO_CHAR( sysdate+loan_duration, 'dd-mon-yyyy')
FROM   time_example2;        --today's date is 26-Sep-2001
```

TO_CHAR(SYS
26-sep-2011

INTERVAL DAY TO SECOND Data Type

INTERVAL DAY TO SECOND stores a period of time in terms of days, hours, minutes, and seconds.

Use INTERVAL DAY TO SECOND to represent the precise difference between two datetime values. For example, you might use this value to set a reminder for a time that is 36 hours in the future, or to record the time between the start and end of a race. To represent long spans of time, including multiple years, with high precision, you can use a large value for the days portion.

In the syntax:

day_precision	Is the number of digits in the DAY datetime field. Accepted values are 0 to 9. The default is 2.
fractional_seconds_precision	Is the number of digits in the fractional part of the SECOND datetime field. Accepted values are 0 to 9. The default is 6.

Examples

- INTERVAL '4 5:12:10.222' DAY TO SECOND(3)
  Indicates 4 days, 5 hours, 12 minutes, 10 seconds, and 222 thousandths of a second.
- INTERVAL '180' DAY(3)
  Indicates 180 days.
- INTERVAL '4 5:12:10.222' DAY TO SECOND(3)
  Indicates 4 days, 5 hours, 12 minutes, 10 seconds, and 222 thousandths of a second
- INTERVAL '4 5:12' DAY TO MINUTE
  Indicates 4 days, 5 hours, and 12 minutes
- INTERVAL '400 5' DAY(3) TO HOUR
  Indicates 400 days and 5 hours.
- INTERVAL '11:12:10.2222222' HOUR TO SECOND(7)
  Indicates 11 hours, 12 minutes, and 10.2222222 seconds.

INTERVAL DAY TO SECOND Data Type (continued)

Example

```
CREATE TABLE time_example3
(day_duration INTERVAL DAY (3) TO SECOND);

INSERT INTO time_example3 (day_duration)
VALUES (INTERVAL '180' DAY(3));

