Lesson-04 : Constraints, Adv. Subqueries & Other Database Objects

Constraints

Constraints enforce Business Rules
Constraints prevent deletion of a table if there are dependencies

Constraint Types

PRIMARY KEY
NOT NULL
UNQIUE
CHECK
FOREIGN KEY -> AKA Referential Integrity Constraint

Defining Constraints

Can be defined at COLUMN Level or TABLE Level

[1] COLUMN Level - While creating the DB table.

```
CREATE TABLE <table_name>
(
      colname1    datatype1   Column_CONSTRAINT,
      :
);
```

OR

[2] TABLE Level - While creating the DB table.

```
CREATE TABLE <table_name>
(
      colname1    ............,
      colname2    ............,
      :
      :
      :
      CONSTRAINT emp_pk PRIMARY KEY( empID )
);
```

If we have forgotten to specify the CONSTRAINT while creating the table we can include constraint with the ALTER TABLE command.

PRIMARY KEY Constraint

Ensures that each and every row/record is identified uniquely.
NOT NULL is IMPLICITLY included in this constraint.

```
ALTER TABLE table2
ADD CONSTRAINT col1_pk PRIMARY KEY (col1);
```

NOT NULL Constraint
~~~~~~~~~~~~~~~~~~~
Ensure THERE IS data in the column.
i.e. the column data CANNOT BE EMPTY.

If we are using ALTER TABLE command to specify the NOT NULL
constraint, we need to MODIFY the column as the ADD
CONSTRAINT clause will not work.

Example:

```
ALTER TABLE customer
MODIFY custCity VARCHAR2(10) CONSTRAINT city_nn NOT NULL;
```

UNIQUE Constraint
~~~~~~~~~~~~~~~~~
Ensure the column has UNIQUE values.
However enables to have NULL value(s).

Example:

```
ALTER TABLE customer
ADD CONSTRAINT email_uk UNIQUE ( emailID );
```

CHECK Constraint
~~~~~~~~~~~~~~~~
Helps in validating the data against a condition.

Example:

```
ALTER TABLE customer
ADD CONSTRAINT age_ck CHECK ( custAge >= 18 );

ALTER TABLE customer
ADD CONSTRAINT gender_ck CHECK (custGender = 'M' or custGender = 'F')
```

FOREIGN KEY Constraint
~~~~~~~~~~~~~~~~~~~~~~
Helps in REFERENTIAL Integrity.

i.e. A foreign key column in one table refers to the primary key column
in another table.

Thus any volition will not be accepted.

```
ALTER TABLE table2
ADD CONSTRAINT ms_fk FOREIGN KEY (mStatus) REFERENCES table3 (mStatus);
```

Viewing Constraints
==========
To view the constraints of a table, use the 'user_constraints' data
dictionary.

Example:

```
SQL> desc user_constraints

SQL> SELECT constraint_name, constraint_type, search_condition
  2  FROM user_constraints WHERE table_name = 'CUSTOMER';
```

```
CONSTRAINT_NAME                CONSTRAINT SEARCH_CONDITION
------------------------------ ---------- --------------------
SYS_C0011124                   C          "CUSTNAME" IS NOT NU
                                          LL

SYS_C0011125                   P
CITY_NN                        C          "CUSTCITY" IS NOT NU
                                          LL

EMAIL_UK                       U
AGE_CK                         C          custage >= 18
GENDER_CK                      C          custgender = 'M' OR
                                          custgender = 'F'
```

NOTE: In CONSTRAINT_NAME column, any constraint name that starts with SYS_Cxxxxxx is the name given by Oracle Database Server.

However, if you give it explicitly it is more clear and better.

Follow coding style, PRIMARY KEY constraint ends with PK,
NOT NULL constraint name ends with NN
UNIQUE constraint name ends with UK
CHECK constraint name ends with CK
FOREIGH KEY constraint name ends with FK

Disable Constraint

Execute the DISABLE clause of the ALTER TABLE statement to deactivate an integrity constraint.

