## Table of Contents



# 1. Data Manipulation Language queries

After we have created tables, we need to be able to add rows of data to the tables; we also need to be able to change the rows in a table. For this, think of a table as a collection of rows- these changes affect one or more rows in the table. With an insert or a delete statement we appear to be changing on a row-basis but that does affect the entire table. With an update statement, we are actually replacing the current version of the rows with a new version of the rows so that is also changing the table.

DML (Data Manipulation Language) queries are used to change the data in the tables.

- Remove all rows from a table: Truncate
- Remove specified rows from a table: Delete
- Change existing rows in a table: Update
- Append single rows to a table: Insert Into
- Append multiple rows into a table, getting the rows from another table: Insert Into… Select
- Create new tables that have the same design as an existing table: Create Table As
- Do updates, insert, deletes in a single statement: Merge

We will do most of these queries using the following two tables AC_Dept and AC_Emp created below.

Since these statements change the data in tables, the changed data must meet the constraints that were created for the tables. If it does not, then the SQL statement will fail. If you attempt to execute an SQL statement that changes several rows of data and any one of the new rows would be invalid, then the entire statement fails and none of the rows are changed.

Demo 01: Drop the tables if they exist. Note that if you have created a series of related tables, you may have to drop child tables before parent tables.

```
drop table ac_emp;
drop table ac_dept;
```

Demo 02: Create table statements. Read these carefully to see the various constraints on the tables.

```
create table  ac_dept  (
   d_id               number(3)     constraint ac_dept_pk    primary key
 , d_name             varchar2(15)  not null
 , d_budget           number(6)     null
 , d_expenses_to_date number(6)     null
 , d_city             varchar2(15)  not null
 , d_state            char(2)       not null
 , constraint  ac_dlocation_un unique(d_name, d_city, d_state)
);

create table ac_emp  (
   e_id     number(3)
```

```
 , e_name   varchar2(15) not null
 , d_id     number(3) null
 , salary   number(5)
            default 30000
            constraint ac_esalary_ck1 check(salary between 20000 and 90000)
            null
 , hiredate date
            default trunc(sysdate)
            constraint ac_ehired_ck check(hiredate> to_date( '2000-01-09',
'yyyy-mm-dd'))
            constraint ac_ehired_Midnight check(Trunc(hiredate) = hiredate)
            null
 , e_status char (4)
            constraint ac_estatus_ck
            check(e_status in ('PERM', 'TEMP'))
            null
 , constraint ac_emp_pk  primary key (e_id )
 , constraint ac_emp_dept_fk foreign key(d_id)  references ac_dept
);
```

Demo 03: Initial inserts- These just set up some rows as I have done in the scripts

```
Insert Into ac_dept (d_id, d_name, d_budget, d_expenses_to_date, d_city, d_state)
  Values (301,'FINANCE', 50000, 15000, 'PEKIN', 'IL');

Insert Into ac_dept (d_id, d_name, d_budget, d_expenses_to_date, d_city, d_state)
  Values (501, 'SALES',15000, 16000,  'RENO', 'NV');

Insert Into ac_dept (d_id, d_name, d_budget, d_expenses_to_date, d_city, d_state)
 Values (201, 'SALES', 35000, Null, 'CHICAGO', 'IL');

Insert Into ac_dept (d_id, d_name, d_budget, d_expenses_to_date, d_city, d_state)
    Values (401,'ADMIN', null, null,'CHICAGO', 'IL');
```

```
  D_ID D_NAME       D_BUDGET D_EXPENSES_TO_DATE D_CITY     D_STATE
------ ---------- ---------- ------------------ ---------- ----------
   201 SALES           35000                    CHICAGO    IL
   301 FINANCE         50000              15000 PEKIN      IL
   401 ADMIN                                    CHICAGO    IL
   501 SALES           15000              16000 RENO       NV
```

```
Insert Into ac_emp (e_id, e_name, d_id, salary, hiredate, e_status)
  Values (30, 'HANSON', 201, 40000, '15-MAY-2013', 'PERM');
Insert Into ac_emp (e_id, e_name, d_id, salary, hiredate, e_status)
  Values (40, 'IBSEN', 201, 45000, '20-MAY-2013', 'PERM');
Insert Into ac_emp (e_id, e_name, d_id, salary, hiredate, e_status)
  Values (50, 'MILES', 401, 25000, '20-JUNE-2013', 'PERM');
```

```
  E_ID E_NAME       D_ID     SALARY HIREDATE  E_STATUS
------ ---------- ------ ---------- --------- ----------
    30 HANSON        201      40000 15-MAY-13 PERM
    40 IBSEN         201      45000 20-MAY-13 PERM
    50 MILES         401      25000 20-JUN-13 PERM
```