SELECT sysdate + day_duration "Half Year"
FROM   time_example3;        --today's date is 26-Sep-2001
```

Half Year
25-MAR-02

Including Constraints

- Constraints enforce rules at the table level.
- Constraints prevent the deletion of a table if there are dependencies.
- The following constraint types are valid:
  - NOT NULL
  - UNIQUE
  - PRIMARY KEY
  - FOREIGN KEY
  - CHECK

Constraints

The Oracle server uses constraints to prevent invalid data entry into tables.

You can use constraints to do the following:

- Enforce rules on the data in a table whenever a row is inserted, updated, or deleted from that table. The constraint must be satisfied for the operation to succeed.
- Prevent the deletion of a table if there are dependencies from other tables
- Provide rules for Oracle tools, such as Oracle Developer

Data Integrity Constraints

Constraint	Description
NOT NULL	Specifies that the column cannot contain a null value
UNIQUE	Specifies a column or combination of columns whose values must be unique for all rows in the table
PRIMARY KEY	Uniquely identifies each row of the table
FOREIGN KEY	Establishes and enforces a foreign key relationship between the column and a column of the referenced table
CHECK	Specifies a condition that must be true

Constraint Guidelines

- You can name a constraint, or the Oracle server generates a name by using the `SYS_Cn` format.
- Create a constraint at either of the following times:
  - At the same time as the table is created
  - After the table has been created
- Define a constraint at the column or table level.
- View a constraint in the data dictionary.

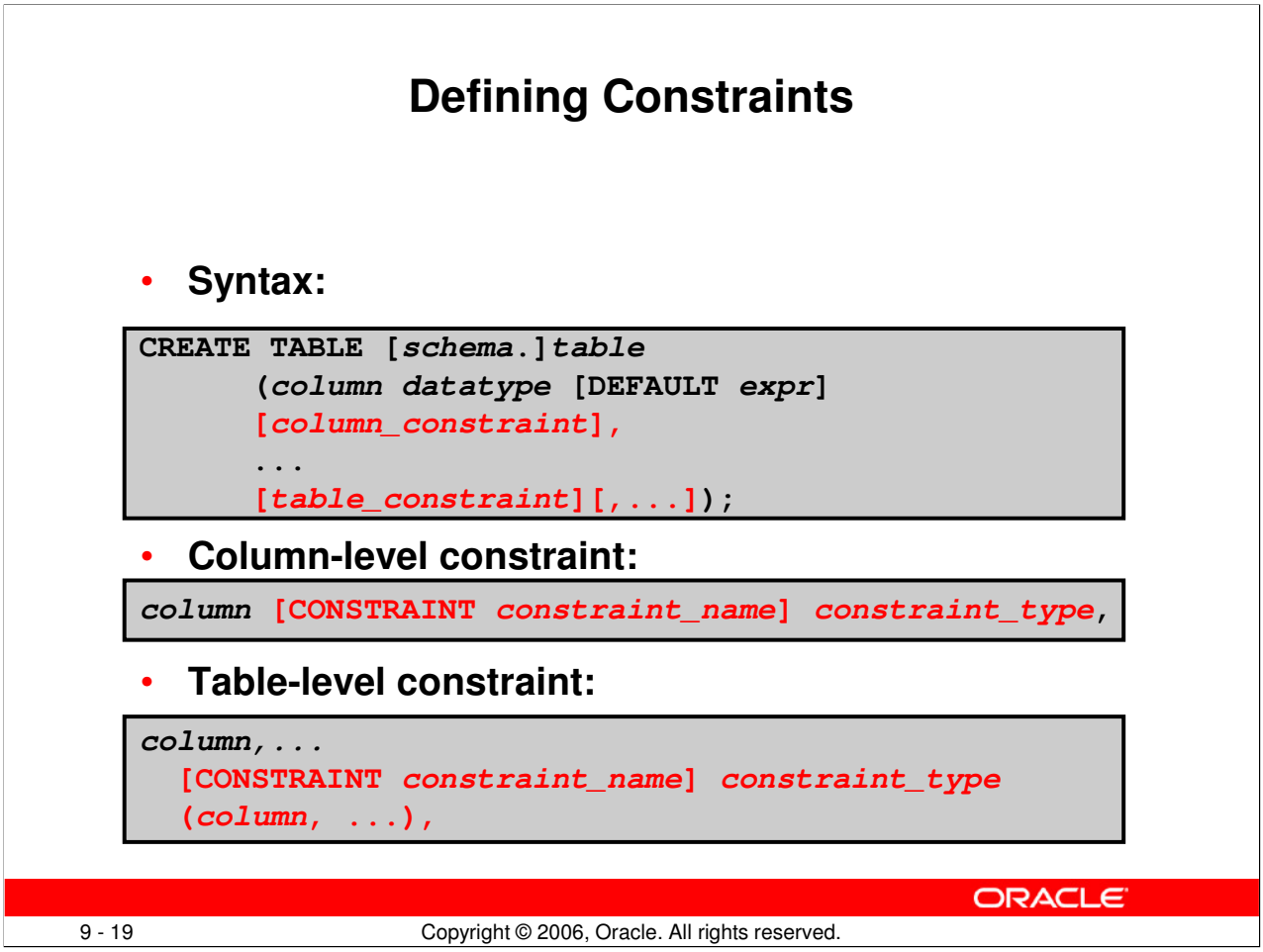
ORACLE

Constraint Guidelines

All constraints are stored in the data dictionary. Constraints are easy to reference if you give them a meaningful name. Constraint names must follow the standard object-naming rules. If you do not name your constraint, the Oracle server generates a name with the format `SYS_Cn`, where *n* is an integer so that the constraint name is unique.

Constraints can be defined at the time of table creation or after the table has been created.

For more information, see "Constraints" in *Oracle Database SQL Reference*.



Defining Constraints

The slide gives the syntax for defining constraints when creating a table. You can create the constraints at either the column level or table level. Constraints defined at the column level are included when the column is defined. Table-level constraints are defined at the end of the table definition and must refer to the column or columns on which the constraint pertains in a set of parentheses.

`NOT NULL` constraints must be defined at the column level.

Constraints that apply to more than one column must be defined at the table level.

In the syntax:

`schema`	Is the same as the owner's name
`table`	Is the name of the table
`DEFAULT expr`	Specifies a default value to use if a value is omitted in the `INSERT` statement
`column`	Is the name of the column
`datatype`	Is the column's data type and length
`column_constraint`	Is an integrity constraint as part of the column definition
`table_constraint`	Is an integrity constraint as part of the table definition

Defining Constraints

- **Column-level constraint:**

```
CREATE TABLE employees(
  employee_id  NUMBER(6)
    CONSTRAINT emp_emp_id_pk PRIMARY KEY,
  first_name   VARCHAR2(20),
  ...);
```

1

- **Table-level constraint:**

