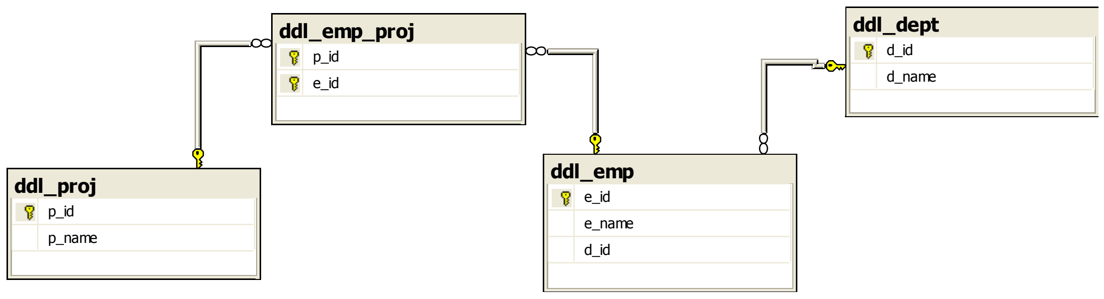
Table of Contents

These queries provide a way to use SQL to create relations, change their design, set relationships, create and drop indexes. These statements are considered Data Definition Language statements. This section shows the queries need to create the relations and relationships shown below.



Some of the demo code will create tables that are used to illustrate a technique. You can delete those tables at the end of the demos for this week. These table names will all start ddl_

1. Removing a table

You can drop a table and all of its data with the Drop Table query. The DBMS does not ask you if you are sure that you want to drop the table.

Demo 01: Dropping a table

```
   DROP TABLE ddl_emp_proj ;;
```

```
Table dropped.
```

If you try to drop a table that does not exist you will get an error message. In Oracle this error does not stop a script that is running. So you can have drop table commands at the start of a script that creates tables. It is your responsibility to be certain that you do not drop the wrong tables in error.

```
   DROP TABLE ddl_emp_proj ;
   DROP TABLE ddl_proj ;
   DROP TABLE ddl_emp ;
   DROP TABLE ddl_dept ;
```

Sometimes you may find that you cannot drop a table if the table is involved in a relationship as the parent. You cannot drop the parent table if there are rows in the child table. The above set of drops starts with the child table and then works up to the parent tables.

2. Simple table creation

The minimal SQL statement to create a table requires that you give the table a name and that you give each column a name and a data type. The table has to have at least one column defined. This statement does not set a primary key, default value, foreign key references, or other rules about the data- commonly called constraints. All this SQL statement does is create the table structure.

You cannot have two tables with the same name, so if you are experimenting with these statements you might need to drop one or more of these tables. Oracle also enforces rules about dropping tables that are involved in relationships.

Demo 02: Minimal statement to create a table with two columns

```
   CREATE TABLE ddl_dept (
      d_id      number(3)
    , d_name    varchar2(15));
```

```
Table created.
```

When you declare a data type for a column, you limit the values that can be entered in that column. In the above table, the values for d_name are limited to 15 characters. Oracle will not truncate string values to fit the defined data type precisions. The values for the d_id values must be in the range -999 to +999 due to the number (3) designation. Oracle will round decimal values to fit the precisions defined for a numeric type- so this will accept the value 123.5 and round it to 124 but it will not accept the value 1234.

2.1. Add a single new row using a value list

In order to understand table creation you need to test with insert statements. This is a model to insert a single row into the table specifying the column names. This is the preferred technique.

```
      insert into my_table (col1_name, col2_name,  . . )
                 values   (value1_name, value2_name,  . .);
```

We will look at variations of the insert statement .