# 2. Truncating a table

```
truncate table mytable;
```

This removes all of the rows but does not remove the table structure. This is faster than using Delete but it cannot be rolled back. It does not activate triggers ( code that can run when you change data in a table). If there are any fk references to this table, those constraints have to be disabled before truncating the table.

Demo 04: Truncate

You can do the following and the table ac_emp is emptied.

```
Truncate table ac_emp;
```

```
Table truncated.
```

```
select * from ac_emp;
```

```
no rows selected
```

But if you try this with the table ac_dept, you get an error and the table is not truncated. Note that the table ac_emp is empty so this is not a Cascade issue; this is based on the existence of a FK constraint.

```
Truncate Table ac_dept;
```

```
Truncate Table ac_dept
               *
ERROR at line 1:
ORA-02266: unique/primary keys in table referenced by enabled foreign keys
```

```
select * from ac_dept;
```

```
 D_ID D_NAME       D_BUDGET D_EXPENSES_TO_DATE D_CITY     D_STATE
------ ---------- ---------- ------------------ ---------- ----------
   301 FINANCE         50000              15000 PEKIN      IL
   501 SALES           15000              16000 RENO       NV
   201 SALES           35000                    CHICAGO    IL
   401 ADMIN                                    CHICAGO    IL
```

If you deleted the rows in ac_emp, use the insert statements to put those rows back into the table.

# 3. Delete queries

**Remove Existing Rows**

**If you do these deletes, put the rows back into the table as each step. You can do this with the inserts in the earlier demo.**

/* Model to delete row.

```
delete
from    tablename
where   condition;
```

Demo 05: This deletes all rows that match the test. ( then reinsert those rows)

```
delete
from ac_emp
where d_id = 201;
```

```
2 rows deleted
```

Demo 06: This uses a subquery to allow a test for the proper rows to be deleted.

```
delete
from ac_emp
where d_id in (
          select d_id
          from ac_dept
          where d_state = 'NV');
```

```
0 rows deleted
```

That department was empty.

Change the query to delete people from the Chicago departments. Then reinsert those rows.

```
delete
from ac_emp
where d_id in (
         select d_id
         from ac_dept
         where d_city = 'CHICAGO');
```

```
3 rows deleted
```

If you do not include a Where clause you will delete all of the rows in the table.

A delete statement deletes rows from one table only.

If you are deleting parent rows and do not have cascade delete set for the relationship, then parent rows with existing child rows cannot be deleted. ORA-02292: integrity constraint (*ConstraintName*) violated - child row found.

Suppose we want to delete the two rows with the highest values for salary. We can use rownum but rownum and sort need to be handled separately. The following query will do a descending sort by salary.

```
select e_id
from ac_emp
order by salary desc;
```

If I use that subquery as a table expression, then I can use rownum in the outer query

```
select e_id
from
(  select e_id
   from ac_emp
   order by salary desc
)
where rownum <=2;
```

Now I could use that query as the subquery for a delete

```
delete from ac_emp
where e_id in (
   select e_id
   from
   (  select e_id
      from ac_emp
      order by salary desc
   )
   where rownum <=2
);
```

This will delete two rows. If there were a tie on the salary value at the second row then one of those tied rows would be deleted but we are not controlling which one. That does not sound like a good business decision.

# 4. Update queries

**Change Existing Rows**

/* Model for an update.

```
update   tablename
set      columnname  =  value or expression
where    condition;
```

Example: We have a column in the employee table that stores an Employee Status attribute. All employees will be changed to "TEMP" status and then some employees will then be changed to "PERM" status.