```
CREATE TABLE employees(
  employee_id  NUMBER(6),
  first_name   VARCHAR2(20),
  ...
  job_id       VARCHAR2(10) NOT NULL,
  CONSTRAINT emp_emp_id_pk
    PRIMARY KEY (EMPLOYEE_ID));
```

2

ORACLE

Defining Constraints (continued)

Constraints are usually created at the same time as the table. Constraints can be added to a table after its creation and also temporarily disabled.

Both slide examples create a primary key constraint on the `EMPLOYEE_ID` column of the `EMPLOYEES` table.

1. The first example uses the column-level syntax to define the constraint.
2. The second example uses the table-level syntax to define the constraint.

More details about the primary key constraint are provided later in this lesson.

NOT NULL Constraint

Ensures that null values are not permitted for the column:

EMPLOYEE_ID	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID
100	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000	90
101	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000	90
102	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000	90
103	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000	60
104	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000	60
178	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	7000	
200	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400	10

...

20 rows selected.

NOT NULL constraint (No row can contain a null value for this column.)

NOT NULL constraint

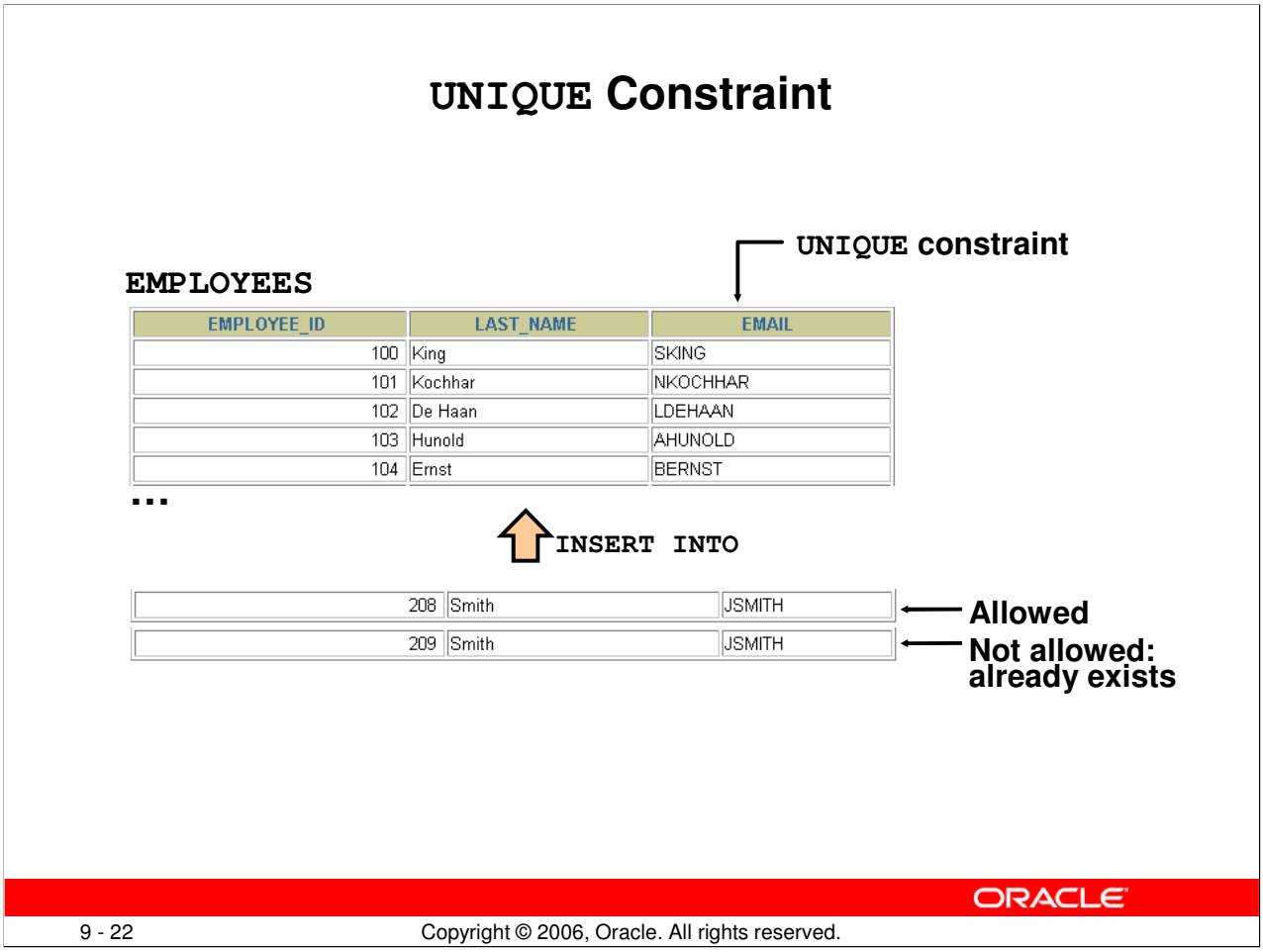
Absence of NOT NULL constraint (Any row can contain a null value for this column.)

ORACLE

9 - 21

NOT NULL Constraint

The NOT NULL constraint ensures that the column contains no null values. Columns without the NOT NULL constraint can contain null values by default. NOT NULL constraints must be defined at the column level.



UNIQUE Constraint

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 can have duplicate values in a specified column or set of columns. The column (or set of columns) included in the definition of the UNIQUE key constraint is called the *unique key*. If the UNIQUE constraint comprises more than one column, that group of columns is called a *composite unique key*.

UNIQUE constraints enable the input of nulls unless you also define NOT NULL constraints for the same columns. In fact, any number of rows can include nulls for columns without NOT NULL constraints because nulls are not considered equal to anything. A null in a column (or in all columns of a composite UNIQUE key) always satisfies a UNIQUE constraint.

Note: Because of the search mechanism for UNIQUE constraints on more than one column, you cannot have identical values in the non-null columns of a partially null composite UNIQUE key constraint.

UNIQUE Constraint

Defined at either the table level or the column level:

