18csc303j -Database Management Systems

Dr.B.Muruganantham
Associate Professor
Department of Computing Technology
SRM Institute of Science and Technology

Outline of the Presentation

S-1 SLO-1 : Basics of SQL-DDL,DML,DCL,TCL

SLO-2 : Structure Creation, alternation

S-2 SLO-1 & SLO-2: Defining Constraints-Primary Key, Foreign Key, Unique, not null, check, IN operator

S-3 SLO-1 : Functions-aggregation functions

SLO-2 : Built-in Functions- Numeric, Date, String functions, Set operations

S 4-5 SLO-1 & SLO-2 : Lab 7 : Join Queries on sample exercise.

S-6 SLO-1 & SLO-2 : Sub Queries, correlated sub queries

S-7 SLO-1 & SLO-2 : Nested Queries, Views and its Types

S-8 SLO-1 : Transaction Control Commands

SLO-2 : Commit, Rollback, Savepoint

S-9-10 SLO-1 & SLO-2 : Lab 8: Set Operators & Views.

S-11 SLO-1 & SLO-2 : PL/SQL Concepts- Cursors

S-12 SLO-1 & SLO-2 : Stored Procedure, Functions Triggers and Exceptional Handling

S-13 SLO-1 & SLO-2 : Query Processing

S-14-15SLO-1 & SLO-2 : Lab9: PL/SQL Conditional and Iterative Statements

S-1 SLO-1 : Basics of SQL-DDL,DML,DCL,TCL

Structured Query Language (SQL)

- ✔ SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987
- ✔ Common language for all Databases
- ✔ Fourth generation Language
- ✔ Non procedural Language
- ✔ Commands like an normal English statements
- ✔ SQL is not a case sensitive language
- ✔ All SQL statements should ended with terminator , the default terminator is semi-colon (;)
- ✔ Based on the operation SQL divided into three categories
  - DDL (Data Definition Language)
  - DML (Data Manipulation Language)
  - DCL (Data Control Language)

S-1 SLO-2 : Structure Creation, alternation

Data Definition Language (DDL)

- ✔ DDL is the subset of SQL and part of DBMS
- ✔ DDL relates only with base tables structure and it is no where relates with the information stored in the table.
- ✔ **Note : All the DDL command statements are AUTO COMMIT Statements**
- ✔ DDL consists of the following commands
  - CREATE
  - ALTER
  - DROP
  - TRUNCATE

S-1 SLO-2 : Structure Creation, alternation

Data Definition Language (DDL)

CREATE COMMAND

Used to create a new object / schema with a defined structure

Syntax :

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

Example :

```
CREATE TABLE EMP
      (EMPNO NUMBER(4) NOT NULL,  ENAME VARCHAR2(10),  JOB VARCHAR2(9),
       MGR NUMBER(4), HIREDATE DATE, SAL NUMBER(7, 2), COMM NUMBER(7, 2),
       DEPTNO NUMBER(2));
```

S-1 SLO-2 : Structure Creation, alternation

Data Definition Language (DDL)

ALTER COMMAND

- ✔ Alter command used to modify the base table structure
- ✔ Using this command
  - a new column can be added with restrictions
  - column data width can be increased / decreased with restrictions
  - a column can be dropped
- ✔ Two key words are using in this command
  - ADD
  - MODIFY

S-1 SLO-2 : Structure Creation, alternation

Data Definition Language (DDL)

ALTER COMMAND

SYNTAX

```
ALTER TABLE table_name ADD / MODIFY column_name datatype;
```

EXAMPLE 1: To add a new column in a table

```
ALTER TABLE emp ADD phone_no number(10);
```

EXAMPLE 2 : TO modify the existing column data width

```
ALTER TABLE emp MODIFY phone_no number(13);
```

S-1 SLO-2 : Structure Creation, alternation

Data Definition Language (DDL)

ALTER COMMAND

Syntax to DROP a column

```
ALTER TABLE table_name DROP column column_name;
```

Example :

```
ALTER TABLE emp DROP column phone_no;
```

S-1 SLO-2 : Structure Creation, alternation

Data Definition Language (DDL)

DROP COMMAND

It is used to remove the base table with records (information) from database permanently.

Syntax:

```
DROP TABLE table_name ;
```

Example:

```
DROP TABLE emp;
```

S-1 SLO-2 : Structure Creation, alternation

TRUNCATE COMMAND

Truncate command used to delete the records (information) from the base table permanently and keeps the structure of the base table alone

Syntax:

```
TRUNCATE TABLE table_name;
```

Example:

```
TRUNCATE TABLE emp;
```

S-1 SLO-2 : Structure Creation, alternation

DML Commands are relates only with base table information (value in an attribute)
There are four commands in DML:

1. INSERT
2. UPDATE
3. DELETE
4. SELECT

- ✓ Where clause (Conditional retrieval)
- ✓ Order by clause (Retrieval in Ascending or Descending Order)
- ✓ Group by clause (Retrieval of distinct values by considering groups)
- ✓ Having clause (Followed by Group by clause with COUNT function)

S-1 SLO-2 : Structure Creation, alternation

INSERT COMMAND

- ✔ It relates only with new records.
- ✔ Only one row can be inserted at a time
- ✔ Multiple rows can be inserted using “&” symbol one by one
- ✔ Can insert in selected columns with some restrictions
- ✔ Must follow the order of the column specified in the query statement

S-1 SLO-2 : Structure Creation, alternation

INSERT COMMAND

Syntax:

```
INSERT INTO <table_name> (column_name1 <datatype>,
                          column_name2 <datatype>,
                          . . . ,
                          column_name_n <datatype>)
                     VALUES
                          (value1,
                           value2,
                           . . . ,
                          value n);
```

Note :

- Number values can be inserted as integer or float
- Char and Date values must be in single quote

S-1 SLO-2 : Structure Creation, alternation

INSERT COMMAND

Example 1: To insert a record using all fields in EMP table

```
INSERT INTO EMP VALUES (7369, 'SMITH',  'CLERK', 7902, '17-12-1980‘, 800, NULL, 20);
                    (OR)
INSERT INTO EMP (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO)
VALUES (7369, 'SMITH',  'CLERK', 7902, '17-12-1980',  800, NULL, 20);
```

Example 2: To insert a record using selected fields in EMP table

```
INSERT INTO EMP (EMPNO, ENAME) VALUES (7499, 'ALLEN);
```

Note : When a record is inserted using selected fields, it must include NOT NULL and Primary key fields.

S-1 SLO-2 : Structure Creation, alternation

Example 3: To insert multiple records using all fields in EMP table

```
INSERT INTO EMP values
(&EMPNO,’&ENAME’,’&JOB’,&MGR,’&HIREDATE’,&SAL,&COMM,&DEPTNO) ;
```

NOTE : ‘&’ (Ampersand) symbol used to ask

Enter value for followed by the string during runtime.

The input value will be store in the appropriate field using bind variable (:OLD and :NEW)

S-1 SLO-2 : Structure Creation, alternation

Update command

- ✔ It works with only existing records
- ✔ It works only column wise
- ✔ It is used to modify the column values (increase / decrease / change)

Syntax

UPDATE <table_name> set <field_name> = value [where <condition>];

Note : Update command without where condition will update all the records.

Update command with where condition will update the records which are satisfy the condition

Example 1:

UPDATE emp set comm = 2000 ; (Update all the records in EMP table)

Example 2 :

Update emp set comm = 1000 where empno = 7369;

(Update the records having the empno as 7369)

S-1 SLO-2 : Structure Creation, alternation

Delete command

- ✔ It works only with existing records
- ✔ It works only with row wise
- ✔ It not possible to delete a single column in a row

Syntax

DELETE from <table_name> [where <condition>];

Note : Delete command with out where condition will delete all the records in the table.

Delete command with where condition will delete the selected records which are satisfy the condition.