Demo 07: A few more employees

```
Insert Into ac_emp (e_id, e_name, d_id, salary, hiredate, e_status)
  Values (10, 'FREUD',  301, 30000, '06-Jun-2002', 'PERM');
Insert Into ac_emp (e_id, e_name, d_id, salary, hiredate, e_status)
  Values (20, 'MATSON', 201, 60000, '06-Jun-2010', 'PERM');
Insert Into ac_emp (e_id, e_name, d_id, salary, hiredate, e_status)
  Values (60, 'TANG',   401, 25000, '15-JULY-2012', 'TEMP');
Insert Into ac_emp (e_id, e_name, d_id, salary, hiredate, e_status)
  Values (70, 'KREMER', 501, 50000, '15-JULY-2012', 'TEMP');
Insert Into ac_emp (e_id, e_name, d_id, salary, hiredate, e_status)
  Values (80, 'PAERT',  201, 65000, '15-JULY-2012', 'TEMP');
Insert Into ac_emp (e_id, e_name, d_id, salary, hiredate, e_status)
  Values (90, 'JARRET', 301, 60000, '08-AUGUST-2015', 'TEMP');
```

```
 E_ID E_NAME       D_ID     SALARY HIREDATE  E_STATUS
------ ---------- ------ ---------- --------- ----------
    10 FREUD         301      30000 06-JUN-02 PERM
    20 MATSON        201      60000 06-JUN-10 PERM
    30 HANSON        201      40000 15-MAY-13 PERM
    40 IBSEN         201      45000 20-MAY-13 PERM
    50 MILES         401      25000 20-JUN-13 PERM
    60 TANG          401      25000 15-JUL-12 TEMP
    70 KREMER        501      50000 15-JUL-12 TEMP
    80 PAERT         201      65000 15-JUL-12 TEMP
    90 JARRET        301      60000 08-AUG-15 TEMP
```

Demo 08: SET the value for E_Status to TEMP for all rows. Everyone now has TEMP status.

```
update ac_emp
set e_status = 'TEMP';
```

Demo 09: The Where clause allows you to update specified rows. This changes some of the data to PERM.

```
update ac_emp
set e_status  = 'PERM'
where hiredate < to_date('01-JUN-2013');
```

Demo 10: You can update multiple attributes. This updates one row since the Where clause selects for a PK. There is only one SET clause even if we update multiple columns

```
update ac_dept
set d_city  = 'SAN FRANCISCO',
    d_state = 'CA'
where d_id = 401;
```

Demo 11: The SET clause can use an expression.

```
update  ac_emp
set     salary = salary * 1.1
where   e_status = 'PERM';
```

```
 E_ID E_NAME       D_ID     SALARY HIREDATE  E_STATUS
------ ---------- ------ ---------- --------- ----------
    10 FREUD         301      33000 06-JUN-02 PERM
    20 MATSON        201      66000 06-JUN-10 PERM
    30 HANSON        201      44000 15-MAY-13 PERM
    40 IBSEN         201      49500 20-MAY-13 PERM
    50 MILES         401      25000 20-JUN-13 TEMP
    60 TANG          401      27500 15-JUL-12 PERM
    70 KREMER        501      55000 15-JUL-12 PERM
    80 PAERT         201      71500 15-JUL-12 PERM
    90 JARRET        301      60000 08-AUG-15 TEMP
9 rows selected.
```

Demo 12: Updating rows in a table using a case function.

```
update ac_emp
set salary =  salary * ( 1 + case
                    when salary < 40000 then 0.25
                    when salary < 50000 then 0.10
                    else  0.05  end  ) ;
```

```
 E_ID E_NAME       D_ID     SALARY HIREDATE  E_STATUS
------ ---------- ------ ---------- --------- ----------
    10 FREUD         301      41250 06-JUN-02 PERM
    20 MATSON        201      69300 06-JUN-10 PERM
    30 HANSON        201      48400 15-MAY-13 PERM
    40 IBSEN         201      54450 20-MAY-13 PERM
    50 MILES         401      31250 20-JUN-13 TEMP
    60 TANG          401      34375 15-JUL-12 PERM
    70 KREMER        501      57750 15-JUL-12 PERM
    80 PAERT         201      75075 15-JUL-12 PERM
    90 JARRET        301      63000 08-AUG-15 TEMP
```