Example:

```
SQL> ALTER TABLE customer
  2  DISABLE CONSTRAINT email_uk;

SQL> INSERT INTO customer VALUES (4, 'Deepak', 'M', 'smartguy@gmail.com',
28, 'Pilani');
```

Enabling Constraint

Activate an integrity constraint that is currently disabled.
Using the ENABLE clause of the ALTER TABLE statement.

Example:

```
SQL> ALTER TABLE customer
  2  ENABLE CONSTRAINT email_uk;
```

Adv. SubQueries

Multi-Column Subqueries

In multi-column subqueries, each row of the main query is compared to values from multiple-row and multiple-column subquery.

Example:

```
WHERE (manager_id, department_id) IN
```

```
      Subquery
            100   90
            102   60
            124   50

Column Comparisons
~~~~~~~~~~~~~~~~~~
Column Comparisons in a multi-column subquery can be
[1] Pairwise comparisons
[2] Non-pairwise comparisons

[1] Pairwise Comparisons
    ~~~~~~~~~~~~~~~~~~~~
Example:
Displaying the details of employees who are managed by the
same manager and work in the same department as the employees
with employee ID 174 and 178

SQL> SELECT first_name, manager_id, department_id
  2  FROM employees
  3  WHERE (manager_id, department_id) IN
  4             ( SELECT manager_id, department_id FROM employees
  5               WHERE employee_id IN (174, 178) )
  6  AND employee_id NOT IN (174, 178);

FIRST_NAME           MANAGER_ID DEPARTMENT_ID
-------------------- ---------- -------------
Alyssa                      149            80
Jonathon                    149            80
Jack                        149            80
Charles                     149            80


[2] Non-Pairwise Comparisons
    ~~~~~~~~~~~~~~~~~~~~~~~~
Example:
Display the details of employees who are manged by the same
manager as the employees with ID 174 OR 141 and work in the
same department as the employees with ID 174 or 141

Example:

SELECT first_name, manager_id, department_id
FROM employees
WHERE manager_id IN
        ( SELECT manager_id FROM employees WHERE employee_id IN (174,141)
)
AND department_id IN
        ( SELECT department_id FROM employees WHERE employee_id IN
(174,141) )
AND employee_id NOT IN (174, 141);

FIRST_NAME           MANAGER_ID DEPARTMENT_ID
-------------------- ---------- -------------
Douglas                     124            50
Donald                      124            50
Kevin                       124            50
Alana                       124            50
```

```
Peter                          124            50
Randall                        124            50
Curtis                         124            50
Charles                        149            80
Jack                           149            80
Jonathon                       149            80
Alyssa                         149            80
```

Scalar Subquery Expression

SCALAR = Single Value
A Scalar Subquery Expression is a subquery that returns exactly one column value for one row.

Can be used in:
- DECODE and CASE condition expressions
- All clauses of SELECT except GROUP BY clause
- The SET clause and the WHERE clause of an UPDATE statement

Example - Using a Subquery in the CASE function

```
SELECT employee_id, last_name,
(  CASE
     WHEN department_id = (SELECT department_id FROM departments
                           WHERE location_id = 1800) THEN 'Canada'
     ELSE 'USA'
   END
) Location
FROM employees;
```

```
EMPLOYEE_ID LAST_NAME                 LOCATI
----------- ------------------------- ------
        100 King                      USA
        101 Kochhar                   USA
        102 De Haan                   USA
        103 Hunold                    USA
        104 Ernst                     USA
        201 Hartstein                 Canada
        202 Fay                       Canada
        203 Mavris                    USA
        204 Baer                      USA
```

Correlated Subqueries

Correlated subqueries are used for row-by-row processing.
Each subquery is executed once for every row in the outer/main query.

The steps in which the processing of a Correlated Subquery works is as follows:

```
      [1] GET - Candidate row from the outer/main query
                        |
      [2] EXECUTE - inner query (subquery) using candidate row value
                        |
      [3] USE - values from the inner query to qualify or disqualify
              candidate row.
```

Syntax:

```
SELECT column1, column2, ...
FROM table1 outer
WHERE column1 operator
            (SELECT colum1, column2
             FROM table2
             WHERE expr1 = outer.expr2);
```

The subquery REFERENCE a column from a table in the parent/main query.

Example:
Find all employees who earn more than the average salary in their department.