Example 1: DELETE from emp; (All records will be deleted from emp)

Example 2: DELETE from emp where empno = 7369;

(Those records holding the value in the field empno as 7369 will be deleted)

S-1 SLO-2 : Structure Creation, alternation

SELECT COMMAND

- ✔ Works with existing records
- ✔ Works with row wise and column wise
- ✔ Works with multiple tables
- ✔ Never affect / change / update / modification in the data base
- ✔ Using this command , we can select a column , multiple columns, all columns, single row, multiple row, all rows
- ✔ Specially called as "QUERY STATEMENT"

S-1 SLO-2 : Structure Creation, alternation

SELECT COMMAND

Syntax

```
SELECT column_list FROM table-name
                    [WHERE Clause]
                    [GROUP BY clause]
                    [HAVING clause]
                    [ORDER BY clause];
```

NOTE : To retrieve all the column from the table ‘ * ’ symbol can be used instead of specifying the column_list.

S-1 SLO-2 : Structure Creation, alternation

SELECT COMMAND

Example 1: To retrieve all the columns and rows from emp table

SELECT * from emp; (‘*’ stands from all columns and rows)

Example 2: To select retrieve the specific columns from all rows

SELECT empno,ename from emp;

Select command with where clause

Example 3: To retrieve the records from emp table which record holds the salary value greater than 1000;

SELECT * from emp WHERE sal> 1000;

Example 4: To retrieve the columns empno and ename from emp table which records holds the value as CLERK in job column.

SELECT empno, ename from emp WHERE job = ‘CLERK’

S-1 SLO-2 : Structure Creation, alternation

SELECT COMMAND

Select command with order by clause

Example 5 : To retrieve the records from emp table in ascending order using empno

```
SELECT * from emp order by empno asc;
                (OR)
SELECT * from emp order by empno;
```

Example 6: To retrieve the records from emp table in ascending order using job and empno

```
SELECT * from emp order by job,empno asc;
                (OR)
SELECT * from emp order by job,empno;
```

NOTE : Ascending order is default condition, no need to specify

S-1 SLO-2 : Structure Creation, alternation

SELECT COMMAND

Select command with order by clause

Example 7 : To retrieve the records from emp table in descending order using empno.

```
SELECT * from emp order by empno desc;
```

Example 8: To retrieve the records from emp table in descending order using job and empno

```
SELECT * from emp order by job desc,empno desc;
```

Example 8: To retrieve the records from emp table in ascending order using job and descending order empno

```
SELECT * from emp order by job asc,empno desc;
```

S-1 SLO-2 : Structure Creation, alternation

SELECT COMMAND

Select command with group by clause

Example 9: To retrieve the different jobs from emp table

SELECT job from emp group by job;

Example 10: To retrieve the different jobs and its average salary from emp table

SELECT job, avg(sal) from emp group by job;

Select command with group by and having clause

Example 11: To retrieve the different jobs from emp table where the total numbers in a group is greater than 2;

SELECT job from emp group by job having count(job) >2;

NOTE : Count is built-in group function

S-1 SLO-2 : Structure Creation, alternation

Data Control Languages

✔Used to give / get back / control the privileges of an object by the owner

GRANT : To give access privileges of an object to other user by the owner

Syntax : GRANT [ALL / INSERT /UPDATE /DELETE /SELECT]

on <OBJECT_NAME> to <USER_NAME>;

Example: GRANT all on emp to scott;

REVOKE : To get back all the privileges from the user who has been granted

Syntax : REVOKE [ALL / INSERT /UPDATE /DELETE /SELECT]

on <OBJECT_NAME> from <USER_NAME>;

Example: REVOKE all on emp from scott;

S-2 SLO-1 & SLO-2: Defining Constraints-Primary Key, Foreign Key, Unique, Not null, Check, IN operator

Constraint

✔ Purpose

Use a constraint to define an integrity constraint--a rule that restricts the values in a database.

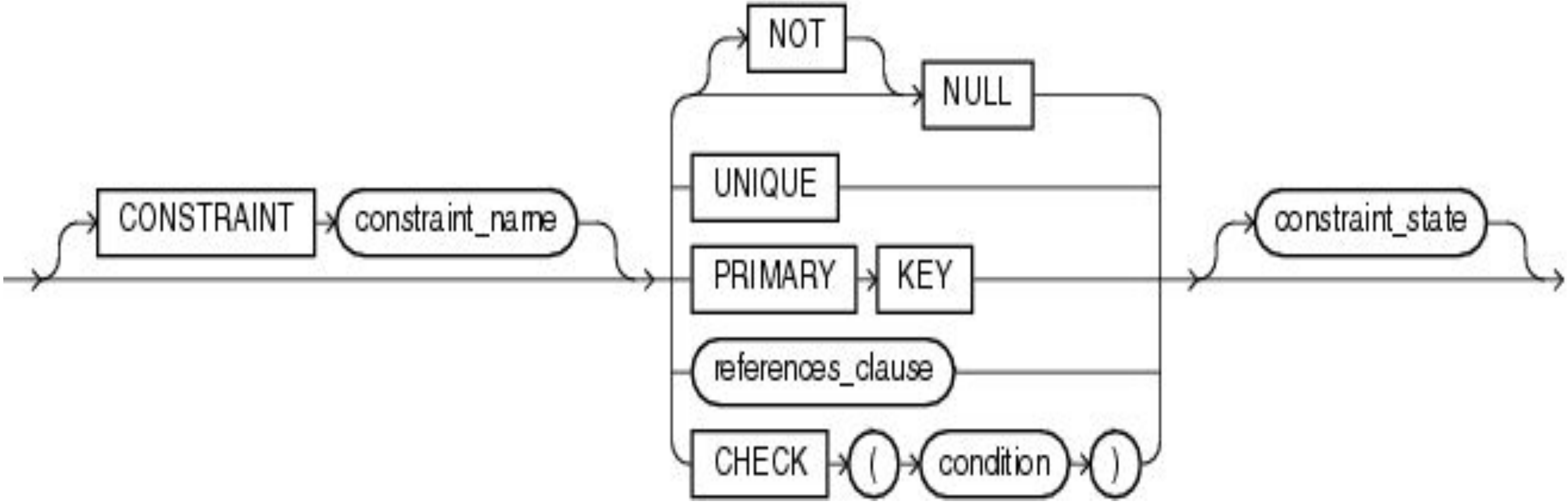
Oracle Database lets you create five types of constraints and lets you declare them in two ways.

✔ There five types of integrity constraint

- NOT NULL constraint
- Unique constraint
- Primary key constraint
- Foreign key constraint
- Check constraint

S-2 SLO-1 & SLO-2: Defining Constraints-Primary Key, Foreign Key, Unique, Not null, Check, IN operator

General Syntax for Constraints:



S-2 SLO-1 & SLO-2: Defining Constraints-Primary Key, Foreign Key, Unique, Not null, Check, IN operator

Not Null Constraint

- ✔ NOT NULL constraint prohibits a database value from being null.
- ✔ Permits duplicate values
- ✔ To satisfy a NOT NULL constraint, every row in the table must contain a value for the column.

Restrictions on NOT NULL Constraints

- ✔ NOT NULL constraints are subject to the following restrictions:
- ✔ You cannot specify NULL or NOT NULL in a view constraint.
- ✔ You cannot specify NULL or NOT NULL for an attribute of an object. Instead, use a CHECK constraint with the IS [NOT] NULL condition.

S-2 SLO-1 & SLO-2: Defining Constraints-Primary Key, Foreign Key, Unique, Not null, Check, IN operator

Syntax for Not Null constraint

```
Create table <table_name> ( column_1 datatype Constraint <constraint_name> 
<constraint_type>,
                    column_2 datatype,
                    ………..
                    ……….
                    column_n datatype);
```

Example

```
Create table emp ( empno number constraint my_cons_NN not null,
                ename varchar2(10), sal number(10), dept varchar2(10));
```

Note : Constraints may specified without name also, in that case system automatically assign some random name.

```
Create table emp ( empno number(10)  not null,  ename varchar2(10),
          sal number(10), dept varchar2(10));
```

S-2 SLO-1 & SLO-2: Defining Constraints-Primary Key, Foreign Key, Unique, Not null, Check, IN operator

Unique Constraint

- ✔ A unique constraint designates a column as a unique key.
- ✔ A composite unique key designates a combination of columns as the unique key.
- ✔ To satisfy a unique constraint, no two rows in the table can have the same value for the unique key.
- ✔ Unique constraint allows null values and it allows more number of null values (Two null values are always not equal).