Demo 03: These are examples of Insert statements that may work- or not

Insert statement that works

```
   insert into ddl_dept (d_id, d_name)
   values(10,'Sales');
```

Insert statement that works; the default is that the columns can accept nulls

```
   insert into ddl_dept (d_id, d_name)
   Values(null,null);
```

Insert statement that works; we did not set a primary key so we can have two rows with the same value for d_id.

```
   insert into ddl_dept (d_id, d_name)
   values(10,'Marketing');
```

Insert statement that works; the value for d_id is rounded to 41. This is an implied cast.

```
   insert into ddl_dept (d_id, d_name)
   values(40.8,'Development');
```

Insert statement that works; We did not restrict negative numbers

```
   insert into ddl_dept (d_id, d_name)
   values(-44,'');
```

Insert statement that fails; value for d_id is larger than specified precision allowed for this column

```
   insert into ddl_dept (d_id, d_name)
   values(1234,'Research');
```

Insert statement that fails; value for d_name is too large for column (actual: 22, maximum: 15)

```
   insert into ddl_dept (d_id, d_name)
   values(20,'Accounting and Payroll');
```

Insert statement that fails; not enough values; we specified two columns in the list but provided only one value. Note that this does not use a null for the missing value.

```
   insert into ddl_dept (d_id, d_name)
   values(35);
```

Insert statement that works; this specifies one column in the list and we provide one value. In this case, d_name gets a null entry.

```
   insert into ddl_dept (d_id)
   values(35);

   select * from ddl_dept order by d_id;
```

```
D_ID                   D_NAME
---------------------- ---------------
-44
10                     Sales
10                     Marketing
35
41                     Development


6 rows selected
```

Drop the table ddl_dept. We need a better version of this table.

```
   DROP TABLE ddl_dept ;
```

3. Renaming a Table

You can rename a table and Oracle will update constraints and fk relationships relating to the table- but will not update the table name in views, script file etc. This is a fairly dangerous change to make to a working system.

Demo 04: Renaming a table

```
   RENAME zoo  TO sfzoo;
```

```
Table renamed.
```

4. Creating Tables with Integrity

You want your database to have integrity- you want to ensure that as far as possible the values in your data are correct. There are three types of integrity:

- Entity integrity- each row should be uniquely identifiable- enforced by primary key constraints
- Referential integrity- a row in a child table needs to be related to a row in a parent table- enforced by foreign key constraints.
- Domain integrity- a data value meets certain criteria- in its simplest form, this is enforced by the data type; this can be more closely defined by a check constraint.

To create a usable database you would want to set these constraints— rules about what data values can be entered in the tables and how the tables are related to each other. Each constraint is an object and has a name.

Oracle will create constraint names for you if you do not create your own names. You will see many table creation statements that do not create constraint names and that let Oracle create the names. This approach will make it harder to manipulate the constraint later. Constraint names should reflect the table name, the attribute name, and the type of constraint. Constraint names follow the normal Oracle rules for identifiers and are also limited to 30 characters. The pattern I follow of ending constraint names with tags such as _pk, fk etc is common but is not a syntax requirement.

Constraint names must be unique within the schema.

A constraint can be declared on the same line as the column declaration (a column constraint); the constraint may be declared in the create statement after all of the columns are declared (a table constraint). You can also add constraints to an existing table.

At a minimum, when you create a table, you would set the primary key and designate which attributes cannot be left empty.

Constraints that you create when you create the table, or that you add to the table, are enforced by the DBMS. These are called declarative constraints. Other rules about data can be enforced with triggers- these are called procedural integrity constraints. We are not discussing triggers in this class. (see CS151P for triggers.)

4.1. Primary key, not null, and unique constraints

Demo 05: Creates the project table; sets the pk and a unique constraint. Chose a constraint name that reflects its purpose.

```
CREATE TABLE  ddl_proj (
   p_id   varchar2(10)    constraint  ddl_proj_PK  primary key
 , p_name varchar2(15)    null
                          constraint ddl_proj_P_Name_UN  unique
);
```

Demo 06: Creates the department table; sets the pk and a unique constraint If you created this table earlier be sure to drop the previous version.

```
CREATE TABLE  ddl_dept(
    d_id     number(3)    constraint ddl_dept_PK  primary key
  , d_name   varchar2(15) not null
                          constraint ddl_dept_D_Name_UN  unique
);
```

Primary Key

Setting a primary key constraint also means that the attribute cannot be null and must have unique values with the table.