```
SELECT last_name, salary, department_id
FROM employees outer
WHERE salary > ( SELECT AVG(salary)
             FROM employees
             WHERE department_id = outer.department_id);
```

```
SQL> SELECT last_name, salary, department_id
  2  FROM employees outer
  3  WHERE salary > ( SELECT AVG(salary)
  4   FROM employees
  5   WHERE department_id = outer.department_id);

LAST_NAME                     SALARY DEPARTMENT_ID
------------------------- ---------- -------------
King                           24000            90
Hunold                          9000            60
Ernst                           6000            60
Greenberg                      12008           100
Faviet                          9000           100
Raphaely                       11000            30
```

Using EXISTS Operator
==========

The EXISTS operator tests for existence of rows in the results set of the subquery.

If a subquery row value is found:
- The search does not continue in the inner query
- The condition is flagged TRUE

If a subquery row value is not found:
- The condition is flagged FALSE
- The search continues in the inner query

Example:
Find employees who have at least one person reporting to them.

```
SELECT employee_id, last_name, job_id, department_id
FROM employees outer
WHERE EXISTS (SELECT 'X' FROM employees WHERE manager_id =
outer.employee_id);
```

```
SQL> SELECT employee_id, last_name, job_id, department_id
  2  FROM employees outer
  3  WHERE EXISTS (SELECT 'X' FROM employees WHERE manager_id =
outer.employee_id);

EMPLOYEE_ID LAST_NAME                 JOB_ID     DEPARTMENT_ID
----------- ------------------------- ---------- -------------
        100 King                      AD_PRES               90
        101 Kochhar                   AD_VP                 90
        102 De Haan                   AD_VP                 90
        103 Hunold                    IT_PROG               60
        108 Greenberg                 FI_MGR               100
        114 Raphaely                  PU_MAN                30
        120 Weiss                     ST_MAN                50
```

Using NOT EXISTS Operator
==========

Find all departments that do not have any employee.

Example:

```
SELECT department_id, department_name
FROM departments d
WHERE NOT EXISTS (SELECT 'X' FROM employees WHERE department_id =
d.department_id);
```

```
SQL> SELECT department_id, department_name
  2  FROM departments d
  3  WHERE NOT EXISTS (SELECT 'X' FROM employees WHERE department_id =
d.department_id);

DEPARTMENT_ID DEPARTMENT_NAME
------------- ------------------------------
          120 Treasury
          130 Corporate Tax
          140 Control And Credit
          150 Shareholder Services
          160 Benefits
          170 Manufacturing
```

Using Correlated UPDATE
==========

The syntax for correlated update is as follows:

Syntax:

```
UPDATE table1 alias1
SET column = (   SELECT expression
           FROM table2 alias2
           WHERE alias1.column = alias2.column );
```

If the subquery reference a column of the parent query for updation then we term this as Correlated Update.

Example

- Denormalize the EMPLOYEES table by adding a column to store the department name.
- Populate the table by using a correlated update.

```
ALTER TABLE emp_temp
ADD(department_name VARCHAR2(16));

UPDATE emp_temp e
SET department_name =
                 ( SELECT department_name
                  FROM departments d
                  WHERE e.department_id = d.department_id );
```

Correlated DELETE
==========
If the subquery reference a column of the parent query for deletion then we term this as Correlated Delete.

The syntax for using the correlated delete is as follows:

```
DELETE FROM table1 alias1
WHERE column operator
            ( SELECT expression
              FROM table2 alias2
              WHERE alias1.column = alias2.column );
```

Use a correlated subquery to delete rows in one table based on rows from another table.

Example:
Use a correlated subquery to delete only those rows
from the EMPLOYEES table that also exist in the
EMP_HISTORY table.