S-2 SLO-1 & SLO-2: Defining Constraints-Primary Key, Foreign Key, Unique, Not null, Check, IN operator

Restrictions on Unique Constraint

- ✔ None of the columns in the unique key can be of LOB, LONG, LONG RAW, VARRAY, NESTED TABLE, OBJECT, REF, or user-defined type.
- ✔ A composite unique key cannot have more than 32 columns.
- ✔ You cannot designate the same column or combination of columns as both a primary key and a unique key.
- ✔ You cannot specify a unique key when creating a subview in an inheritance hierarchy. The unique key can be specified only for the top-level (root) view.

S-2 SLO-1 & SLO-2: Defining Constraints-Primary Key, Foreign Key, Unique, Not null, Check, IN operator

Syntax for unique constraint

```
Create table <table_name> ( column_1 datatype Constraint <constraint_name>
     <constraint_type>, column_2 datatype,
                         ………..
                         ……….
                         column_n datatype);
```

Example

```
Create table emp ( empno number(10) constraint my_cons_UK unique,
                    ename varchar2(10), sal number(10), dept varchar2(10));
```

Example for Composite Unique Key

```
Create table emp ( empno number(10), ename varchar2(10), sal                    number(10), dept varchar2(10),
     Constraint my_unique_key unique (empno,ename));
```

S-2 SLO-1 & SLO-2: Defining Constraints-Primary Key, Foreign Key, Unique, Not null, Check, IN operator

Primary Key

- ✔ A primary key constraint designates a column as the primary key of a table or view.
- ✔ A composite primary key designates a combination of columns as the primary key.
- ✔ A primary key constraint combines a NOT NULL and unique constraint in one declaration
- ✔ Therefore, to satisfy a primary key constraint:
  - No primary key value can appear in more than one row in the table.
  - No column that is part of the primary key can contain a null.

S-2 SLO-1 & SLO-2: Defining Constraints-Primary Key, Foreign Key, Unique, Not null, Check, IN operator

Restrictions on Primary Key Constraints

- ✔ A table or view can have only one primary key.
- ✔ None of the columns in the primary key can be LOB , LONG , LONG RAW , VARRAY, NESTED TABLE , BFILE, REF, TIMESTAMP WITH TIME ZONE, or user-defined type.
- ✔ The size of the primary key cannot exceed approximately one database block.
- ✔ A composite primary key cannot have more than 32 columns.
- ✔ You cannot designate the same column or combination of columns as both a primary key and a unique key.

S-2 SLO-1 & SLO-2: Defining Constraints-Primary Key, Foreign Key, Unique, Not null, Check, IN operator

Syntax for Primary key constraint

```
Create table <table_name> ( column_1 datatype Constraint
     <constraint_name> <constraint_type>, column_2 datatype,
                           ………..
                           ……….
                           column_n datatype);
```

Example

```
Create table emp ( empno number(10) constraint my_cons_PK  primary key,
                       ename varchar2(10), sal number(10), dept varchar2(10));
```

Example for composite primary key

```
Create table emp ( empno number(10, ename varchar2(10),
                    sal number(10), dept varchar2(10),
                    constraint my_cons_CPK primary key (empno,dept));
```

S-2 SLO-1 & SLO-2: Defining Constraints-Primary Key, Foreign Key, Unique, Not null, Check, IN operator

Foreign key constraint

- ✔ A foreign key constraint (also called a referential integrity constraint) designates a column as the foreign key and establishes a relationship between that foreign key and a specified primary or unique key, called the referenced key.
- ✔ A composite foreign key designates a combination of columns as the foreign key.
- ✔ The table or view containing the foreign key is called the child object, and the table or view containing the referenced key is called the parent object.
- ✔ The foreign key and the referenced key can be in the same table or view.
- ✔ To satisfy a composite foreign key constraint, the composite foreign key must refer to a composite unique key or a composite primary key in the parent table or view, or the value of at least one of the columns of the foreign key must be null.
- ✔ You can define multiple foreign keys in a table or view

S-2 SLO-1 & SLO-2: Defining Constraints-Primary Key, Foreign Key, Unique, Not null, Check, IN operator

Restrictions on Foreign Key Constraints

- ✔ None of the columns in the foreign key can be of LOB, LONG, LONG RAW, VARRAY, NESTED TABLE, BFILE, REF, TIMESTAMP WITH TIME ZONE, or user-defined type.
- ✔ The referenced unique or primary key constraint on the parent table or view must already be defined.
- ✔ A composite foreign key cannot have more than 32 columns.
- ✔ The child and parent tables must be on the same database.

S-2 SLO-1 & SLO-2: Defining Constraints-Primary Key, Foreign Key, Unique, Not null, Check, IN operator

Syntax for Foreign Key

```
CREATE TABLE <child_table_name> ( column_1 datatype,
                                  column_1 datatype ,
                                 -------
                                                                                     Dept_id references
    <parent_table_name>                                        (Parent_table_Primary key));
```

Example

```
CREATE TABLE emp ( empno number (10) Primary key,
                   ename varchar2(25),
                   salary number(10,2),
                   dept_id references DEPT (DEPT_ID));
```

An entity emp created with foreign key constraint referencing dept entity primary key attribute dept_id.

S-2 SLO-1 & SLO-2: Defining Constraints-Primary Key, Foreign Key, Unique, Not null, Check, IN operator

Check Constraint

- ✔ A check constraint lets you specify a condition that each row in the table must satisfy.
- ✔ To satisfy the constraint, each row in the table must make the condition either TRUE or NULL

Restrictions on Check Constraints

- ✔ You cannot specify a check constraint for a view.
- ✔ The condition of a check constraint can refer to any column in the table, but it cannot refer to columns of other tables.

S-2 SLO-1 & SLO-2: Defining Constraints-Primary Key, Foreign Key, Unique, Not null, Check, IN operator

Syntax for Check Constraint

```
Create table <table_name> ( column_1 datatype Constraint <constraint_name>
    <constraint_type> (condition),
                        column_2 datatype,
                        ………..
                        ……….
                        column_n datatype);
```

Example

```
Create table emp ( empno number(10) , ename varchar2(10),
                sal number(10) constraint my_cons_ck (sal >10000)                dept
    varchar2(10));
```

S3 SLO-1 : Functions - aggregation functions

Aggregation Functions

- ✔ MAX – To find the number of the maximum values in the SQL table.
- ✔ MIN – Number of the minimum values in the table.
- ✔ COUNT – Get the number of count values in the SQL table.
- ✔ AVG – Find average values in the SQL table.
- ✔ SUM – Return the summation of all non-null values in the SQL table.
- ✔ DISTINCT – Return the distinct values of a column.

S3 SLO-1 : Functions - aggregation functions

Note : For all the examples , consider the “EMP” Table

- ✔ Max Function

  Syntax: Max(Column_name)

  Example : Select max(sal) from emp;

- ✔ Min Function

  Syntax : Min(Column_name)

  Example : Select min(sal) from emp;

- ✔ Avg Function

  Syntax : Avg(Column_name)

  Example : Select avg(sal) from emp;

S3 SLO-1 : Functions - aggregation functions

- ✔ Sum Function

  Syntax : Sum(Column_name)

  Example : Select sum(sal) from emp;

- ✔ Count Function

  Syntax : Count(Column_name)

  Example : Select count(sal) from emp;

Note : (1) When Count (*) is used , it will count the number of rows using rowid value (Rowid is the unique id and it is a 16bit hexa decimal number for all the rows assigned at the time of creation)

(2) When Count (Column_name) is used, it will count the number of values in a specified column except null values

S3 SLO-1 : Functions - aggregation functions

- ✔ Distinct Function

  Syntax : Distinct (Column_name)

  Example : Select distinct (job) from emp;

- ✔ STDDEV Function

  Syntax : Stddev (Column_name)

  Example : Select stddev(sal) from emp;

- ✔ Variance Function

  Syntax : Variance (Coulmn_name)

  Example : Select variance (sal) from emp;

