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

Database Objects

An Oracle Database can contain multiple data structures.

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 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.

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
  - Must be 1–30 characters long
  - Must contain only A–Z, a–z, 0–9, _, $, and #
  - Must not duplicate the name of another object owned by the same user
  - 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`.

`CREATE TABLE` Statement

- You create tables to store data by executing the SQL `CREATE TABLE` statement.
- These statements have an immediate effect on the database, and they also record information in the data dictionary.
  - You must have:
    - The `CREATE TABLE` privilege
    - A storage area

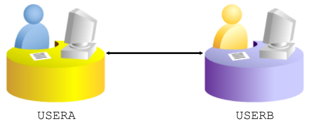
```
CREATE TABLE [schema.]table
          (column datatype [DEFAULT expr][, ...]);
```

- 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

- 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.
  - 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.



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

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

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.

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

  - Literal values, expressions, or SQL functions are legal values.
  - Another column's name or a pseudo column 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);
CREATE TABLE succeeded.
```

Creating Tables

- 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.

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

- Confirm table creation.

```
DESCRIBE dept
```

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
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

Data Types

- 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

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.
- 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

Datetime Data Types

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

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

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

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.):

Including 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

- 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`

- **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.
  - Define a constraint at the column or table level.
  - View a constraint in the data dictionary.

Defining Constraints

- Syntax:

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

- Column-level constraint:
  - Constraints defined at the column level are included when the column is defined

```
column [CONSTRAINT constraint_name] constraint_type,
```

- Table-level constraint:
  - Table-level constraints are defined at the end of the table definition

```
column,...
  [CONSTRAINT constraint_name] constraint_type
  (column, ...),
```

Defining Constraints

- 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

- 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.
  - Column-level constraint:

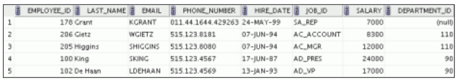
```
CREATE TABLE employees(
  employee_id  NUMBER(6)
    CONSTRAINT emp_emp_id_pk PRIMARY KEY,
  first_name   VARCHAR2(20),
  ...);
```

  - 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));
```

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.



	EMPLOYEE_ID	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID
1	178	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	7000	(null)
2	206	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	8300	110
3	205	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000	110
4	100	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000	90
5	102	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000	90

...

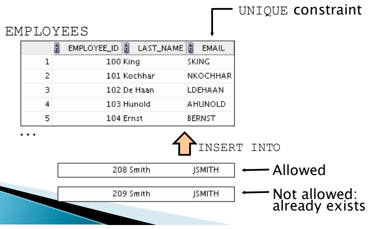
NOT NULL constraint (No row can contain a null value for this column.)

NOT NULL constraint

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

UNIQUE Constraint

- UNIQUE Constraint
  - A UNIQUE key integrity constraint requires that every value in a column or set of columns (key) be
  - 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.



UNIQUE constraint

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	EMAIL
1	100	King	SKING
2	101	Kochhar	NKOCHHAR
3	102	De Haan	LDEHAAN
4	103	Hunold	AHUNOLD
5	104	Ernst	BERNST

...

INSERT INTO

208	Smith	JSMITH

Allowed

209	Smith	JSMITH

Not allowed: already exists

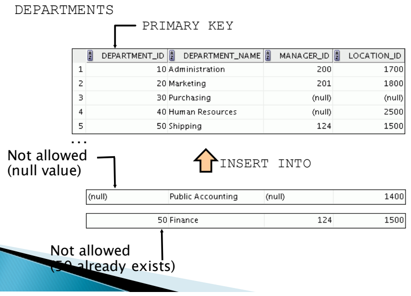
UNIQUE Constraint

- 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.

```
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));
```

PRIMARY KEY Constraint

- A PRIMARY KEY constraint creates a primary key for the table.
- Only one primary key can be created for each table.



DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	30	Purchasing	(null)	(null)
4	40	Human Resources	(null)	2500
5	50	Shipping	124	1500

...