```
DELETE FROM employees E
WHERE employee_id =
               ( SELECT employee_id
                 FROM emp_history
                 WHERE employee_id = E.employee_id );
```

==========
Database Objects
==========

Object	Description
Table	Basic unit of storage, composed of rows & columns
View	Logically represents sub set of data from one or more tables
Sequence	Generate a sequence for primary key values
Index	Improve the performance of queries
Synonym	Alternative name for an object

What is a View?
~~~~~~~~~~~~~~
Refer to PPTs
- NOTE: It is referred a Virtual Table
~~~~~~~~~~~~~~

Why use Views?
~~~~~~~~~~~~~
• To restrict data access
• To make complex queries easy
• To provide data independence
• To present different views of the same data

Simple v/s Complex Views
~~~~~~~~~~~~~~~~~~~~~~~~
Features	Simple	Complex
No. of table	One	One or more
Contains functions	No	Yes
Contains group of data	No	Yes
DML operations	Yes	No

Creating a View
~~~~~~~~~~~~~~~
Done using CREATE VIEW statement.

We embed a subquery within a CREATE VIEW statement.
The subquery can contain complex SELECT syntax

Example-1:

```
SQL> CREATE VIEW empvu1
  2  AS SELECT employee_id, first_name, last_name, salary
  3     FROM employees WHERE department_id IN (100, 110);
```

Example-2:
```
SQL> CREATE VIEW salvu50
  2  AS SELECT employee_id ID_NUMBER, last_name NAME, salary*12
ANN_SALARY FROM employees
  3     WHERE department_id = 50;

View created.

SQL> select * from salvu50;

 ID_NUMBER NAME                      ANN_SALARY
---------- ------------------------- ----------
       120 Weiss                          96000
       121 Fripp                          98400
       122 Kaufling                       94800
       123 Vollman                        78000
       124 Mourgos                        69600
       125 Nayer                          38400
```

Example-3 : Complex View
```
CREATE VIEW  deptsumsal_vu ( dname, minsal, maxsal, avgsal )
AS SELECT d.department_name, MIN(e.salary), MAX(e.salary), AVG(e.salary)
   FROM employees e, departments d
   WHERE e.department_id = d.department_id
   GROUP BY d.department_name;

View created.
```
~~~~~~~~~~~~~~~

```
SQL> SELECT * FROM deptsumsal_vu;

DNAME                              MINSAL     MAXSAL     AVGSAL
------------------------------ ---------- ---------- ----------
Administration                       4400       4400       4400
Accounting                           8300      12008      10154
Purchasing                           2500      11000       4150
Human Resources                      6500       6500       6500
IT                                   4200       9000       5760
Public Relations                    10000      10000      10000
Executive                           17000      24000 19333.3333
Shipping                             2100       8200 3475.55556
Sales                                6100      14000 8955.88235
Finance                              6900      12008 8601.33333
Marketing                            6000      13000       9500
```

NOTE: The above created view is a complex view, inasmuch it uses the GROUP functions and the GROUP BY clause; besides accessing the data from two tables.

Rules for Performing DML Operations on Views

- You can perform DML operations on simple views.

- You cannot remove a row if the view contains the following:
  - Group functions
  - A GROUP BY clause
  - The DISTINCT keyword
  - The pseudo column ROWNUM keyword

- You cannot modify data in a view if it contains:
  - Group functions
  - A GROUP BY clause
  - The DISTINCT keyword
  - The pseudo column ROWNUM keyword
  - Columns defined by expressions

- You cannot add data through a view if the view includes:
  - Group functions
  - A GROUP BY clause
  - The DISTINCT keyword
  - The pseudo column ROWNUM keyword
  - Columns defined by expressions
  - NOT NULL columns in the base tables that are not selected by the view

Inline Views

- An inline view is a subquery with an alias (or correlation name) that you can use within a SQL statement.
- A named subquery in the FROM clause of the main query is an example of an inline view.
- An inline view is NOT a SCHEMA Object.
- Column of inline view can be used in outer query

Example:
List those employees whose salary is more than the
average salary of employees

```
SQL> SELECT first_name || ' ' || last_name "NAME", Salary, AvgSalary
  2  FROM employees, (SELECT AVG(salary) AvgSalary FROM employees)
  3  WHERE salary > avgsalary;