S3 SLO-1 : Functions - aggregation functions

Some more examples using EMP table

- ✓ Select sum(sal), avg(sal), min(sal), max(sal) from emp;
- ✓ Select Job, sum(sal) from emp group by job;
- ✓ Select Job, count(job) from emp group by job;
- ✓ Select job, avg(sal), max(sal),min(sal) from emp group by job;
- ✓ Select distinct (job) from emp;
- ✓ Select count(*) from emp;
- ✓ Select count(comm) from emp;
- ✓ Select count(sal) from emp;
- ✓ Select stddev(sal) from emp;
- ✓ select variance(sal) from emp;

S3 SLO-2 : Built-in Functions-Numeric, Date, String functions, Set operations

Numeric Functions

Functions	Value Returned	Example
Abs(n)	Absolute value of n	Select abs(-15) from dual;
Ceil(n)	Smallest int >= n	Select ceil(33.645) from dual;
Cos(n)	Cosine of n	Select cos(180) from dual;
Cosh(n)	Hyperbolic cosine of n	Select cosh(0) from dual;
Exp(n)	en	Select exp(2) from dual;
Floor(n)	Largest int <= n	Select floor(100.2) from dual;
Ln(n)	Natural log of n (base e)	Select ln(5) from dual;
Log(b,n)	Log n base b	Select log(2,64) from dual;
Mod(m,n)	Remainder of m divided by n	Select mod(17,3) from dual;

S3 SLO-2 : Built-in Functions-Numeric, Date, String functions, Set operations

Numeric Functions

Functions	Value Returned	Example
Power(m,n)	m power n	Select power(5,3) from dual;
Round(m,n)	m rounded to n decimal places	Select round(125.67854,2) from dual;
Sign(n)	If n<0, -1 if n=0, 0 otherwise 1.	Select sin(-19) from dual;
Sin(n)	Sin of n	Select sin(90) from dual;
Sinh(n)	Hyperbolic sin of n	Select sinh(45) from dual;
Sqrt(n)	Square root of n	Select sqrt(7) from dual;
Tan(n)	Tangent of n	Select tan(45) from dual;
Tanh(n)	Hyperbolic tangent of n	Select tanh(60) from dual;
Trunc(m,n)	m truncated to n decimal places	Select trunc(125.5764,2) from dual;

S3 SLO-2 : Built-in Functions-Numeric, Date, String functions, Set operations

Date Functions

Functions	Value Returned	Example
add_months(d,n)	‘n’ months added to date ‘d’.	Select add_months(sysdate,2) from dual;
last_day(d)	Date corresponding to the last day of the month	Select last_day(sysdate) from dual;
to_date(str,’format’)	Converts the string ina given format into Oracle date.	Select to_date(’10-02-09’,’dd-mm-yy’) from dual;
to_char(date,’format’)	Reformats date according to format	Select to_char(sysdate,’dy dd mon yyyy’) from dual;
months_between(d1,d2)	No. of months between two dates	Select months_between(sysdate, to_date(’10-10-07’,’dd-mm-yy’)) from dual;
next_day(d,day)	Date of the ‘day’ that immediately follows the date ‘d’	Select next_day(sysdate,’wednesday’) from dual;

S3 SLO-2 : Built-in Functions-Numeric, Date, String functions, Set operations

Date Functions

Functions	Value Returned	Input
round(d,'format')	Date will be the rounded to nearest day.	Select round(sysdate,'year') from dual;
		Select round(sysdate,'month') from dual;
		Select round(sysdate,'day') from dual;
		Select round(sysdate) from dual;
trunc(d,'format')	Date will be the truncated to nearest day.	Select trunc(sysdate,'year') from dual;
		Select trunc(sysdate,'month') from dual;
		Select trunc(sysdate,'day') from dual;
		Select trunc(sysdate) from dual;
greatest(d1,d2,…)	Picks latest of list of dates	Select greatest(sysdate, to_date('02-10-06','dd-mm-yy'),to-date('12-07- 12','dd-mm-yy')) from dual;
Date Arithmetic	Add /Subtract no. of days to a date	Select sysdate+25 from dual;
		Select sysdate-25 from dual;
	Subtract one date from another, producing a no. of days	Select sysdate - to_date('02-10-06','dd- mm-yy') from dual;

S3 SLO-2 : Built-in Functions-Numeric, Date, String functions, Set operations

String Functions

Functions	Value Returned	Input
initcap(char)	First letter of each word capitalized	Select initcap('database management') from dual;
lower(char)	Lower case	Select lower('WELCOME') from dual;
upper(char)	Upper case	Select upper('srmist') from dual;
ltrim(char, set)	Initial characters removed up to the character not in set.	Select ltrim('muruganantham','murug') from dual;
rtrim(char, set)	Final characters removed after the last character not in set.	Select rtrim('muruganantham','antham') from dual;
translate(char, from, to)	Translate 'from' by 'to' in char.	Select translate('jack','j','b') from dual;
replace(char, search, repl)	Replace 'search' string by 'repl' string in 'char'.	Select replace('jack and jue','j','bl') from dual;
substr(char, m, n)	Substring of 'char' at 'm' of size 'n' char long.	Select substr('muruganantham',7,6) from dual;

S3 SLO-2 : Built-in Functions-Numeric, Date, String functions, Set operations

Set Operators

- ✔ Set operators are used to join the results of two (or more) SELECT statement.
- ✔ The following set operators are available in SQL
  - Union
  - Union All
  - Intersect
  - Minus

S3 SLO-2 : Built-in Functions-Numeric, Date, String functions, Set operations

Set Operators

- ✔ Union operator retrieves the records from both queries without duplicate.
- ✔ Coulmn heading will be selected from the prior query statement.
- ✔ Union All retrieves all the records from both queries (with duplicate).
- ✔ Intersect operator retrieve the common records from both query statements.
- ✔ Minus operator retrieve the records from first query , the records are not available in second query.

S3 SLO-2 : Built-in Functions-Numeric, Date, String functions, Set operations

Set Operators

Point to be followed while using SET operators

- ✔ The number of columns must be same in all participating query
- ✔ Column heading will be selected from the first query for displaying the output.
- ✔ Data types of the column list must be match with all the query.
- ✔ Positional ordering must be used to sort the result set.
- ✔ UNION and INTERSECT operators are commutative, i.e. the order of queries is not important; it doesn't change the final result.
- ✔ Set operators can be the part of sub queries.
- ✔ Set operators can't be used in SELECT statements containing TABLE collection expressions.
- ✔ The LONG, BLOB, CLOB, BFILE, VARRAY,or nested table are not permitted for use in Set operators.
- ✔ For update clause is not allowed with the set operators.

S3 SLO-2 : Built-in Functions-Numeric, Date, String functions, Set operations

Examples: Set Operators

✔ Create two tables named a and b with the columns f1,f2 and id,name respectively

```
Sql> create table a (f1 number, f2 varchar2(5))
 Table created.

Sql> create table b (id number, name varchar2(5))
 Table created.
```

S3 SLO-2 : Built-in Functions-Numeric, Date, String functions, Set operations

Examples: Set Operators

- ✔ Insert three rows as given below in tables a and b

```
Sql>insert into a values( 10,'A')
1 row(s) inserted.

Sql>insert into a values( 20,'B')
 row(s) inserted.

Sql>insert into a values( 30,'C')
1 row(s) inserted.

Sql>insert into b values(30,'C')
1 row(s) inserted.

Sql>insert into b values(40,'D')
1 row(s) inserted.

Sql>insert into b values(50,'E')
1 row(s) inserted.
```

S3 SLO-2 : Built-in Functions-Numeric, Date, String functions, Set operations

Examples: Set Operators

Records in table a

```
Sql > Select * from a
F1        F2
30  C
10  A
20  B
```

Records in table b

```
Sql > Select * from b;
ID  NAME

30  C

40  D

50  E
```

S3 SLO-2 : Built-in Functions-Numeric, Date, String functions, Set operations

Examples: Set Operators

Example 1 : Union

```
Sql>    select f1  from a
     union
     select id from b;
          F1
          10
          20
          30
          40
          50
```

Column Heading from first query for display purpose

Example 2 : Union

```
Sql>    select id from b
     union
     select f1  from a;
          ID
          10
          20
          30
          40
          50
```

S3 SLO-2 : Built-in Functions-Numeric, Date, String functions, Set operations

Examples: Set Operators

Example : Union All

```
Sql>select f1 from a union all select id from b
        F1
        30
        10
        20
        30
        40
        50
```

Note : Output with duplication, 30 is common in both tables

S3 SLO-2 : Built-in Functions-Numeric, Date, String functions, Set operations

Examples: Set Operators

Example 1: Intersect

```
Sql>select f1  from a
        intersect
       select id from b;
          F1
          30
```

Example 2: Intersect

```
Sql>select id from b
        intersect
       select f1  from a;
          ID
          30
```

S3 SLO-2 : Built-in Functions-Numeric, Date, String functions, Set operations

Examples: Set Operators

Example 1: Intersect

```
Sql>    select f1 from a
        minus
    select id from b
        F1
        10
        20
```

Example 2 : Intersect

```
Sql>    select id from b
        minus
    select f1 from a
        ID
        40
        50
```

S 4-5 SLO-1 & SLO-2 :

Lab 7 : Join Queries on sample exercise.