You can declare a primary key constraint on sets of columns using a table constraint. In that case use the syntax shown here where col1 and col2 are already defined as columns in the table.

```
constraint constraint_name_PK   primary key (col1, col2)
```

Null, Not Null

The default setting for nullability in most dbms is that the column can accept null values. Since defaults can be changed, it is a good idea to always specify Null or Not Null for attributes. The NOT NULL constraint is generally not given a constraint name. This constraint can be set only as a column constraint.

Unique

The unique constraint means that no two rows in the table can have the same value for that attribute . Setting a column to be Not Null and Unique means that it is an identifier and is a candidate key for this table. You can have only one primary key defined per table; you can define as many candidate keys as you need for a table.

When you define an attribute to be null and unique, Oracle will let you have multiple rows in the table with a null value for that column. This agrees with the attitude that a null does not equal another null.

You can declare a unique constraint on sets of columns using a table constraint. In that case use the syntax shown here where col1 and col2 are already defined as columns in the table.

```
   constraint constraint_name_UN   unique(col1, col2)
```

Demo 07: The Unique constraint on a two column set

```
   create table ddl_un (id number(3)
   , city varchar2(15), state char(2)
   , constraint location_UN  unique(city, state)
   );
```

The first three inserts are ok

```
   Insert into ddl_un (id, city, state) values (1, 'Chicago', 'IL');
   Insert into ddl_un (id, city, state) values (2, 'Chicago', 'CA');
   Insert into ddl_un (id, city, state) values (3, 'Pekin', 'IL');
```

This one fails because it is a duplicate of (city, state)

```
   Insert into ddl_un (id, city, state) values (4, 'Pekin', 'IL');
ORA-00001: unique constraint (ROSE151A.LOCATION_UN) violated
```

What about nulls? These are both allowed; one null is not equal to another null.

```
   Insert into ddl_un (id, city, state) values (5, null, null);
   Insert into ddl_un (id, city, state) values (6, null, null);
```

```
ID                     CITY            STATE
---------------------- --------------- -----
1                      Chicago         IL
2                      Chicago         CA
3                      Pekin           IL
5
6
```

But the second of this pair of inserts is blocked.

```
   Insert into ddl_un (id, city, state) values (7, null, 'NY');
   Insert into ddl_un (id, city, state) values (8, null, 'NY');

   drop table ddl_un;
```

To satisfy a unique constraint, no two rows in the table can have the same value for the unique key. However, the unique key made up of a single column can contain nulls. To satisfy a composite unique key, no two rows in the table or view can have the same combination of values in the key columns. Any row that contains nulls in all key columns automatically satisfies the constraint. However, two rows that contain nulls for one or more key columns and the same combination of values for the other key columns violate the constraint.

4.2. Foreign key constraints

The department table and the project table are parent tables and can be created independently. The Employee table is a child of the Department table, therefore you have to make the Department table before the Employee table. The referenced attribute in the parent table must have been defined as the PK or with a Unique constraint. This is so that a child row refers to uniquely one parent row.

Demo 08: Creates the employee table and creates the relationship object to the department table.

```
CREATE TABLE  ddl_emp(
  e_id    number(3)    constraint ddl_emp_pk  primary key
, e_name  varchar2(15) not null
, d_id    number(3)    not null constraint ddl_emp_dept_fk references ddl_dept
);
```

Demo 09: Creates the employee project table with a composite primary key. The composite pk has to be defined on a separate line as a table constraint. This also creates the relationships to the employee and project tables.

```
CREATE TABLE  ddl_emp_proj (
  p_id  varchar2(10)not null
        constraint ddl_emp_proj_proj_fk references ddl_proj( p_id)
, e_id  number(3)  not null
        constraint ddl_emp_proj_emp_fk references ddl_emp(e_id)
, constraint  ddl_empproj_pk  primary key  ( p_id, e_id)
);
```

After running these queries, you will have four tables.

Foreign Key

The foreign key constraint specifies the foreign key attribute(s) in the child table and the parent table and column(s) for the relationship. The referenced column(s) in the parent table must have a Unique or Primary Key constraint.