NAME                                               SALARY  AVGSALARY
-------------------------------------------------- ---------- ----------
Steven King                                         24000 6461.83178
Neena Kochhar                                       17000 6461.83178
Lex De Haan                                         17000 6461.83178
```

Using the WITH CHECK OPTION Clause

- You can ensure that DML operations performed on the view stay within the domain of the view by using the WITH CHECK OPTION clause.

- Any attempt to change the department number for any row in the view fails because it violates the WITH CHECK OPTION constraint.

Example:

```
SQL> CREATE VIEW empvu110
  2  AS SELECT * FROM employees WHERE department_id = 110
  3  WITH CHECK OPTION CONSTRAINT empvu110_ck;
```

If we try to change the department_id from 110 to any other department_id we get the following error:

```
SQL> UPDATE empvu110
  2  SET department_id = 100;
UPDATE empvu110
       *
ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause violation
```

Denying DML Operations

- You can ensure that no DML operations occur by adding the WITH READ ONLY option to your view definition.
- Any attempt to perform a DML on any row in the view results in an Oracle server error.

Example:

```
SQL> CREATE VIEW empvu10
  2  AS SELECT * FROM employees WHERE department_id = 10
  3  WITH READ ONLY;

SQL> DELETE FROM empvu10;
DELETE FROM empvu10
            *
ERROR at line 1:
ORA-42399: cannot perform a DML operation on a read-only view
```

Removing a View

To remove a view use the DROP VIEW statement.

Example:

```
SQL> DROP VIEW salvu50;

View dropped.
```

Indexes

What is an Index?

An index:

- Is a schema object
- Is used by the Oracle server to speed up the retrieval of rows by using a pointer
- Can reduce disk I/O by using a rapid path access method to locate data quickly
- Is independent of the table it indexes
- Is used and maintained automatically by the Oracle server

How are Indexes Created?

- AUTOMATICALLY: A unique index is created automatically when you define a PRIMARY KEY or UNIQUE constraint in a table definition.
- MANUALLY: Users can create non-unique indexes on columns to speed up access to the rows.

Creating an Index

Done using CREATE INDEX statement

Syntax:

```
CREATE INDEX index_name
ON table (column[, column]...);
```

Example:

```
SQL> CREATE INDEX emp_lastname_idx
  2  ON employees (last_name);

Index created.
```

When to Create an Index?

You should create an index if:

- A column contains a wide range of values
- A column contains a large number of null values
- One or more columns are frequently used together in a WHERE clause or a join condition
- The table is large and most queries are expected to retrieve less than 2 to 4 percent of the rows

When NOT to Create an Index?
~~~~~~~~~~~~~~~~~~~~~~~~~~~
It is usually not worth creating an index if:
• The table is small
• The columns are not often used as a condition in the query
• Most queries are expected to retrieve more than 4 percent of the rows
• The table is updated frequently
• The indexed columns are referenced as part of an expression

Removing an Index
~~~~~~~~~~~~~~~~~
To remove an Index, the DROP INDEX statment is used.

```
SQL> DROP INDEX emp_lastname_idx;

Index dropped.
```

==========
Sequences
==========
What is a Sequence?
~~~~~~~~~~~~~~~~~~~
A sequence:
• Automatically generates unique numbers
• Is a SHARABLE OBJECT
• Is typically used to create a primary key value
• Replaces application code
• Speeds up the efficiency of accessing sequence values when cached

Creating a Sequence
~~~~~~~~~~~~~~~~~~~
To create a sequence the CREATE SEQUENCE statement is used.

Syntax:
```
CREATE SEQUENCE sequence
[INCREMENT BY n]
[START WITH n]
[{MAXVALUE n | NOMAXVALUE}]
[{MINVALUE n | NOMINVALUE}]
[{CYCLE | NOCYCLE}]
[{CACHE n | NOCACHE}];
```

Example:
```
SQL> CREATE SEQUENCE my_seq
  2  START WITH 5
  3  INCREMENT BY 2
  4  MAXVALUE 99
  5  NOCACHE
  6  NOCYCLE;

Sequence created.
```

Using a Sequence
~~~~~~~~~~~~~~~~
NEXTVAL and CURRVAL Pseudo columns
• NEXTVAL returns the next available sequence value.
  It returns a unique value every time it is referenced, even for
  different users.
~~~~~~~~~~~~~~~~

• CURRVAL obtains the current sequence value.
• NEXTVAL must be issued for that sequence before CURRVAL contains a value.

Example: Use sequence 'my_seq' NEXTVAL in the INSERT statement.