Joins

- ✔ SQL Joins are used to fetch records from two are more tables using a common field.
- ✔ To implement join condition minimum two tables are required
- ✔ Join conditions used in where clause as given below

```
<Table_Name1> . <Column_name> = <Table_Name1> . <Column_name>
```

Note : Column name used in where condition need not to same , but data type should be same.

S 4-5 SLO-1 & SLO-2 :
Lab 7 : Join Queries on sample exercise.

✔ The Join conditions are classified as follows

- Simple Join - Equi Join
  - Non Equi Join
- Self Join
- Outer Join - Left Outer Join
  - Right Outer Join

S 4-5 SLO-1 & SLO-2 :

Lab 7 : Join Queries on sample exercise.

Note : Consider EMP and Dept Tables

Simple Join

- ✓ For the below query the output is cartesian prodcut

```
Sql> select * from emp,dept;
```

- ✓ For all the records in the first table (emp) , Each and every record in the second table (dept) will be executed.
- ✓ That is , 14 x 4 Records will be in output.

S 4-5 SLO-1 & SLO-2 :

Lab 7 : Join Queries on sample exercise.

Simple Join

Syntax : where <table_name1>.<column_name> =
<table_name2>.<column_name>

Example 1:

```
Sql> select * from emp,dept where emp.deptno= dept.deptno;
```

Example 2:

```
Sql> select ename,dname from emp,dept where emp.deptno= dept.deptno;
```

S 4-5 SLO-1 & SLO-2 :

Lab 7 : Join Queries on sample exercise.

Simple Join

Example 3: (Simple join and Equi Join)

```
Sql>   select ename,dname from emp,dept
     where emp.deptno= dept.deptno and ename like '%S‘
```

Example 4: (Common column should be specified with tablename as shown in example below)

```
Sql> select ename,dname,dept.deptno from emp,dept where
       emp.deptno= dept.deptno and ename like '%S'
```

S 4-5 SLO-1 & SLO-2 :
Lab 7 : Join Queries on sample exercise.

Simple Join

Example :

Sql> select a.ename,b.ename,a.empno,b.empno from emp a, emp b where b.mgr=a.empno;

Outer Join

Example: (Right Outer Join)

Sql> select * from emp ,dept where emp.deptno = dept.deptno(+);

Example: (Left Outer Join)

Sql> select * from emp ,dept where emp.deptno (+) = dept.deptno;

S 6 SLO-1 & SLO-2 : Sub Queries, correlated sub queries

- ✔ A Query statement contains another query is called sub query or nested query
- ✔ A subquery is used to return value(s) that will be used in the main query as a condition
- ✔ Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.
- ✔ The followings to be considered while using sub query
  - Subqueries must be enclosed within parentheses.
  - An ORDER BY command cannot be used in a subquery
  - Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
  - The BETWEEN operator cannot be used with a subquery, but the BETWEEN operator can be used within the subquery.

S 6 SLO-1 & SLO-2 : Sub Queries, correlated sub queries

Basic Syntax of Subquery

SELECT column_name [, column_name] FROM table1 [, table2] WHERE column_name OPERATOR

(SELECT column_name [, column_name] FROM table1 [, table2]

S 6 SLO-1 & SLO-2 : Sub Queries, correlated sub queries

Note : Consider emp table

Example : Subqueries with the SELECT Statement

```
SQL> SELECT * FROM emp WHERE deptno IN (SELECT deptno FROM dept WHERE dname =
‘SALES’ or dname = ‘RESEARCH’) ;
```

Example: Subqueries with the INSERT Statement

```
SQL> INSERT INTO emp_backup SELECT * FROM emp WHERE deptno IN (SELECT deptno
FROM dept) ;
```

Example: Subqueries with the delete Statement

```
SQL> DELETE FROM emp WHERE deptno IN (SELECT deptno FROM dept WHERE deptno=10 );
```

Example: Subqueries with the update Statement

```
SQL> UPDATE emp SET comm = SAL * 0.25 WHERE deptno IN (SELECT deptno FROM dept
WHERE deptno >30);
```

S 6 SLO-1 & SLO-2 : Sub Queries, correlated sub queries

Sample Sub queries

- ✔ SELECT ename FROM EMP WHERE sal > (SELECT sal FROM emp WHERE empno=7566);
- ✔ SELECT ename, sal, deptno, job FROM EMP WHERE job = SELECT job FROM emp WHERE empno=7369);
- ✔ SELECT ename, sal, deptno FROM EMP WHERE sal IN (SELECT MIN(sal) FROM emp GROUP BY deptno);
- ✔ SELECT empno, ename, job FROM emp WHERE sal < ANY (SELECT sal FROM emp WHERE job = 'CLERK');
- ✔ SELECT empno, ename, job FROM emp WHERE sal > ALL (SELECT AVG(sal) FROM emp GROUP BY deptno) ;
- ✔ SELECT ename, sal, deptno FROM EMP WHERE sal IN (SELECT MIN(sal) FROM emp GROUP BY deptno) ;
- ✔ SELECT job, AVG(sal) FROM emp GROUP BY job HAVING AVG(sal) = (SELECT MIN(AVG(sal)) FROM emp GROUP BY job);

S 6 SLO-1 & SLO-2 : Sub Queries, correlated sub queries

correlated sub queries

- ✔ SQL correlated subquery is a query which is executed one time for each record returned by the outer query.
- ✔ It is called correlated as it is a correlation between the number of times the sub query is executed with the number of records returned by the outer query

Examples for Correlated Sub query

- ✔ List the employees who have never received a comm.

```
SELECT ename,comm  FROM emp e1  WHERE NOT EXISTS (SELECT ename
FROM emp e2 WHERE e2.empno = e1.empno AND e2,comm = null)
```

S 6 SLO-1 & SLO-2 : Sub Queries, correlated sub queries

✔ Using EXISTS the following query display the empno, mgr, ename of those employees who manage other employees.

```
select empno, mgr, ename from emp a where exists
(select empno from emp b where b.mgr = a.empno)
```

✔ Find the nth maximum salry in emp table

```
select * from( select ename, sal, dense_rank() over(order by sal desc) r from Emp)
where r=&n;
```

Note : Dense_rank () function

Example : select ename,sal, dense_rank() over (order by sal desc) Sal_rank from emp

S-7 SLO-1 & SLO-2 : Nested Queries, Views and its Types

Nested Sub query

- A subquery can be nested inside other subqueries.
- SQL has an ability to nest queries within one another. A subquery is a SELECT statement that is nested within another SELECT statement and which return intermediate results.
- SQL executes innermost subquery first, then next level.