The foreign key constraint is set in the child table. The child claims the parent.

To set this as a column constraint, you use the following

```
myCol  DataType  CONSTRAINT cnstrName REFERENCES parentTbl(col_name)
```

If the referenced key is the primary key of the parent table, you do not have to mention it in the constraint.

It is also possible to skip the keyword constraint and the constraint name- but then you get a system generated name.

```
myCol DataType  REFERENCES parentTbl
```

To set this as a table constraint, you use the following. Since this is a table constraint you have to include the name of the attribute that is the foreign key.

```
CONSTRAINT cnstrName FOREIGN KEY (myCol) REFERENCES parentTbl(col_name)
```

If the foreign key is made up of multiple columns, create a single constraint listing both columns

```
CONSTRAINT cnstrName FOREIGN KEY (myCol1, myCol2)
           REFERENCES parentTbl(col_name1, col_name2 )
```

You can include a rule as to how to handle deletes with On Delete clause

ON DELETE CASCADE, ON DELETE SET NULL: with the foreign key constraint, you can specify what should happen to the child rows if a related parent row is deleted. The default behavior is that you cannot delete a parent row if it has any child rows.

```
constraint cnstrName references parentTbl on delete cascade
constraint cnstrName references parentTbl on delete set null
```

Demo 10: Creates a parent and a child table with cascade delete. These will be very simple table and I am not naming the constraints.

```
CREATE TABLE  ddl_parent (p_id number(3)  primary key);
CREATE TABLE  ddl_child  (c_id number(3)  primary key
                        , fk_id number(3) references ddl_parent
                          on delete cascade);

Insert into ddl_parent(p_id) values (1);
Insert into ddl_parent(p_id) values (2);
Insert into ddl_parent(p_id) values (3);

Insert into ddl_child(c_id, fk_id) values (100,2);
Insert into ddl_child(c_id, fk_id) values (101,2);
Insert into ddl_child(c_id, fk_id) values (103,2);
Insert into ddl_child(c_id, fk_id) values (104,3);

select p_id, c_id, fk_id
from ddl_parent
join ddl_child on ddl_parent.p_id = ddl_child.fk_id;
```

```
      P_ID       C_ID      FK_ID
---------- ---------- ----------
         2        100          2
         2        101          2
         2        103          2
         3        104          3
```

Demo 11: Deleting rows. The parent row with id 2 is related to three child rows.

```
delete from ddl_parent where p_id = 2;
```

```
1 row deleted.
```

If I select the rows in the parent table- one row was removed

```
Select * from ddl_parent;
```

```
      P_ID
----------
         1
         3
```

If I select the rows in the child table- three rows were removed; even though the feedback from Oracle had indicated only one deleted row when I did the original delete.

```
select * from ddl_child;
```

```
      C_ID      FK_ID
---------- ----------
       104          3
```

Demo 12: You can demonstrate this with the set null constraint. Clean up the tables and reinsert the rows.

```
drop table  ddl_child ;
create table  ddl_child  (c_id number(3) primary key
                        , fk_id number(3) references ddl_parent
                          on delete set null);
insert into ddl_parent(p_id) values (2);
insert into ddl_child(c_id, fk_id) values (100,2);
insert into ddl_child(c_id, fk_id) values (101,2);
insert into ddl_child(c_id, fk_id) values (103,2);
insert into ddl_child(c_id, fk_id) values (104,3);
```

```
   delete from ddl_parent where p_id = 2;
```
```
1 row deleted.
```
Now when you look at the rows in the child table the rows are still there but the values for the foreign key that were originally set to 2 are now nulled out. This also requires that the fk_id in the child table was a nullable field.
```
   select * from ddl_child;
```
```
      C_ID      FK_ID
---------- ----------
       100
       101
       103
       104          3
```
```
   drop table  ddl_child ;
   drop table  ddl_parent ;
```

4.3. Other constraint types

DEFAULT: This declaration is used to specify a default value that will be placed in a column.