```
SQL> INSERT INTO table2 VALUES(my_seq.NEXTVAL, 'Raju', 'M', 'S');

1 row created.

SQL> INSERT INTO table2 VALUES(my_seq.NEXTVAL, 'Rani', 'F', 'M');

1 row created.

SQL> SELECT * FROM table2;

      COL1 COL2                 C M
---------- -------------------- - -
         5 Raju                 M S
         7 Rani                 F M
```

• View the current value for the 'my_seq' sequence.

```
SQL> SELECT my_seq.CURRVAL FROM dual;

   CURRVAL
----------
         7
```

Since, the last inserted value was 7, we are able to get that value

Using a Sequence
~~~~~~~~~~~~~~~~
• Caching sequence values in memory gives faster access to those values.
• Gaps in sequence values can occur when:
  – A rollback occurs
  – The system crashes
  – A sequence is used in another table
• If the sequence was created with NOCACHE, view the next available value,
  by querying the USER_SEQUENCES table.

Modify a Sequence
~~~~~~~~~~~~~~~~~
Change the increment value, maximum value, minimum value, cycle option, or cache option.

Example:

```
ALTER SEQUENCE my_seq
INCREMENT BY 5
MAXVALUE 999
CACHE
NOCYCLE;
```

Removing a Sequence
~~~~~~~~~~~~~~~~~~~
• Remove a sequence by using the DROP SEQUENCE statement.
• Once removed, the sequence can no longer be referenced.

```
DROP SEQUENCE my_seq;
```
~~~~~~~~~~~~~~~~~~~

Synonyms

Simplify access to objects by creating a synonym (another name for an object).

With synonyms, you can:
- Ease referring to a table owned by another user
- Shorten lengthy object names

Syntax:
```
CREATE [PUBLIC] SYNONYM synonym
FOR object;
```

Creating and Removing Synonyms

- Create a shortened name for the MY_SEQ sequence.

Example:
```
SQL> CREATE SYNONYM ms FOR my_seq;
```

Checking the synonym
```
SQL> SELECT ms.CURRVAL FROM dual;

   CURRVAL
----------
         9
```

- Drop a synonym.

Example:
```
SQL> DROP SYNONYM ms;
```

Multi-Row Subqueries

- Return more than one row
- Use multiple-row comparison operators

Mutli-Row Operators

Operator	Description
IN	Equal to any member in the list
ANY	Compare value to each value returned by the subquery
ALL	Compare value to every value returned by the subquery

NOTE:

< ANY	Means, LESSER than the HIGHEST value returned by the subquery
< ALL	Means, LESSER than the LOWEST value returned by the subquery

To understand the above, let us first get the salary values for 'MANAGER'

```
SQL> SELECT sal FROM emp WHERE job = 'MANAGER';

       SAL
----------
      2975
      2850
      2450
```

Examples:

```
SQL> SELECT ename, sal FROM emp
  2  WHERE sal < ANY ( SELECT sal FROM emp WHERE job = 'MANAGER' );

ENAME             SAL
---------- ----------
SMITH              800
JAMES              950
ADAMS             1100
WARD              1250
MARTIN            1250
MILLER            1300
TURNER            1500
ALLEN             1600
CLARK             2450
BLAKE             2850

SQL> SELECT ename, sal FROM emp
  2  WHERE sal < ALL ( SELECT sal FROM emp WHERE job = 'MANAGER' );

ENAME             SAL
---------- ----------
ALLEN             1600
TURNER            1500
MILLER            1300
WARD              1250
MARTIN            1250
ADAMS             1100
JAMES              950
SMITH              800
```