Eaxmple :

```
SQL>    SELECT job,AVG(sal),Min(sal),Max(sal) FROM emp GROUP BY job HAVING
  2     AVG(sal)  <  (SELECT MAX(AVG(sal)) FROM emp WHERE job IN
  3    (SELECT job FROM emp WHERE deptno BETWEEN 10 AND 40) GROUP BY job);

        JOB        AVG(SAL)    MIN(SAL)   MAX(SAL)
        ---------        ----------       ----------     ----------
        CLERK          1037.5           800      1300
        SALESMAN             1400             1250       1600
        ANALYST              3000             3000       3000
        MANAGER    2758.33333      2450       2975
```

S-7 SLO-1 & SLO-2 : Nested Queries, Views and its Types

Difference between Nested Subquery and Correlated subquery :

Nested Query

- ✔ In Nested Query, Inner query runs first, and only once. Outer query is executed with result from Inner query. Hence, Inner query is used in execution of Outer query.
- ✔ Example : Consider EMP and Dept Tables

```
SQL> select deptno from dept where deptno not in ( select deptno from emp );

DEPTNO
----------
    40
```

Correlated Query

- ✔ In Correlated Query, Outer query executes first and for every Outer query row Inner query is executed. Hence, Inner query uses values from Outer query.
- ✔ Example : Consider EMP,Dept Tables

```
SQL>  Select dname from dept where deptno not in ( select deptno from emp );

DNAME
--------------
OPERATIONS
```

S-7 SLO-1 & SLO-2 : Nested Queries, Views and its Types

Difference between Nested Query, Correlated Query and Join Operation

Parameters	Nested Query	Correlated Query	Join Operation
Definition	In Nested query, a query is written inside another query and the result of inner query is used in execution of outer query.	In Correlated query, a query is nested inside another query and inner query uses values from outer query.	Join operation is used to combine data or rows from two or more tables based on a common field between them. INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN are different types of Joins.
Approach	Bottom up approach i.e. Inner query runs first, and only once. Outer query is executed with result from Inner query.	Top to Down Approach i.e. Outer query executes first and for every Outer query row Inner query is executed.	It is basically cross product satisfying a condition.

S-7 SLO-1 & SLO-2 : Nested Queries, Views and its Types

Difference between Nested Query, Correlated Query and Join Operation

Parameters	Nested Query	Correlated Query	Join Operation
Dependency	Inner query execution is not dependent on Outer query.	Inner query is dependent on Outer query.	There is no Inner Query or Outer Query. Hence, no dependency is there.
Performance	Performs better than Correlated Query but is slower than Join Operation.	Performs slower than both Nested Query and Join operations as for every outer query inner query is executed.	By using joins we maximize the calculation burden on the database but joins are better optimized by the server so the retrieval time of the query using joins will almost always be faster than that of a subquery.

S-7 SLO-1 & SLO-2 : Nested Queries, Views and its Types

Views

- ✔ Views are defined using view updation rule set by Edger.F Codds
- ✔ View updation rule is not fully satisfied till.
- ✔ View is a Virtual table
- ✔ Virtual table (view) based on the result of query statement.
- ✔ Views can be created from single table
- ✔ Views can be created from single table using selected columns
- ✔ Views can be created from single table using selected rows
- ✔ Views can be created from multiple tables
- ✔ Views can be created from multiple tables using selected columns
- ✔ Views can be created from multiple tables using selected rows
- ✔ Data manipulation is possible in views with restrictions
- ✔ The changes made in the base table(s) will be reflected in view(s)
- ✔ The changes made in the views will reflected in base table with restrictions

S-7 SLO-1 & SLO-2 : Nested Queries, Views and its Types

VIEWS

General Syntax

```
CREATE VIEW view_name AS SELECT column1, column2, ...
FROM table_name(s) WHERE condition;
```

Example 1: Creating a view from emp table

```
SQL>create view emp_view as select * from emp;
View created

SQL> desc emp_view
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                     NOT NULL NUMBER(4)
 ENAME                                              VARCHAR2(10)
 JOB                                       VARCHAR2(9)
 MGR                                                NUMBER(4)
 HIREDATE                                           DATE
 SAL                                                NUMBER(7,2)
 COMM                                               NUMBER(7,2)
 DEPTNO                                             NUMBER(2)
```

S-7 SLO-1 & SLO-2 : Nested Queries, Views and its Types

Views

Output of Example 1:

```
SQL> select * from emp_view;
 EMPNO ENAME     JOB          MGR HIREDATE       SAL      COMM    DEPTNO
----------   ----------    --------- -        ---------  ---------          ----------     ----------        ----------
     7369 SMITH     CLERK         7902 17-DEC-80      800                        20
     7499 ALLEN      SALESMAN  7698 20-FEB-81     1600       300              30
     7521 WARD       SALESMAN  7698 22-FEB-81     1250       500              30
     7566 JONES     MANAGER   7839 02-APR-81     2975                         20
     7654 MARTIN     SALESMAN 7698 28-SEP-81     1250      1400             30
     7698 BLAKE      MANAGER     7839 01-MAY-81      2850                      30
     7782 CLARK      MANAGER    7839 09-JUN-81      2450                      10
     7788 SCOTT      ANALYST      7566 09-DEC-82      3000                     20
     7839 KING          PRESIDENT            17-NOV-81      5000                    10
     7844 TURNER   SALESMAN 7698 08-SEP-81      1500         0              30
     7876 ADAMS      CLERK          7788 12-JAN-83      1100                      20
     7900 JAMES      CLERK          7698 03-DEC-81       950                       30
     7902 FORD        ANALYST      7566 03-DEC-81      3000                      20
     7934 MILLER     CLERK          7782 23-JAN-82      1300                      10

14 rows selected.
```

S-7 SLO-1 & SLO-2 : Nested Queries, Views and its Types

Views

Example 2: Creating a view from emp table using ename, job and sal columns

```
SQL> create view emp_view1 as select ename, job, sal from emp;

View created.

SQL> select * from emp_view1;

ENAME      JOB              SAL
----------  ---------   ----------
SMITH      CLERK            800
ALLEN      SALESMAN    1600
WARD       SALESMAN    1250
JONES      MANAGER     2975
MARTIN     SALESMAN    1250
BLAKE      MANAGER     2850
CLARK      MANAGER     2450
SCOTT      ANALYST      3000
KING         PRESIDENT   5000
TURNER     SALESMAN   1500
ADAMS      CLERK           1100
JAMES      CLERK          950
FORD       ANALYST       3000
MILLER     CLERK          1300

14 rows selected.
```

Views

Example 3: Creating a view from emp table using selected rows

```
SQL> create view emp_view2 as select * from emp where deptno=10;
View created.

 EMPNO ENAME      JOB          MGR HIREDATE        SAL       COMM     DEPTNO
---------- ---------- ---------         ---------- ---------       ----------     ----------     ----------
      7782 CLARK   MANAGER      7839 09-JUN-81       2450                            10
      7839 KING      PRESIDENT            17-NOV-81       5000                            10
      7934 MILLER  CLERK           7782 23-JAN-82       1300                            10
```

S-7 SLO-1 & SLO-2 : Nested Queries, Views and its Types

Views

Example 4: Creating a view from emp table using selected rows

```
SQL> create view emp_view3 as select ename,job from emp where sal>2000;
View created.

SQL> select * from emp_view3;

ENAME      JOB
----------  ---------
JONES     MANAGER
BLAKE      MANAGER
CLARK      MANAGER
SCOTT     ANALYST
KING       PRESIDENT
FORD      ANALYST

6 rows selected.
```

S-7 SLO-1 & SLO-2 : Nested Queries, Views and its Types

Views

Example 4: Creating a view from emp and dept table

```
SQL> create view emp_dept as select ename,dname from emp, dept where emp.deptno=dept.deptno;
View created.

ENAME      DNAME
---------- --------------
SMITH      RESEARCH
ALLEN      SALES
WARD       SALES
JONES      RESEARCH
MARTIN     SALES
BLAKE      SALES
CLARK      ACCOUNTING
SCOTT      RESEARCH
KING       ACCOUNTING
TURNER     SALES
ADAMS      RESEARCH
JAMES      SALES
FORD       RESEARCH
MILLER     ACCOUNTING

14 rows selected.
```

S-7 SLO-1 & SLO-2 : Nested Queries, Views and its Types

Data manipulation in Views

- ✔ If view has been created from single table using all the fields the data manipulation is possible in view and table.
  - Whatever the changes made in table , that will reflected in view and vice versa.
- ✔ If view has been created from single table using selected fields including NOT NULL and PRIMARY KEY columns , data manipulation is possible.
  - Whatever the changes made in table , that will reflected in view and vice versa.
- ✔ If view has been created from single table using selected fields excluding NOT NULL and PRIMARY key fields, data manipulation is having some restrictions
  - Insertion is not possible
  - Updation and Deletion is possible
- ✔ If view has been created from multiple tables , data manipulation is not possible.