MyCol DataType Default *default_value*

Demo 13:

```
   create table ddl_default (
       id number(3)
     , d_state char(2) default 'CA'
   );
```

You can use the word default in the insert or skip the column to get the default in the column list.
```
   insert into ddl_default (id, d_state) values (1, 'PA');
   insert into ddl_default (id, d_state) values (2, 'CA');
   insert into ddl_default (id, d_state) values (3, default);
   insert into ddl_default (id)          values (4);
```
```
        ID D_
---------- --
         1 PA
         2 CA
         3 CA
         4 CA
```
```
   drop table ddl_default;
```

CHECK: use this to set other types of tests that a value must meet to be allowed into the table. The CHECK constraint condition must evaluate to TRUE or NULL to be satisfied. It cannot use SYSDATE as part of its test. If you create this as a table constraint, then it can refer to other values in the same column-
such as fld1 <= fld2. If you declare this as a column constraint, then it can refer only to this column.

The syntax for a check constraint is

CONSTRAINT *MyConstraintName* CHECK (*test*)

The test can use equality tests, In, Between. Use the full test expression even if you are declaring this as a column constraint. (List_Price > 15) The test must be enclosed in parentheses.

Demo 14:

```
   create table ddl_check (
       id number(3)
     , d_state char(2) constraint dstated_ck
```

```
                      check(d_state in ('CA','NV','IL'))
   );
```

The first two inserts work.

```
   insert into ddl_check (id, d_state) values (1, 'CA');
   insert into ddl_check (id, d_state) values (2, 'IL');
```

The following insert fails because NY is not in the approved list of values.

```
   Insert into ddl_check (id, d_state) values (3, 'NY');
```

```
ORA-02290: check constraint (ROSE151A.DSTATED_CK) violated
```

The following inserts succeeds. The rule for tests in inserts is different than the rule in Where clauses. If the value to be inserted does not specifically fail the test then it is allowed.

```
   insert into ddl_check (id, d_state) values (4, null);
```

If you really want the value in d_state to be one of those three values you can also set a not null constraint on the column.

```
   Drop table ddl_check;
```

You can create the constraints in the same SQL statement as you use to create the table. You can also use a Create table statement to set the column names and data types and then use Alter Table statements to add the constraints. You can add constraints after data has been added to the table if the existing data meets the constraint rule.

Constraint names need to be unique with your schema.

4.4. Disable and enable constraints

You can disable a constraint and then enable it later. This is sometimes done during maintenance operations.

Demo 15:

```
   alter table ddl_dept_2
   disable constraint DSTATE2_CK;
```

Now I can insert the record from a previous demo that would have been rejected due to the check constraint violation

```
   insert into ddl_dept_2 values(77, 'Payroll', 'Bowling Green', 'OH', NULL);
```

But if I try to enable that constraint, I cannot since there is now data in the table that violates the constraint. The dbms takes the integrity rules seriously! (this behavior varies with the dbms)

```
   alter table ddl_dept_2
   enable     constraint DSTATE2_CK;
```

```
ERROR at line 2:
ORA-02293: cannot validate (ROSE151A.DSTATE2_CK) - check constraint violated
```

You might disable a constraint if you are adding a lot of records that are guaranteed to be clean. It takes time for the dbms to validate the data and it can be more efficient to do all of the validations after the inserts are complete.

4.5. Drop cascade constraints

Having relationship created between tables limits your ability to drop tables. You cannot drop a parent table if there is a related child table.

Demo 16:

```
   drop table ddl_dept;
```

```
ERROR at line 1:
ORA-02449: unique/primary keys in table referenced by foreign keys
```

Oracle has a cascade constraint options that allows you to drop the parent table.

```
   drop table ddl_dept cascade constraints;
```

```
Table dropped.
```

Demo 17: To see how this works, set up a simple pair of parent and child tables.

```
   create table z_parent(pr_pk integer constraint parent_pk primary key
   , col1 integer );
   create table z_child (ch_pk integer constraint child_pk  primary key
   , col2 integer constraint child_fk references z_parent);
```