INSERT INTO

(null)	Public Accounting	(null)	1400
50	Finance	124	1500

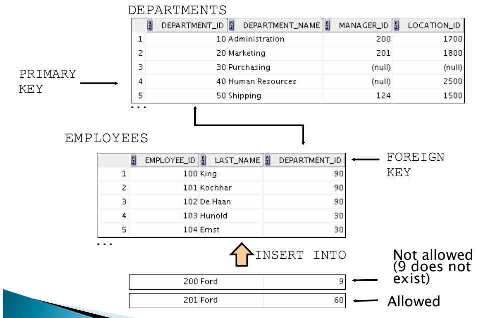
FOREIGN KEY Constraint

- 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



DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	30	Purchasing	(null)	(null)
4	40	Human Resources	(null)	2500
5	50	Shipping	124	1500

...

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	100	King	90
2	101	Kochhar	90
3	102	De Haan	90
4	103	Hunold	30
5	104	Ernst	30

...

INSERT INTO

200	Ford	9
201	Ford	60

FOREIGN KEY Constraint

- 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.

```
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));
```

FOREIGN KEY Constraint:Keywords

- The foreign key is defined in the child table, and the table containing the referenced column is the parent table.
- 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

- The 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),...
```

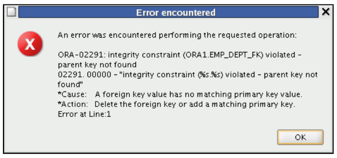
CREATE TABLE: Example

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

```
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));
```

Violating Constraints

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



Error encountered

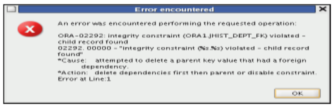
An error was encountered performing the requested operation:

ORA-02291: integrity constraint (ORA1.EMP_DEPT_FK) violated - parent key not found
02291. 00000 - "integrity constraint (%s.%s) violated - parent key not found"
*Cause: A foreign key value has no matching primary key value.
*Action: Delete the foreign key or add a matching primary key.
Error at Line:1

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

- 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;
```



Error encountered

An error was encountered performing the requested operation:

ORA-02292: integrity constraint (ORA1.JHIST_DEPT_FK) violated - child record found
02292. 00000 - "integrity constraint (%s.%s) violated - child record found"
*Cause: attempted to delete a parent key value that had a foreign
        dependency.
*Action: delete dependencies first then parent or disable constraint.
Error at Line:1

- 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.

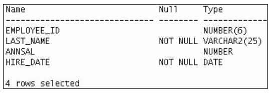
Creating a Table by Using a Subquery

- 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.

- Creates a table named DEPT80, which contains details of all the employees working in department 80.

```
CREATE TABLE  dept80
  AS
    SELECT  employee_id, last_name,
            salary*12 ANNSAL,
            hire_date
    FROM    employees
    WHERE   department id = 80;
CREATE TABLE Succeeded.
```

```
DESCRIBE dept80
```



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

4 rows selected
```

- The expression SALARY*12 is given the alias ANNSAL. Without the alias, the following error is generated

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.

ALTER TABLE Statement

- Use the ALTER TABLE statement to add, modify, or drop columns:

```
ALTER TABLE table
ADD          (column datatype [DEFAULT expr]
             [, column datatype]...);
```

```
ALTER TABLE table
MODIFY       (column datatype [DEFAULT expr]
             [, column datatype]...);
```

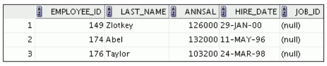
```
ALTER TABLE table
DROP        (column);
```

Adding a Column

- You use the ADD clause to add columns:

```
ALTER TABLE dept80
ADD          (job_id VARCHAR2(9));
ALTER TABLE succeeded.
```

- The new column becomes the last column:



	EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE	JOB_ID
1	149	Zlotkey	126000	29-JAN-00	(null)
2	174	Abel	132000	11-MAY-96	(null)
3	176	Taylor	103200	24-MAR-98	(null)

- You cannot specify where the column is to appear. The new column becomes the last column.
- **Note**: If a table already contains rows when a column is added, then the new column is initially null for all the rows. You cannot add a mandatory NOT NULL column to a table that contains data in the other columns. You can only add a NOT NULL column to an empty table.

Modifying a Column

- You can modify a column definition by using the ALTER TABLE statement with the MODIFY clause.
- Column modification can include changes to a column's data type, size, and default value.