S-7 SLO-1 & SLO-2 : Nested Queries, Views and its Types

Data manipulation in Views

Click the following link for examples

- ✔ C:\Users\Admin\Desktop\Academic Year 2021_2022 EVEN Semester\18CSC303J DBMS\Example for Data Manipulation in View.docx

S-8 SLO-1 : Transaction Control Commands

TCL Commands

✔Used to give / get back / control the privileges of an object by the owner

GRANT : To give access privileges of an object to other user by the owner

Syntax : GRANT [ALL / INSERT /UPDATE /DELETE /SELECT]

on <OBJECT_NAME> to <USER_NAME>;

Example: GRANT all on emp to scott;

REVOKE : To get back all the privileges from the user who has been granted

Syntax : REVOKE [ALL / INSERT /UPDATE /DELETE /SELECT]

on <OBJECT_NAME> from <USER_NAME>;

Example: REVOKE all on emp from scott;

S-8 SLO-2 : Commit, Rollback, Savepoint

TCL Commands

- ✔ To control the database operation
  - COMMIT: Commits a Transaction. Save the changes permanently , can’t rollback
  - ROLLBACK: Rollbacks a transaction in case of any error occurs.
  - SAVEPOINT: Sets a savepoint within a transaction. Rolled back from the specified savepoint

S-9-10 SLO-1 & SLO-2 : Lab 8: Set Operators & Views

Set Operators

- ✔ Refer slide numbers : 50 to 59

Views

- ✔ Refer slide nubmbers : 76 to 83
- ✔ Click the following link for examples
- ✔ C:\Users\Admin\Desktop\Academic Year 2021_2022 EVEN Semester\18CSC303J DBMS\Example for Data Manipulation in View.docx

S-11 SLO-1 & SLO-2 : PL/SQL Concepts- Cursors

PL/SQL concepts

- ✔ PL/SQL is an extension of SQL.
- ✔ It is a procedural language , where SQL is a non procedural language.
- ✔ The PL/SQL programming language was developed by Oracle Corporation in the late 1980s as procedural extension language for SQL and the Oracle relational database.
- ✔ Block of SQL statements can be executed using PL/SQL
- ✔ PL/SQL is a completely portable, high-performance transaction-processing language.
- ✔ PL/SQL provides a built-in, interpreted and OS independent programming environment.
- ✔ There are four type of PL/SQL blocks
- ✔ Anonymous Block
- ✔ Named Block
- ✔ Sub Programs (Procedures, Functions and Packages)
- ✔ Triggers

S-11 SLO-1 & SLO-2 : PL/SQL Concepts- Cursors

Features of PL/SQL

- ✓ Tightly integrated with SQL.
- ✓ Extensive error checking.
- ✓ Several data types.
- ✓ Variety of programming structures.
- ✓ Structured programming through functions and procedures.
- ✓ Supports object-oriented programming.
- ✓ Easy development of web applications and server pages.

S-11 SLO-1 & SLO-2 : PL/SQL Concepts- Cursors

Structure of PL/SQL block

```
DECLARE
   <declarations section>
BEGIN
   <executable statements>
EXCEPTION
   <exception handling>
END;
```

S-11 SLO-1 & SLO-2 : PL/SQL Concepts- Cursors

Structure of PL/SQL block

Declarations

- ✔ This section starts with the keyword DECLARE.
- ✔ It is an optional section
- ✔ Defines all variables, cursors, subprograms, and other elements to be used in the program.
- ✔ Each and every variables to be declared individually.

Executable Statements

- ✔ Program execution starts from BEGIN
- ✔ Between BEGIN and END is called BODY of PL/SQL block.
- ✔ Nested BODY is permitted
- ✔ It consists of executable statements.

Exception Handling

- ✔ This is an optional section , starts with Exception
- ✔ It is used to handle the logical errors during run time.

S-11 SLO-1 & SLO-2 : PL/SQL Concepts- Cursors

Simple Example

```
DECLARE
    message varchar2(100):= ‘Welcome to SRMIST';
BEGIN
    dbms_output.put_line(message);
END;
```

Output :

```
Welcome to SRMIST
PL/SQL procedure successfully completed
```

S-11 SLO-1 & SLO-2 : PL/SQL Concepts- Cursors

Comments in PL/SQL

- ✔ Double hypen (--) is Single line comment
- ✔ Multiline comments enclosed by /* and */

Example

```
DECLARE
-- variable declaration
message varchar2(20):= ‘Welcome to SRMIST ';
BEGIN
/* PL/SQL executable statement(s)
    This Program display a Welcome note*/
dbms_output.put_line(message);
END;
```

Single line comment

Multi-line Comment

S-11   SLO-1 & SLO-2 : PL/SQL Concepts- Cursors

Data types in PL/SQL

All data types used in SQL can be used in PL/SQL

- ✔ Numeric - Numeric values on which arithmetic operations are performed.
- ✔ Character - Alphanumeric values that represent single characters or strings of characters.
- ✔ Boolean - Logical values on which logical operations are performed.
- ✔ Datetime - Dates and times.

S-11 SLO-1 & SLO-2 : PL/SQL Concepts- Cursors

Variable Declaration in PL/SQL

variable_name [CONSTANT] datatype [NOT NULL] [:= | DEFAULT initial_value]

Examples

```
salary number(10, 2);
pi  number(10,2) := 3.1415;
ename varchar2(25);
address varchar2(100);
```

S-11 SLO-1 & SLO-2 : PL/SQL Concepts- Cursors

Simple Example for variable declarations

```
DECLARE
   a number := 10;
   b number := 20;
   c number (10,4);
   f float;
BEGIN
   c := a + b;
   dbms_output.put_line('Value of c: ' || c);
   f := 100.0/3.0;
   dbms_output.put_line('Value of f: ' || round(f,4));
END;
```

Output

```
Value of c: 30
Value of f: 33.3333

PL/SQL procedure successfully completed.
```

S-11 SLO-1 & SLO-2 : PL/SQL Concepts- Cursors

Assigning SQL query result to PL/SQL variables using INTO clause
(Consider EMP table)

```
DECLARE
       emp_no emp.empno%type:=&emp_no;
       emp_name  emp.ename%type;
       emp_job emp.job%type;
       emp_sal emp.sal%type;
    BEGIN
       SELECT ename,job,sal INTO emp_name,emp_job,emp_sal
       FROM emp WHERE empno = emp_no;
      dbms_output.put_line
      ('Employee ' ||emp_name || ' working as ' || emp_job || ' and his salary is ' || emp_sal);
 END;
```

Output

```
Enter value for emp_no: 7499
old     2:        emp_no emp.empno%type:=&emp_no;
new    2:        emp_no emp.empno%type:=7499;

Employee ALLEN working as SALESMAN his salary is 1600

PL/SQL procedure successfully completed.
```

S-11 SLO-1 & SLO-2 : PL/SQL Concepts- Cursors

PL/SQL Operators

- ✓ Arithmetic operators
- ✓ Relational operators
- ✓ Comparison operators
- ✓ Logical operators
- ✓ String operators

S-11 SLO-1 & SLO-2 : PL/SQL Concepts- Cursors

Conditional Statement in PL/SQL

- ✔ IF - THEN statement
- ✔ IF-THEN-ELSE statement
- ✔ IF-THEN-ELSIF statement
- ✔ Case statement
- ✔ Nested IF-THEN-ELSE

Loop Statement in PL/SQL

- ✔ Basic Loop statement
- ✔ While Loop statement
- ✔ For loop statement
- ✔ Nested Loop statement

Loop Control Statement

- ✔ Exit
- ✔ Continue
- ✔ Goto

S-11 SLO-1 & SLO-2 : PL/SQL Concepts- Cursors

Cursors

- ✔ Cursor is a private SQL workgroup area allocated temporarily
- ✔ The required amount of memory space will be allocated in cursor name
- ✔ A cursor holds the records written by select statement
- ✔ There are two types of cursors
  - Implicit Cursors
  - Explicit Cursors