```
CREATE TABLE employees(
    employee_id      NUMBER(6),
    last_name        VARCHAR2(25) NOT NULL,
    email            VARCHAR2(25),
    salary           NUMBER(8,2),
    commission_pct   NUMBER(2,2),
    hire_date        DATE NOT NULL,
...
    CONSTRAINT emp_email_uk UNIQUE(email));
```

ORACLE

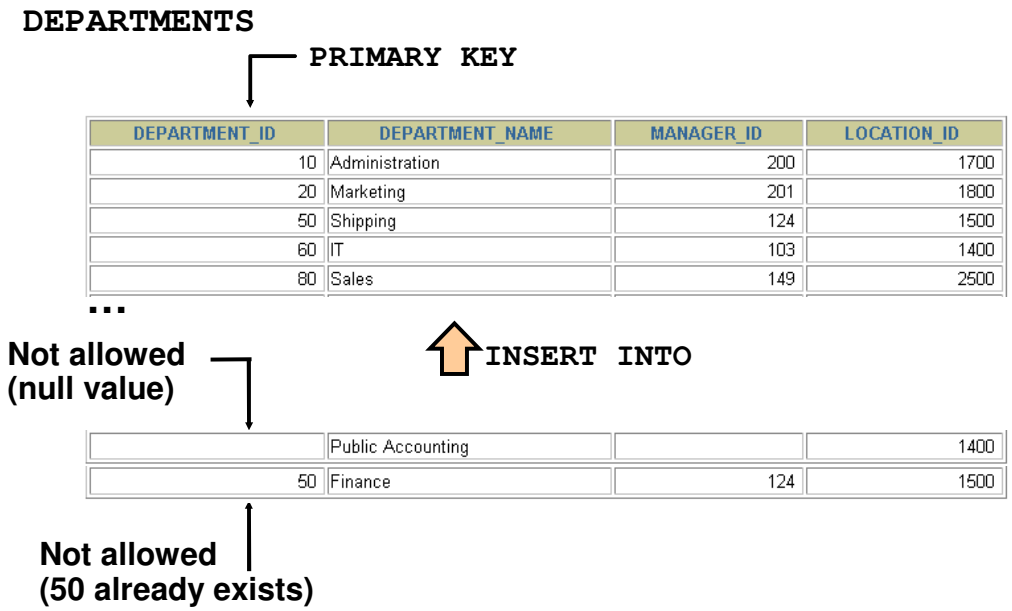
UNIQUE Constraint (continued)

UNIQUE constraints can be defined at the column level or table level. A composite unique key is created by using the table-level definition.

The example in the slide applies the UNIQUE constraint to the EMAIL column of the EMPLOYEES table. The name of the constraint is EMP_EMAIL_UK.

Note: The Oracle server enforces the UNIQUE constraint by implicitly creating a unique index on the unique key column or columns.

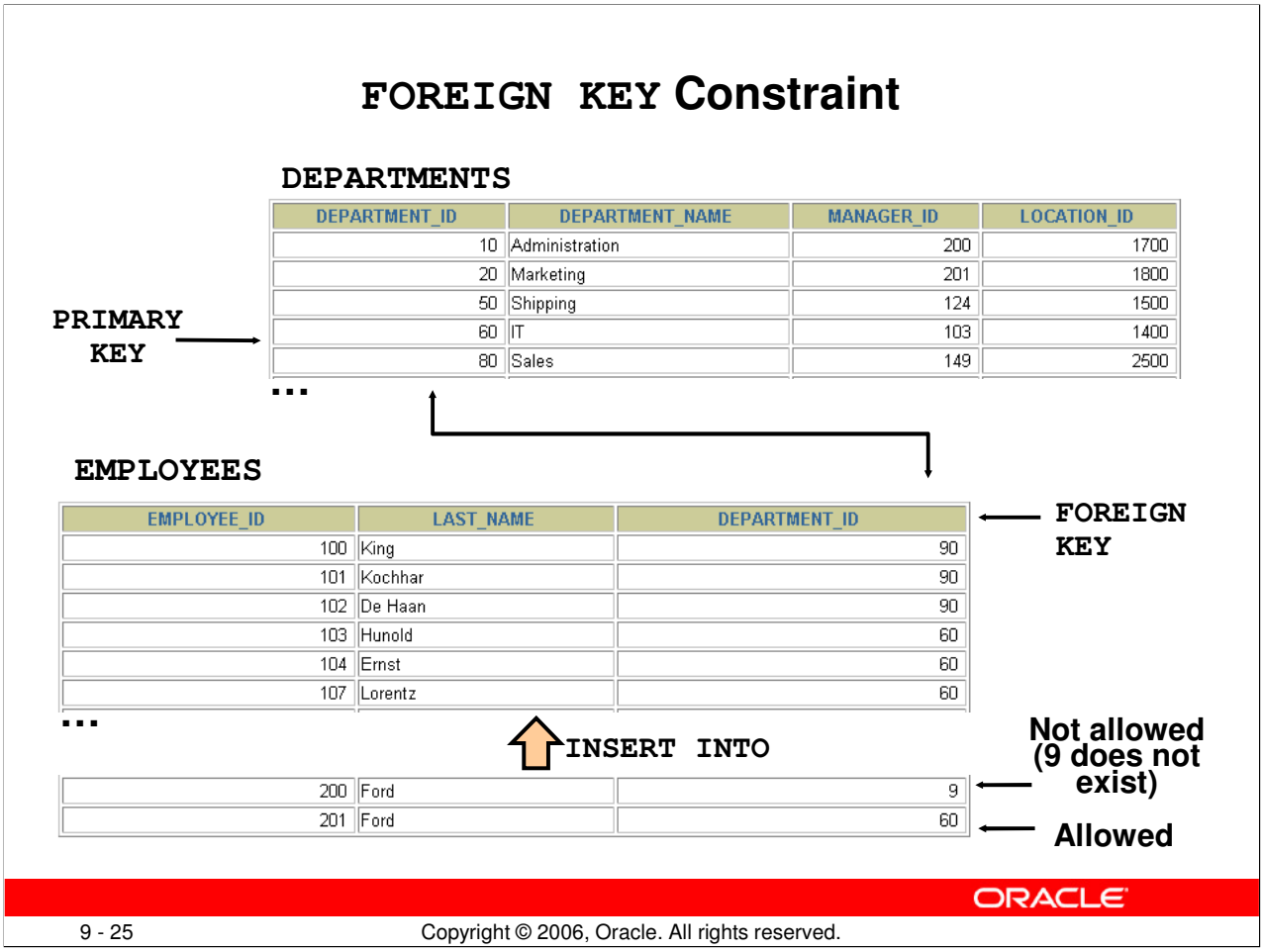
PRIMARY KEY Constraint



PRIMARY KEY Constraint

A PRIMARY KEY constraint creates a primary key for the table. Only one primary key can be created for each table. The PRIMARY KEY constraint is a column or set of columns that uniquely identifies each row in a table. This constraint enforces uniqueness of the column or column combination and ensures that no column that is part of the primary key can contain a null value.

Note: Because uniqueness is part of the primary key constraint definition, the Oracle server enforces the uniqueness by implicitly creating a unique index on the primary key column or columns.



FOREIGN KEY Constraint

The FOREIGN KEY (or referential integrity) constraint designates a column or combination of columns as a foreign key and establishes a relationship between a primary key or a unique key in the same table or a different table.

In the example in the slide, DEPARTMENT_ID has been defined as the foreign key in the EMPLOYEES table (dependent or child table); it references the DEPARTMENT_ID column of the DEPARTMENTS table (the referenced or parent table).

Guidelines

- A foreign key value must match an existing value in the parent table or be NULL.
- Foreign keys are based on data values and are purely logical, rather than physical, pointers.

FOREIGN KEY Constraint

Defined at either the table level or the column level:

```
CREATE TABLE employees(
    employee_id      NUMBER(6),
    last_name        VARCHAR2(25) NOT NULL,
    email            VARCHAR2(25),
    salary           NUMBER(8,2),
    commission_pct   NUMBER(2,2),
    hire_date        DATE NOT NULL,
...
    department_id    NUMBER(4),
    CONSTRAINT emp_dept_fk FOREIGN KEY (department_id)
      REFERENCES departments(department_id),
    CONSTRAINT emp_email_uk UNIQUE(email));
```

ORACLE

FOREIGN KEY Constraint (continued)

FOREIGN KEY constraints can be defined at the column or table constraint level. A composite foreign key must be created by using the table-level definition.

The example in the slide defines a FOREIGN KEY constraint on the DEPARTMENT_ID column of the EMPLOYEES table, using table-level syntax. The name of the constraint is EMP_DEPTID_FK.

The foreign key can also be defined at the column level, provided the constraint is based on a single column. The syntax differs in that the keywords FOREIGN KEY do not appear. For example:

```
CREATE TABLE employees
(...
department_id NUMBER(4) CONSTRAINT emp_deptid_fk
REFERENCES departments(department_id),
...
)
```

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**

ORACLE

FOREIGN KEY Constraint: Keywords

The foreign key is defined in the child table, and the table containing the referenced column is the parent table. The foreign key is defined using a combination of the following keywords:

- `FOREIGN KEY` is used to define the column in the child table at the table-constraint level.
- `REFERENCES` identifies the table and column in the parent table.
- `ON DELETE CASCADE` indicates that when the row in the parent table is deleted, the dependent rows in the child table are also deleted.
- `ON DELETE SET NULL` converts foreign key values to null when the parent value is removed.

The default behavior is called the *restrict rule*, which disallows the update or deletion of referenced data.

Without the `ON DELETE CASCADE` or the `ON DELETE SET NULL` options, the row in the parent table cannot be deleted if it is referenced in the child table.

CHECK Constraint

- Defines a condition that each row must satisfy
- The following expressions are not allowed:
  - References to CURRVAL, NEXTVAL, LEVEL, and ROWNUM pseudocolumns
  - Calls to SYSDATE, UID, USER, and USERENV functions
  - Queries that refer to other values in other rows

```
..., salary   NUMBER(2)
     CONSTRAINT emp_salary_min
            CHECK (salary > 0),...
```

ORACLE

CHECK Constraint

The CHECK constraint defines a condition that each row must satisfy. The condition can use the same constructs as query conditions, with the following exceptions:

- References to the CURRVAL, NEXTVAL, LEVEL, and ROWNUM pseudocolumns
- Calls to SYSDATE, UID, USER, and USERENV functions
- Queries that refer to other values in other rows

A single column can have multiple CHECK constraints that refer to the column in its definition. There is no limit to the number of CHECK constraints that you can define on a column.

CHECK constraints can be defined at the column level or table level.

```
CREATE TABLE employees
   (...
    salary NUMBER(8,2) CONSTRAINT emp_salary_min
                       CHECK (salary > 0),
   ...
```

CREATE TABLE: Example

```
CREATE TABLE employees
    ( employee_id    NUMBER(6)
        CONSTRAINT     emp_employee_id   PRIMARY KEY
    , first_name     VARCHAR2(20)
    , last_name      VARCHAR2(25)
        CONSTRAINT     emp_last_name_nn  NOT NULL
    , email          VARCHAR2(25)
        CONSTRAINT     emp_email_nn      NOT NULL
        CONSTRAINT     emp_email_uk      UNIQUE
    , phone_number   VARCHAR2(20)
    , hire_date      DATE
        CONSTRAINT     emp_hire_date_nn  NOT NULL
    , job_id         VARCHAR2(10)
        CONSTRAINT     emp_job_nn        NOT NULL
    , salary         NUMBER(8,2)
        CONSTRAINT     emp_salary_ck     CHECK (salary>0)
    , commission_pct NUMBER(2,2)
    , manager_id     NUMBER(6)
    , department_id  NUMBER(4)
        CONSTRAINT     emp_dept_fk       REFERENCES
           departments (department_id));
```

ORACLE

The CREATE TABLE Example

The example shows the statement used to create the EMPLOYEES table in the HR schema.

Violating Constraints

```
UPDATE employees
SET    department_id = 55
WHERE  department_id = 110;
```

```
UPDATE employees
       *
ERROR at line 1:
ORA-02291: integrity constraint (HR.EMP_DEPT_FK)
violated - parent key not found
```

Department 55 does not exist.

ORACLE

Integrity Constraint Error

When you have constraints in place on columns, an error is returned to you if you try to violate the constraint rule.

For example, if you attempt to update a record with a value that is tied to an integrity constraint, an error is returned.

In the example in the slide, department 55 does not exist in the parent table, `DEPARTMENTS`, and so you receive the *parent key* violation `ORA-02291`.

Violating Constraints

You cannot delete a row that contains a primary key that is used as a foreign key in another table.

```
DELETE FROM departments
WHERE       department_id = 60;
```

```
DELETE FROM departments
            *
ERROR at line 1:
ORA-02292: integrity constraint (HR.EMP_DEPT_FK)
violated - child record found
```

ORACLE

Integrity Constraint Error (continued)

If you attempt to delete a record with a value that is tied to an integrity constraint, an error is returned.

The example in the slide tries to delete department 60 from the `DEPARTMENTS` table, but it results in an error because that department number is used as a foreign key in the `EMPLOYEES` table. If the parent record that you attempt to delete has child records, then you receive the *child record found* violation `ORA-02292`.

The following statement works because there are no employees in department 70:

```
DELETE FROM  departments
WHERE        department_id = 70;

1 row deleted.
```

Creating a Table by Using a Subquery

- **Create a table and insert rows by combining the `CREATE TABLE` statement and the `AS subquery` option.**