Display the constraints for these two tables. (this technique is discussed in the data dictionary document)

```
   select table_name, constraint_name, constraint_type, status
   from user_constraints
   where table_name in( 'Z_PARENT','Z_CHILD');
```

```
TABLE_NAME                     CONSTRAINT_NAME                C STATUS
------------------------------ ------------------------------ - --------
Z_CHILD                        CHILD_PK                       P ENABLED
Z_CHILD                        CHILD_FK                       R ENABLED
Z_PARENT                       PARENT_PK                      P ENABLED
```

Now drop the parent with cascade

```
   drop table z_parent cascade constraints;
```

Display the constraints again

```
   select table_name, constraint_name, constraint_type, status
   from user_constraints
   where table_name in( 'Z_PARENT','Z_CHILD');
```

```
TABLE_NAME                     CONSTRAINT_NAME                C STATUS
------------------------------ ------------------------------ - --------
Z_CHILD                        CHILD_PK                       P ENABLED
```

We lost the parent_pk table- since we dropped that table. But we also lost the fk constraint on the child table. The child table remains and keeps its primary key- but it loses its foreign key.

5. Data types you can use in tables

For our tables we have been using only a few types of data to define our columns. Oracle supports more data types and you can find information about all of these in the appendix of the Price book. The following is a description of some of the most commonly used types. I am not going to give you a lot of ranges for the types- you can look those up if you need them.

5.1. Character strings

CHAR is used for fixed length strings. We use CHAR(2) to store state abbreviations since they all have a 2 letter. If you are storing data where all of the data has the same number of characters- such as SSN, ISBN13, some product codes, then CHAR is appropriate. If necessary the data will be end-padded with blanks to the stated length.

VARCHAR2 is used for variable length strings. We could use VARCHAR2(25) to store customer last names assuming we won't wont to store a name longer than 25 characters.

For both of these - if you are defining a table column with char or varchar and try to insert a value longer than the defined length, you will get an error. Char defaults to a length of 1 if you do not state a length' varchar2 requires a length.

5.2. Integers

We usually use INTEGER or INT or we can use NUMBER(9) specifying a width.

5.3. Fixed precision

Number this lets you set up a type such as number(6,2) which can hold numbers from -9999.99 to +9999.99. The first value is the number of digits and the second the number of digits after the decimal point

For integer you can use a definition such as Number(5) specifying a width or Integer. Number(5) limits the values entered to 5 digits (-99999 to +99999)

5.4. Floating point/ Approximate numbers

Float and real- Use these types for numbers where the number of digits after the decimal is not fixed.

Suppose you wanted a table of different types of animals and their approximate weight. An elephant weights about 6800 kg and an ant about 0.000003 kg. It might make more sense to use a float than a decimal.

```
create table weights (an_type varchar2(15), an_weight_kg  float);
insert into weights values ( 'ant', 0.000003);
insert into weights values ( 'elephant', 6800);
select * from weights;
```

```
AN_TYPE         AN_WEIGHT_KG
--------------- ----------------------
ant             0.000003
elephant        6800
```

5.5. Temporal data

These are for storing date and time values and it helps to see the range of possible values for these

DATE Jan 1 4712 B.C. and December 31 9999 A.D.

The Date type stores both a date and a time component.

5.1. XML

We will get to XML data later

6. Changing the Table Design

You can add and drop columns and constraints, modify columns data types and constraints, and Set Unused columns. If the table contains data, some of these changes are restricted. For example, you cannot add a check constraint to a table if the existing data would violate that constraint.

Demo 18: An alter query to add another column to an existing table

```
Create table ddl_alter (id integer primary key);

desc ddl_alter;
```

```
Name                           Null     Type
------------------------------ -------- ------------------------------
ID                             NOT NULL NUMBER
```

```
  alter   table  ddl_alter
  add     d_office varchar2(10) constraint office_un unique;
  desc ddl_alter;
```

```
Name                           Null     Type
------------------------------ -------- ------------------------------
ID                             NOT NULL NUMBER
D_OFFICE                                VARCHAR2(10)
```