Demo 13: This uses a subquery to allow a test for the proper rows to update. The subquery uses one table to provide the filter to update a different table.

```
update ac_emp
set salary = salary * 1.1
where d_id in
         (select d_id
          from ac_dept
          where d_state = 'CA');
```

Demo 14: Another use of a subquery in an update. This uses an inline view and the inline view filters for the rows to be updated.

```
update (
      select salary
      from ac_emp
      join ac_dept using (d_id)
      where d_state = 'CA'
      )
 set  salary = salary * 1.1;
```

Demo 15: You can use variables to do the update. You want to start thinking about variables as ways to get data from an application layer program that delivers values to a query.

```
select d_city, d_state, e_name, salary
from ac_dept
join ac_emp on ac_dept.d_id = ac_emp.d_id
order by d_state, d_city ;
```

```
D_CITY        D_STATE    E_NAME         SALARY
------------- ---------- ---------- ----------
SAN FRANCISCO CA         TANG            41594
SAN FRANCISCO CA         MILES           37813
CHICAGO       IL         IBSEN           54450
CHICAGO       IL         HANSON          48400
CHICAGO       IL         MATSON          69300
CHICAGO       IL         PAERT           75075
PEKIN         IL         FREUD           41250
PEKIN         IL         JARRET          63000
RENO          NV         KREMER          57750
```

```
variable city varchar2(15);
variable state char(2);
```

```
exec :city := 'CHICAGO';
exec :state := 'IL';
variable wageDifferential number
exec :wageDifferential := 1.05


update ac_emp
set salary = salary * :wageDifferential
where d_id in
         (select d_id
          from ac_dept
          where d_state = :state and d_city = :city);

select d_city, d_state, e_name, salary
from ac_dept
join ac_emp on ac_dept.d_id = ac_emp.d_id
order by d_state, d_city ;
```

```
D_CITY        D_STATE    E_NAME         SALARY
------------- ---------- ---------- ----------
SAN FRANCISCO CA         TANG            41594
SAN FRANCISCO CA         MILES           37813
CHICAGO       IL         IBSEN           57173
CHICAGO       IL         HANSON          50820
CHICAGO       IL         MATSON          72765
CHICAGO       IL         PAERT           78829
PEKIN         IL         FREUD           41250
PEKIN         IL         JARRET          63000
RENO          NV         KREMER          57750
```

Demo 16: This updates data in a row using other data in the same row. It subtracts the expenses amount from the budget and then sets the expenses to null. Of course this creates problems for the business since we end up doing arithmetic with nulls and we let one department go over budget.

```
update ac_dept
set
   d_budget = d_budget - d_expenses_to_date
 , d_expenses_to_date = null;

 select * from ac_dept;
```

```
  D_ID D_NAME       D_BUDGET D_EXPENSES_TO_DATE D_CITY        D_STATE
------ ---------- ---------- ------------------ ------------- ----------
   301 FINANCE         35000                    PEKIN         IL
   501 SALES           -1000                    RENO          NV
   201 SALES                                    CHICAGO       IL
   401 ADMIN                                    SAN FRANCISCO CA
```

Return to the original rows for the ac_dept table

Demo 17: We can use a more complex expression for the new values

```
Update ac_dept
Set d_budget = Case
               When d_budget > COALESCE(d_expenses_to_date, 0)
               Then d_budget - COALESCE(d_expenses_to_date, 0)
               Else 0
               End
  , d_expenses_to_date = Case
                         When d_budget > COALESCE(d_expenses_to_date, 0)
                         Then 0
```

```
                            Else COALESCE(d_expenses_to_date, 0) - d_budget
                            End
  Where d_budget Is Not Null;
```

| D_ID | D_NAME | D_BUDGET | D_EXPENSES_TO_DATE | D_CITY | D_STATE |
|---|---|---|---|---|---|
| 301 | FINANCE | 35000 | 0 | PEKIN | IL |
| 501 | SALES | 0 | 1000 | RENO | NV |
| 201 | SALES | 35000 | 0 | CHICAGO | IL |
| 401 | ADMIN | | | CHICAGO | IL |