```
ALTER TABLE dept80
MODIFY       (last_name VARCHAR2(30));
ALTER TABLE succeeded.
```

- A change to the default value affects only subsequent insertions to the table.

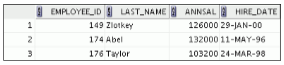
Modifying a Column:Guidelines

- You can increase the width or precision of a numeric column.
- You can increase the width of numeric or character columns.
- You can decrease the width of a column if:
  - The column contains only null values
  - The table has no rows
  - The decrease in column width is not less than the existing values in that column
- You can change the data type if the column contains only null values. The exception to this is CHAR-to-VARCHAR2 conversions, which can be done with data in the columns.
- You can convert a CHAR column to the VARCHAR2 data type or convert a VARCHAR2 column to the CHAR data type only if the column contains null values or if you do not change the size.
- A change to the default value of a column affects only subsequent insertions to the table.

Dropping a Column

- Use the DROP COLUMN clause to drop columns you no longer need from the table:

```
ALTER TABLE  dept80
DROP COLUMN  job_id;
ALTER TABLE succeeded.
```



	EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE
1	149	Zlotkey	126000	29-JAN-00
2	174	Abel	132000	11-MAY-96
3	176	Taylor	103200	24-MAR-98

Dropping a Column: Guidelines

- Guidelines
  - The column may or may not contain data.
  - Using the ALTER TABLE statement, only one column can be dropped at a time.
  - The table must have at least one column remaining in it after it is altered.
  - After a column is dropped, it cannot be recovered.
  - A column cannot be dropped if it is part of a constraint or part of an index key unless the cascade option is added.
  - Dropping a column can take a while if the column has a large number of values. In this case, it may be better to set it to be unused and drop it when there are fewer users on the system to avoid extended locks.
- Note: Certain columns can never be dropped such as columns that form part of the partitioning key of a partitioned table or columns that form part of the primary key of an index-organized table.

SET UNUSED Option

- You use the SET UNUSED option to mark one or more columns as unused.
- You use the DROP UNUSED COLUMNS option to remove the columns that are marked as unused.

```
ALTER TABLE     <table_name>
SET   UNUSED(<column_name>);
OR
ALTER TABLE  <table_name>
SET   UNUSED COLUMN <column_name>;
```

```
ALTER TABLE <table_name>
DROP  UNUSED COLUMNS;
```

- After a column has been marked as unused, you have no access to that column.
- A SELECT * query will not retrieve data from unused columns.
- You can add to the table a new column with the same name as an unused column.

- DROP UNUSED COLUMNS removes from the table all columns currently marked as unused.
- You can use this statement when you want to reclaim the extra disk space from unused columns in the table.
- If the table contains no unused columns, the statement returns with no errors.

```
ALTER TABLE  dept80
SET   UNUSED (last_name);
ALTER TABLE succeeded.

ALTER TABLE  dept80
DROP  UNUSED COLUMNS;
ALTER TABLE succeeded.
```

Dropping a Table

- All data and structure in the table are deleted.
- Any pending transactions are committed.
- All indexes are dropped.
- All constraints are dropped.
- You *cannot* roll back the DROP TABLE statement.

```
DROP TABLE dept80;
DROP TABLE succeeded.
```

Adding a Constraint Syntax

- You can add a constraint for existing tables by using the ALTER TABLE statement with the ADD clause.