```
CREATE TABLE table
            [(column, column...)]
AS subquery;
```

- **Match the number of specified columns to the number of subquery columns.**
- **Define columns with column names and default values.**

ORACLE

Creating a Table from Rows in Another Table

A second method for creating a table is to apply the `AS subquery` clause, which both creates the table and inserts rows returned from the subquery.

In the syntax:

`table`	is the name of the table
`column`	is the name of the column, default value, and integrity constraint
`subquery`	is the `SELECT` statement that defines the set of rows to be inserted into the new table

Guidelines

- The table is created with the specified column names, and the rows retrieved by the `SELECT` statement are inserted into the table.
- The column definition can contain only the column name and default value.
- If column specifications are given, the number of columns must equal the number of columns in the subquery `SELECT` list.
- If no column specifications are given, the column names of the table are the same as the column names in the subquery.
- The column data type definitions and the NOT NULL constraint are passed to the new table. The other constraint rules are not passed to the new table. However, you can add constraints in the column definition.

Creating a Table by Using a Subquery

```
CREATE TABLE  dept80
  AS
    SELECT   employee_id, last_name,
             salary*12 ANNSAL,
             hire_date
    FROM     employees
    WHERE    department_id = 80;
Table created.
```

```
DESCRIBE dept80
```

Name	Null?	Type
EMPLOYEE_ID		NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
ANNSAL		NUMBER
HIRE_DATE	NOT NULL	DATE

ORACLE

Creating a Table from Rows in Another Table (continued)

The slide example creates a table named `DEPT80`, which contains details of all the employees working in department 80. Notice that the data for the `DEPT80` table comes from the `EMPLOYEES` table.

You can verify the existence of a database table and check column definitions by using the *i*SQL*Plus `DESCRIBE` command.

Be sure to provide a column alias when selecting an expression. The expression `SALARY*12` is given the alias `ANNSAL`. Without the alias, the following error is generated:

```
ERROR at line 3:
ORA-00998: must name this expression with a column alias
```

ALTER TABLE Statement

Use the ALTER TABLE statement to:

- **Add a new column**
- **Modify an existing column**
- **Define a default value for the new column**
- **Drop a column**

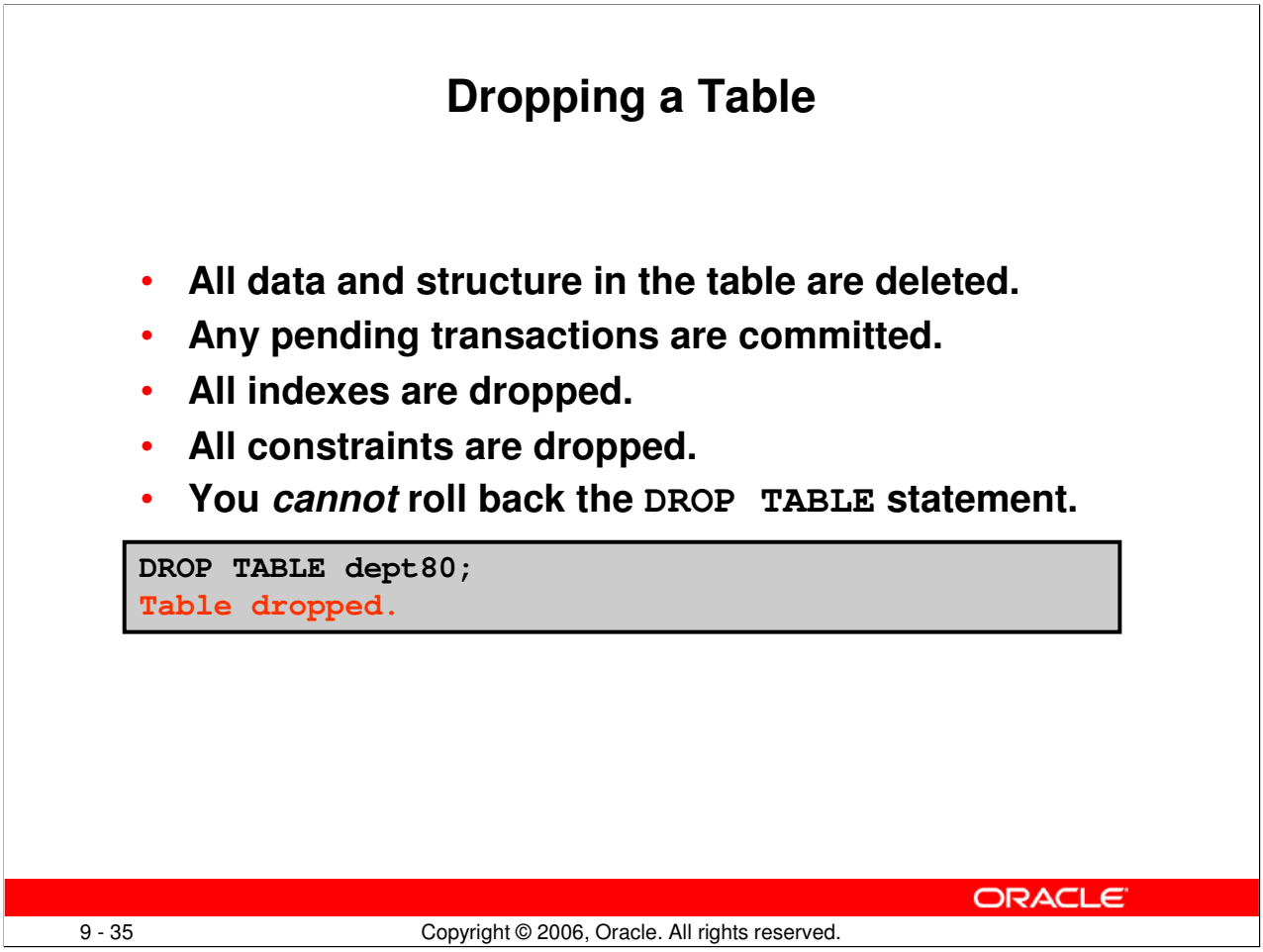
ORACLE

ALTER TABLE Statement

After you create a table, you may need to change the table structure for any of the following reasons:

- You omitted a column.
- Your column definition needs to be changed.
- You need to remove columns.

You can do this by using the `ALTER TABLE` statement. For information about the `ALTER TABLE` statement, see the *Oracle Database 10g SQL Fundamentals II* course.



Dropping a Table

The DROP TABLE statement removes the definition of an Oracle table. When you drop a table, the database loses all the data in the table and all the indexes associated with it.

Syntax