Demo 18: Or we could use a simple expression and just increase the budgets by a random value , letting people with no budget have something.

```
  update ac_dept
  set d_budget = round(dbms_random.value,2)* coalesce(d_budget, 500);
```

| D_ID | D_NAME | D_BUDGET | D_EXPENSES_TO_DATE | D_CITY | D_STATE |
|---|---|---|---|---|---|
| 301 | FINANCE | 33950 | 0 | PEKIN | IL |
| 501 | SALES | 0 | 1000 | RENO | NV |
| 201 | SALES | 21000 | 0 | CHICAGO | IL |
| 401 | ADMIN | 350 | | CHICAGO | IL |

Demo 19: Or just give up on budgets

```
  update ac_dept
  set d_budget = null;
```

**Discussion**

The keyword Update is used to identify the table to be changed. The keyword Set is used to identify the column(s) to be changed and the new value(s) to be used. The new value supplied overwrites the original value. You can change more than one column's value in the row.

Updates are done 'all at once'. If any of the potential changes do not meet the table's description or constraints, then none of the changes are made.

You can use many of the techniques we have for Select statements to get the proper rows to be updated.

As a reminder that SQL is a set-oriented language; Oracle can swap columns with a single query.

```
  select  * from  demo;
```

| COL_1 | COL_2 |
|---|---|
| 1 | 101 |
| 2 | 202 |
| 3 | 999 |

```
  update   demo
  set col_1 = col_2,
      col_2 = col_1
```

```
3 rows updated.
```

```
  select  * from  demo;
```

| COL_1 | COL_2 |
|---|---|
| 101 | 1 |
| 202 | 2 |
| 999 | 3 |

# 5. Insert into queries

## 5.1. Add a single new row using a value list

Model to insert a single row into the table specifying the column names.

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

The preferred technique for a single row insert is to list the column names in parentheses after the table name, followed by the key word Values and then the list of values/expressions in parentheses. The order in which you list the column and the order of the values must be consistent. Since I am listing the columns in the column_list,, the order of the columns in the insert statement does not have to match the order of the columns in the table definition. I do not need to supply a value for a nullable column.

Demo 20: Inserts a single row into the table specifying the column names.

```
insert into ac_dept (d_name, d_id, d_city,d_state, d_budget,d_expenses_to_date)
values  ('FINANCE', 258, 'MORTON', 'IL',     50000,     15000);

insert into ac_dept (d_id,d_budget,d_expenses_to_date,d_name,d_city,d_state)
values  (745,  15000, 16000,  'SALES', 'ZENO',  'NV');
```

An alternative syntax for the insert skips the column list; in that case the order of the values must match the order of the columns in the table definition and no columns can be skipped.

Demo 21: Insert a single row into the table; the data values must be in the proper order, matching the order of the columns in the table definition. If you do not want to inset a value for a nullable column, use the word null.

```
insert into ac_dept
values ( 465, 'SALES', 35000, null, 'PEKIN', 'IL');
```

Demo 22: Listing the column names allows you to skip column names if there are no values for that column.

```
insert into ac_dept (d_name,  d_id, d_city, d_state )
values  ('ADMIN', 654, 'MORTON', 'IL') ;
```

Demo 23: When the table has default values defined, **you can use the word DEFAULT instead of a value**, or use the column list that omits that column.

```
insert into ac_emp (e_id, e_name, d_id, salary, hiredate, e_status)
            values ( 101, 'FREUD', 301, DEFAULT, DEFAULT, 'PERM');
```

Demo 24: When the table has default values defined, you can use the word DEFAULT instead of a value**, or use a column list that omits that column.**

```
insert into ac_emp (e_id, e_name, d_id,  e_status)
            values ( 102, 'MATSON', 201, 'PERM');


select * from ac_emp where e_id in (101, 102);
```

| E_ID | E_NAME | D_ID | SALARY | HIREDATE | E_STATUS |
|---|---|---|---|---|---|
| 101 | FREUD | 301 | 30000 | 08-NOV-15 | PERM |
| 102 | MATSON | 201 | 30000 | 08-NOV-15 | PERM |

**Discussion**

The keywords Insert Into are followed by the name of the table. You can list the columns of the table in parentheses after the name of the table. Then you use the keyword Values to indicate that the lists of values will follow. Include the data values inside parentheses after the keyword Values.