Demo 19: Adding more than one column. Delimit the column collection with parentheses.

```
  alter   table  ddl_alter
  add    ( e_ssn char(11)
         , e_namefirst varchar2(20)
         , e_salary number(6)
           constraint e_salary_ck check (e_salary between 20000 and 100000)
        );
```

```
Name                           Null     Type
------------------------------ -------- ------------------------------
ID                             NOT NULL NUMBER
D_OFFICE                                VARCHAR2(10)
E_SSN                                   CHAR(11)
E_NAMEFIRST                             VARCHAR2(20)
E_SALARY                                NUMBER(6)
```

Demo 20: You can drop a column.

```
  alter table ddl_alter
  drop column e_ssn;
```

```
Name                           Null     Type
------------------------------ -------- ------------------------------
ID                             NOT NULL NUMBER
D_OFFICE                                VARCHAR2(10)
E_NAMEFIRST                             VARCHAR2(20)
E_SALARY                                NUMBER(6)
```

Named constraints can be dropped by using an Alter table statement. You can use the system defined constraint names. You can also drop some constraints by naming the constraint type- for example:

Demo 21: Dropping a primary key. This is not ambiguous since a table has only one primary key.

```
  alter table ddl_alter
  drop primary key;
```

Note that the attribute id was not dropped - the PK constraint was dropped - which then removed the not null constraint.

```
Name                           Null     Type
------------------------------ -------- ------------------------------
ID                                      NUMBER
D_OFFICE                                VARCHAR2(10)
E_NAMEFIRST                             VARCHAR2(20)
E_SALARY                                NUMBER(6)
```

In some cases you may need to use Cascade; for example, when you are dropping a pk from a table which is referenced by a fk in a child table: `ALTER TABLE tblDemo DROP PRIMARY KEY CASCADE;`

Demo 22: You can drop a constraint without dropping the column; this does not drop any data.

```
alter   table  ddl_alter
drop    constraint office_un;
```

Demo 23: Increasing the width of a column

```
alter   table ddl_alter
modify  e_namefirst varchar2(25);
```

Demo 24: Changing a constraint by dropping it and recreating it. This should be done at a time when other people are not using the tables so that no rows are inserted between these two statements..

```
alter   table  ddl_alter
drop    constraint e_salary_ck;

alter   table  ddl_alter
add     constraint e_salary_ck check (e_salary between 2000 and 10000);
```

Demo 25: Show the table description

```
desc ddl_alter;
```

```
Name                    Null?    Type
----------------------- -------- ----------------
ID                               NUMBER(38)
D_OFFICE                         VARCHAR2(10)
E_NAMEFIRST                      VARCHAR2(25)
E_SALARY                         NUMBER(6)
```

Demo 26: Add a few rows of data to the table ddl_alter

```
Insert into ddl_alter values (1, 'sales', 'Joe', 5000);
Insert into ddl_alter values (2, 'sales', 'Jill', 9000);

select  *
from    ddl_alter;
```

```
        ID D_OFFICE   E_NAMEFIRST                 E_SALARY
---------- ---------- ------------------------- ----------
         1 sales      Joe                             5000
         2 sales      Jill                            9000
```

Demo 27: Set unused columns. If you mark a column as unused, it is no longer accessible

```
alter   table ddl_alter
set     unused column e_salary;
```

```
Table altered.
```

Demo 28: Select * does not see the unused column.

```
select  *
from    ddl_alter;
```

```
        ID D_OFFICE   E_NAMEFIRST
---------- ---------- -------------------------
         1 sales      Joe
         2 sales      Jill
```

Demo 29: describe does not see the unused column.

```
desc ddl_alter;
```

```
Name                    Null?    Type
```

```
 ---------------------- -------- ----------------
 ID                               NUMBER(38)
 D_OFFICE                         VARCHAR2(10)
 E_NAMEFIRST                      VARCHAR2(25)
```

Demo 30: You can later do the actual drop of the columns

```
alter   table ddl_alter
drop    unused columns;
```

One reason to do the actual column drop later is that this may involve a lot of work being done by the dbms to recreate the physical files; there may be a time when the system is not as impacted and the drop column can be run at that time.

Once you have marked a column as unused you cannot change your mind and undo that setting