```
DROP TABLE table
```

In the syntax, *table* is the name of the table.

Guidelines

- All data is deleted from the table.
- Any views and synonyms remain but are invalid.
- Any pending transactions are committed.
- Only the creator of the table or a user with the DROP ANY TABLE privilege can remove a table.

Note: The DROP TABLE statement, once executed, is irreversible. The Oracle server does not question the action when you issue the DROP TABLE statement. If you own that table or have a high-level privilege, then the table is immediately removed. As with all DDL statements, DROP TABLE is committed automatically.

Summary

In this lesson, you should have learned how to use the `CREATE TABLE` statement to create a table and include constraints.

- **Categorize the main database objects**
- **Review the table structure**
- **List the data types that are available for columns**
- **Create a simple table**
- **Explain how constraints are created at the time of table creation**
- **Describe how schema objects work**

ORACLE

Summary

In this lesson, you should have learned how to do the following:

`CREATE TABLE`

- Use the `CREATE TABLE` statement to create a table and include constraints.
- Create a table based on another table by using a subquery.

`DROP TABLE`

- Remove rows and a table structure.
- Once executed, this statement cannot be rolled back.

Practice 9: Overview

This practice covers the following topics:

- **Creating new tables**
- **Creating a new table by using the `CREATE TABLE AS` syntax**
- **Verifying that tables exist**
- **Dropping tables**

ORACLE

Practice 9: Overview

Create new tables by using the `CREATE TABLE` statement. Confirm that the new table was added to the database. Create the syntax in the command file, and then execute the command file to create the table.

Practice 9

1. Create the DEPT table based on the following table instance chart. Place the syntax in a script called lab_09_01.sql, and then execute the statement in the script to create the table. Confirm that the table is created.

Column Name	ID	NAME
Key Type	Primary key	
Nulls/Unique		
FK Table		
FK Column		
Data type	NUMBER	VARCHAR2
Length	7	25

Name	Null?	Type
ID		NUMBER(7)
NAME		VARCHAR2(25)

2. Populate the DEPT table with data from the DEPARTMENTS table. Include only columns that you need.
3. Create the EMP table based on the following table instance chart. Place the syntax in a script called lab_09_03.sql, and then execute the statement in the script to create the table. Confirm that the table is created.

Column Name	ID	LAST_NAME	FIRST_NAME	DEPT_ID
Key Type				
Nulls/Unique				
FK Table				DEPT
FK Column				ID
Data type	NUMBER	VARCHAR2	VARCHAR2	NUMBER
Length	7	25	25	7

Name	Null?	Type
ID		NUMBER(7)
LAST_NAME		VARCHAR2(25)
FIRST_NAME		VARCHAR2(25)
DEPT_ID		NUMBER(7)

Practice 9 (continued)

4. Create the EMPLOYEES2 table based on the structure of the EMPLOYEES table. Include only the EMPLOYEE_ID, FIRST_NAME, LAST_NAME, SALARY, and DEPARTMENT_ID columns. Name the columns in your new table ID, FIRST_NAME, LAST_NAME, SALARY, and DEPT_ID, respectively.
5. Drop the EMP table.