If you are going to insert a value into each column of the row and you are going to list the data values in the same order as the column order in the table, then you can use the second version shown and skip the column name list. This is not as safe as the first version since the table structure can be changed to add additional columns. From a logical point of view, the order of the columns in a table cannot be logically significant and this shorthand version relies on the column order.

If you are inserting a row that allows a null in a specific column, you use the word null in the value list to insert the null.

The data value you are trying to insert must fit the column definition. Any constraints you have placed on the table must be met.

Oracle will not truncate a string to fit into a VARCHAR attribute if it is longer than the defined attribute. Character literals must be enclosed in quotes. The case of the data values is maintained in the table.

A numeric value may not be larger in magnitude than the defined column. If there are more digits after the decimal than was defined, Oracle will round the number to fit. So if the column was defined as NUMBER (5,2) you can insert a value 123.4567 but you cannot insert 1234.567

A date value is traditionally written as a string literal in the default format used by the DBMS – such as '06-AUG-2007' . Or you can use the To_Date function with a format string. This is safer in case the default string format is changed. You can also use the ansi standard date format- such as date '2005-05-19'

You can use SYSDATE as a value to insert the current date-and-time into a column.

These rules also apply to updates- you cannot update a row in a table in a way that violates the table constraints.

Oracle does not support a multi-row insert you may know from other dbms,

Demo 25: Our tables at this time. ac_dept, ac_emp

```
  D_ID D_NAME       D_BUDGET D_EXPENSES_TO_DATE D_CITY        D_STATE
------ ---------- ---------- ------------------ ------------- ----------
   201 SALES           21000                  0 CHICAGO       IL
   258 FINANCE         50000              15000 MORTON        IL
   301 FINANCE         33950                  0 PEKIN         IL
   401 ADMIN             350                    CHICAGO       IL
   465 SALES           35000                    PEKIN         IL
   501 SALES               0               1000 RENO          NV
   654 ADMIN                                    MORTON        IL
   745 SALES           15000              16000 ZENO          NV
8 rows selected.
```

```
  E_ID E_NAME       D_ID     SALARY HIREDATE  E_STATUS
------ ---------- ------ ---------- --------- ----------
    10 FREUD         301      30000 06-JUN-02 PERM
    20 MATSON        201      60000 06-JUN-10 PERM
    30 HANSON        201      40000 15-MAY-13 PERM
    40 IBSEN         201      45000 20-MAY-13 PERM
    50 MILES         401      25000 20-JUN-13 PERM
    60 TANG          401      25000 15-JUL-12 TEMP
    70 KREMER        501      50000 15-JUL-12 TEMP
    80 PAERT         201      65000 15-JUL-12 TEMP
    90 JARRET        301      60000 08-AUG-15 TEMP
   101 FREUD         301      30000 08-NOV-15 PERM
   102 MATSON        201      30000 08-NOV-15 PERM
11 rows selected.
```

# 6. Append rows to an existing table

```
/* Model to insert row from one table into another.
   insert into my_table (col1_name, col2_name,  . . )
      select value1_expr, value2_expr , . .
      from . . . ;
```

Assumptions: We have a table of potential hires (AC_PotentialHire). We have added rows to this table with nulls for the Hiredate. If we decide to actually hire someone, we put a value into the HireDate column. We want to add anyone from that table with an actual hire date into our employee table and then remove them from the potential hire table.