```
ALTER TABLE  <table_name>
ADD [CONSTRAINT <constraint_name>]
type (<column_name>);
```

- In the syntax:

table	Is the name of the table
constraint	Is the name of the constraint
type	Is the constraint type
column	Is the name of the column affected by the constraint

- The constraint name syntax is optional, although recommended. If you do not name your constraints, the system generates constraint names.

Adding a Constraint Syntax

- Guidelines
  - You can add, drop, enable, or disable a constraint, but you cannot modify its structure.
  - You can add a NOT NULL constraint to an existing column by using the MODIFY clause of the ALTER TABLE statement.
- Note: You can define a NOT NULL column only if the table is empty or if the column has a value for every row.

Adding a Constraint

- Add a FOREIGN KEY constraint to the EMP2 table indicating that a manager must already exist as a valid employee in the EMP2 table.

```
ALTER TABLE emp2
modify employee_id Primary Key;
ALTER TABLE succeeded.
```

```
ALTER TABLE emp2
ADD CONSTRAINT emp_mgr_fk
  FOREIGN KEY(manager_id)
  REFERENCES emp2(employee_id);
ALTER TABLE succeeded.
```

ON DELETE CASCADE

- The ON DELETE CASCADE action allows parent key data that is referenced from the child table to be deleted, but not updated.
- When data in the parent key is deleted, all rows in the child table that depend on the deleted parent key values are also deleted.
- To specify this referential action, include the ON DELETE CASCADE option in the definition of the FOREIGN KEY constraint.

```
ALTER TABLE Emp2 ADD CONSTRAINT emp_dt_fk
FOREIGN KEY (Department_id)
REFERENCES departments ON DELETE CASCADE);
ALTER TABLE succeeded.
```

Dropping a Constraint

- To drop a constraint, you can identify the constraint name from the USER_CONSTRAINTS and USER_CONS_COLUMNS data dictionary views.
- Then use the ALTER TABLE statement with the DROP clause.
- The CASCADE option of the DROP clause causes any dependent constraints also to be dropped.

```
ALTER TABLE table
 DROP  PRIMARY KEY | UNIQUE (column) |
       CONSTRAINT   constraint   [CASCADE];
```

- In the syntax:

table	Is the name of the table
column	Is the name of the column affected by the constraint
constraint	Is the name of the constraint

- When you drop an integrity constraint, that constraint is no longer enforced by the Oracle server and is no longer available in the data dictionary.

Dropping a Constraint

- Remove the manager constraint from the EMP2 table:

```
ALTER TABLE emp2
DROP CONSTRAINT emp_mgr_fk;
ALTER TABLE succeeded.
```

- Remove the PRIMARY KEY constraint on the DEPT2 table and drop the associated FOREIGN KEY constraint on the EMP2.DEPARTMENT_ID column:

```
ALTER TABLE dept2
DROP PRIMARY KEY CASCADE;
ALTER TABLE succeeded.
```

Disabling Constraints

- You can disable a constraint without dropping it or re-creating it by using the ALTER TABLE statement with the DISABLE clause.

```
ALTER   TABLE   table
  DISABLE CONSTRAINT constraint [CASCADE];
```

- **Guidelines**
  - You can use the DISABLE clause in both the CREATE TABLE statement and the ALTER TABLE statement.
  - The CASCADE clause disables dependent integrity constraints.
  - Disabling a unique or primary key constraint removes the unique index.

```
ALTER TABLE emp2
DISABLE CONSTRAINT emp_dt_fk;
ALTER TABLE succeeded.
```

Enabling Constraints

- You can enable a constraint without dropping it or re-creating it by using the ALTER TABLE statement with the ENABLE clause.

```
ALTER   TABLE       table
ENABLE  CONSTRAINT constraint;
```

- **Guidelines**
  - If you enable a constraint, that constraint applies to all the data in the table. All the data in the table must comply with the constraint.
  - If you enable a UNIQUE key or a PRIMARY KEY constraint, a UNIQUE or PRIMARY KEY index is created automatically. If an index already exists, then it can be used by these keys.
  - You can use the ENABLE clause in both the CREATE TABLE statement and the ALTER TABLE statement.

```
ALTER TABLE        emp2
ENABLE CONSTRAINT emp_dt_fk;
ALTER TABLE succeeded.
```