Demo 26:

```
Create table ac_potentialHire (  e_id number(3)
, e_name varchar2(15) not null
, d_id number(3)
, salary number(5)
, hiredate date
, e_status char (4)
);

insert into ac_potentialHire (e_id, e_name, d_id, salary, hiredate, e_status)
             values( 201, 'ROBYN', 301, 20000, '15-NOV-2015', 'TEMP');
insert into ac_potentialHire (e_id, e_name, d_id, salary, hiredate, e_status)
             values( 202, 'ECHART', 201, 20000, null, 'TEMP');
insert into ac_potentialHire (e_id, e_name, d_id, salary, hiredate, e_status)
             values( 203, 'TATUM', 301, 25000, '18-NOV-2015', 'TEMP');
```

The table AC_PotentialHire

| E_ID | E_NAME | D_ID | SALARY | HIREDATE | E_ST |
|---|---|---|---|---|---|
| 201 | ROBYN | 301 | 20000 | 15-NOV-15 | TEMP |
| 202 | ECHART | 201 | 20000 | | TEMP |
| 203 | TATUM | 301 | 25000 | 18-NOV-15 | TEMP |

The next two demos should be considered as a single transaction.

Demo 27:  Append rows from sc_potentialhires into ac_emp. This inserts 2 rows

```
insert into  ac_emp (e_id, e_name, d_id, salary, hiredate, e_status)
    select e_id, e_name, d_id, salary, hiredate, e_status
    from ac_potentialhire
    where hiredate is not null;
```

This uses the keywords Insert Into . . .  followed by a subquery that returns the data for the rows.

With this syntax for insert,  you do not use the keyword values- you simply include the subquery.

Demo 28:  Delete these rows from the potential hire table.

```
delete
from ac_potentialhire
where hiredate is not null;

Select * from ac_emp;
```

```
  E_ID E_NAME       D_ID     SALARY HIREDATE  E_STATUS
------ ---------- ------ ---------- --------- ----------
    30 HANSON        201      40000 15-MAY-13 PERM
    40 IBSEN         201      45000 20-MAY-13 PERM
    50 MILES         401      25000 20-JUN-13 PERM
    10 FREUD         301      30000 06-JUN-02 PERM
    20 MATSON        201      60000 06-JUN-10 PERM
    60 TANG          401      25000 15-JUL-12 TEMP
    70 KREMER        501      50000 15-JUL-12 TEMP
    80 PAERT         201      65000 15-JUL-12 TEMP
    90 JARRET        301      60000 08-AUG-15 TEMP
   101 FREUD         301      30000 08-NOV-15 PERM
   102 MATSON        201      30000 08-NOV-15 PERM
   201 ROBYN         301      20000 15-NOV-15 TEMP
   203 TATUM         301      25000 18-NOV-15 TEMP
```

```
Select * from ac potentialhire;
```

```
     E_ID E_NAME               D_ID     SALARY HIREDATE  E_ST
---------- --------------- ---------- ---------- --------- ----
       202 ECHART                 201      20000           TEMP
```

The use of the column list is optional if the subquery select clause include all of the columns in the correct order. You can also use Select * in the subquery if the select columns are in the correct order. The columns in the subquery can include calculated columns. The subquery could include a join clause.

Note that a query is done "all at once". If any of the potential new rows do not meet the table's description or constraints, then none of the rows are added.

# 7. Create table As queries

This creates a new table and inserts rows into the table.

**Create A New Table From An Existing Table**

```
CREATE TABLE NewTableName  AS
    SELECT   ColumnExpresssions
      FROM   ExistingTableName
     WHERE   Condition;
```

Demo 29: Creates a new table with rows from an existing table.

```
create table  ac_emp_d201 as
    select *
    from   ac_emp
    where  d_id = 201;

select  * from  ac_emp_d201;
```

```
  E_ID E_NAME       D_ID     SALARY HIREDATE  E_STATUS
------ ---------- ------ ---------- --------- ----------
    30 HANSON        201      40000 15-MAY-13 PERM
    40 IBSEN         201      45000 20-MAY-13 PERM
    20 MATSON        201      60000 06-JUN-10 PERM
    80 PAERT         201      65000 15-JUL-12 TEMP
   102 MATSON        201      30000 08-NOV-15 PERM
```

Demo 30: Creates a new table with specified attributes from the rows in existing tables.

```
create table   ac_emp_IL  as
    select  e_name, salary, d_city, d_state
    from ac_dept
    join ac_emp using (d_id)
    where  d_state = 'IL' ;
```

```
E_NAME         SALARY D_CITY        D_STATE
---------- ---------- ------------ ----------
HANSON          40000 CHICAGO      IL
IBSEN           45000 CHICAGO      IL
MILES           25000 CHICAGO      IL
FREUD           30000 PEKIN        IL
MATSON          60000 CHICAGO      IL
TANG            25000 CHICAGO      IL
PAERT           65000 CHICAGO      IL
JARRET          60000 PEKIN        IL
FREUD           30000 PEKIN        IL
MATSON          30000 CHICAGO      IL
ROBYN           20000 PEKIN        IL
TATUM           25000 PEKIN        IL
```

Demo 31: Suppose you did not want all the columns and you wanted to use different column names. You could modify the select to include only the columns you want and the expressions you want and use the column aliases to set up the new names.

```
create table  ac_emp_2 as
    select e_id as EMPID
    , initcap(e_name) as EMPNAME
    , extract( year from hiredate) as EMPHIREYEAR
    from ac_emp
   where d_id = 201;

   select * from ac_emp_2;
```

```
     EMPID EMPNAME         EMPHIREYEAR
---------- -------------- -----------
        30 Hanson                2013
        40 Ibsen                 2013
        20 Matson                2010
        80 Paert                 2012
       102 Matson                2015
```

```
Desc ac_emp_2;
```

```
Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------
 EMPID                                              NUMBER(3)
 EMPNAME                                            VARCHAR2(15)
 EMPHIREYEAR                                        NUMBER
```

Demo 32: Creates an empty table that has the same columns as an existing table. The Where clause is just to have a False criteria so that no rows are returned into the new table.

```
create table ac_potentialhire_2 as
    select   *
    from     ac_emp
    where    1=2;
```

Using this technique to create tables does not copy over check constraints, primary or foreign keys, unique keys, column default values. You can use the Alter Table commands to create the needed constraints. It does keep the data types.

If you do desc of ac_emp and of ac_potentialhire_2, they will appear mostly the same.

```
desc ac_emp
```

```
Name                                      Null?    Type
----------------------------------------- -------- -----------------
E_ID                                      NOT NULL NUMBER(3)
E_NAME                                    NOT NULL VARCHAR2(15)
D_ID                                               NUMBER(3)
```

```
SALARY                                                NUMBER(5)
HIREDATE                                              DATE
E_STATUS                                              CHAR(4)
```

```
desc ac_potentialhire_2;
```

```
Name                                      Null?    Type
----------------------------------------- -------- -----------------
E_ID                                               NUMBER(3)
E_NAME                                    NOT NULL VARCHAR2(15)
D_ID                                               NUMBER(3)
SALARY                                             NUMBER(5)
HIREDATE                                           DATE
E_STATUS                                           CHAR(4)
```

But you can insert rows into ac_potentialhire_2 that do not follow the ac_emp constraints. For example:

```
Insert Into ac_potentialhire_2 (e_id, e_name, d_id, salary, hiredate, e_status)
  Values (10, 'FREUD', 301, Default, '06-Jun-2002', 'PERM');

Insert Into ac_potentialhire_2 (e_id, e_name, d_id, salary, hiredate, e_status)
  Values (10, 'FREUD', 301, Default, '06-Jun-2002', 'PERM');

Insert Into ac_potentialhire_2 (e_id, e_name, d_id, salary, hiredate, e_status)
  Values (21, 'FREUD', 7, 35.12, '06-Jun-1847', 'unkn');

Insert Into ac_potentialhire_2 (e_id, e_name, d_id, salary, hiredate, e_status)
  Values (10, 'FREUD', 301, Default, '06-Jun-2002', 'PERM');
```

We have duplicate values for e_id, the salary default from ac_emp was not followed; someone was hired in 1847 with a status of unkn and a salary of 35.

```
   E_ID E_NAME       D_ID     SALARY HIREDATE  E_STATUS
------ ---------- ------ ---------- --------- ----------
    10 FREUD         301            06-JUN-02 PERM
    10 FREUD         301            06-JUN-02 PERM
    21 FREUD           7         35 06-JUN-47 unkn
    10 FREUD         301            06-JUN-02 PERM

4 rows selected.
```