S-11 SLO-1 & SLO-2 : PL/SQL Concepts- Cursors

Implicit Cursors

- ✔ Oracle create implicit cursor automatically whenever the DML statements (INSERT, UPDATE and DELETE) are executed.
- ✔ The implicit cursors are SQL cursors
- ✔ The SQL cursors has four attributes

Attribute	Description
SQL%ISOPEN	Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
SQL%FOUND	Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.
SQL%NOTFOUND	The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
SQL%ROWCOUNT	Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

S-11 SLO-1 & SLO-2 : PL/SQL Concepts- Cursors

Example for Implicit Cursors (Consider EMP table)

```
DECLARE
   total_rows number(2);
  BEGIN
    UPDATE emp SET sal =  sal + 500 where comm is null ;
    IF sql%notfound THEN
      dbms_output.put_line('No Employee selected');
    ELSIF sql%found THEN
      total_rows := sql%rowcount;
      dbms_output.put_line( total_rows || ' Employees selected ');
   END IF;
 END;
```

Output

```
10 Employees selected

PL/SQL procedure successfully completed.
```

Note : Sal updated in EMP table for 10 employees , those commission is null

S-11 SLO-1 & SLO-2 : PL/SQL Concepts- Cursors

Explicit Cursors

- ✓ Explicit cursors are user-defined cursors
- ✓ It should be defined in the declaration section of the PL/SQL Block.

Syntax

```
CURSOR cursor_name IS select_statement;
```

The following steps to be followed for explicit cursors

- ✓ Declare the cursor for initialize the memory
- ✓ Open the cursor for allocating memory
- ✓ Fetch the cursor values into local variables
- ✓ Close the cursor for release the memory

S-11 SLO-1 & SLO-2 : PL/SQL Concepts- Cursors

Example for Explicit Cursors

```
DECLARE
emp_no emp.empno%type;
emp_name emp.ename%type;
emp_sal emp.sal%type;
CURSOR emp_cur is SELECT empno,ename,sal FROM emp;
BEGIN
OPEN emp_cur;
dbms_output.put_line(‘emp_no’ || ' ' || ‘emp_name’ || ' ' ||’emp_sal’);
LOOP
FETCH emp_cur into emp_no,emp_name,emp_sal;
EXIT WHEN emp_cur%notfound;
dbms_output.put_line(emp_no  || ' ' || emp_name || ' ' || emp_sal);
END LOOP;
CLOSE emp_cur;
END;
```

```
Output

emp_no  emp_name      emp_sal
7369    SMITH              800
7499    ALLEN            1600
7521    WARD             1250
7566    JONES            2975
7654    MARTIN            1250
7698    BLAKE             2850
7782    CLARK             2450
7788    SCOTT            3000
7839    KING             5000
7844    TURNER            1500
7876    ADAMS             1100
7900    JAMES             950
7902    FORD             3000
7934    MILLER            1300
PL/SQL procedure successfully completed.
```

S-11 SLO-1 & SLO-2 : PL/SQL Concepts- Cursors

Cursor based Records

```
DECLARE
CURSOR emp_currec is SELECT empno, ename FROM emp;
emp_rec emp_currec%rowtype;
BEGIN
OPEN emp_currec;
DBMS_OUTPUT.put_line(‘Employee Number’ || ' ' || ‘Name’);
LOOP
FETCH emp_currec into emp_rec;
 EXIT WHEN emp_currec%notfound;
DBMS_OUTPUT.put_line(emp_rec.empno || ' ' || emp_rec.ename);
END LOOP;
END;
```

```
Output
Employee Number Name
7369            SMITH
7499            ALLEN
7521            WARD
7566             JONES
7654            MARTIN
7698            BLAKE
7782            CLARK
7788             SCOTT
7839            KING
7844            TURNER
7876             ADAMS
7900            JAMES
7902            FORD
7934            MILLER

PL/SQL procedure successfully completed.
```

S-12 SLO-1 & SLO-2 : Stored Procedure, Functions Triggers and Exceptional Handling

Sub Program

- ✔ A program which can be called and executed in another program.
- ✔ Sub program will user identification and program identification (procedure / function name).
- ✔ The program which is calling the sub program is known as calling program or main program.
- ✔ In PL/SQL there are two types of sub programs.
  - Procedures
  - Functions
- ✔ The collection of procedure(s) and Function(s) is a PACKAGE.

Procedures

- ✓ Procedures do not return directly
- ✓ Used to perform a specific task
- ✓ General Syntax for Procedure

```
CREATE [OR REPLACE] PROCEDURE procedure_name [(parameter_name [IN
   | OUT | IN OUT] type [, ...])] {IS | AS}
BEGIN
< procedure_body >
END procedure_name;
```

S-12 SLO-1 & SLO-2 : Stored Procedure, Functions Triggers and Exceptional Handling

Procedures

In the given syntax

- ✔ procedure-name specifies the name of the procedure.
- ✔ [OR REPLACE] option allows the modification of an existing procedure.
- ✔ The optional parameter list contains name, mode and types of the parameters.
  - IN parameter lets you pass a value to the subprogram. It is a read-only parameter.
  - OUT parameter returns a value to the calling program. Inside the subprogram, an OUT parameter acts like a variable.
  - IN OUT parameter passes an initial value to a subprogram and returns an updated value to the caller.
- ✔ procedure-body contains the executable part.
- ✔ The AS keyword is used instead of the IS keyword for creating a standalone procedure.

S-12 SLO-1 & SLO-2 : Stored Procedure, Functions Triggers and Exceptional Handling

Example: Welcome note

```
CREATE OR REPLACE PROCEDURE welcome
AS
BEGIN
   dbms_output.put_line
      ('Welcome to SRMIST');
END;
```

Output

```
SQL> exec welcome; -- to run the procedure
Welcome to SRMIST
PL/SQL procedure successfully completed.
```

The procedure can be executed as given below also

```
SQL> begin
  2  welcome;
  3  end;
  4  /
Welcome to SRMIST

PL/SQL procedure successfully completed.
```

S-12 SLO-1 & SLO-2 : Stored Procedure, Functions Triggers and Exceptional Handling

Example : Parameters

```
DECLARE
   a number:=&a;
   b number:=&b;
   c number;
PROCEDURE findMin
 (x IN number, y IN number, z OUT number) IS
BEGIN
   IF x < y THEN
      z:= x;
   ELSE
      z:= y;
   END IF;
END;
BEGIN
   findMin(a, b, c);
   dbms_output.put_line
   (' Minimum of '|| a ||'  and  ' ||b || ' is :'  || c);
END;
```

Output

```
Enter value for a: 100
old   2:     a number:=&a;
New 2:     a number:=100;
Enter value for b: 200
old   3:     b number:=&b;
New  3:     b number:=200;
Minimum of 100  and  200 is :100

PL/SQL procedure successfully completed.
```

Example : Parameters

```
DECLARE
a number := &a;
PROCEDURE square (x IN OUT number) IS
BEGIN
x := x * x;
END;
BEGIN
square(a);
dbms_output.put_line (' Square is : ' || a);
END;
```

Output

```
Enter value for a: 10
old   2:  a number := &a;
new   2:  a number := 10;
Square is : 100

PL/SQL procedure successfully completed.
```

S-12 SLO-1 & SLO-2 : Stored Procedure, Functions Triggers and Exceptional Handling

Example : Display ename from emp table

```
create or replace PROCEDURE get_ename IS
emp_name       VARCHAR2(10);
CURSOR         c1 IS SELECT ename FROM emp;
BEGIN
        OPEN c1;
        LOOP
            FETCH c1 INTO emp_name;
            EXIT WHEN c1%NOTFOUND;
            DBMS_OUTPUT.PUT_LINE(emp_name);
       END LOOP;
       CLOSE c1;
 END get_ename;
```

Output

```
SQL> exec get_ename;
SMITH
ALLEN
WARD
JONES
MARTIN
BLAKE
CLARK
SCOTT
KING
TURNER
ADAMS
JAMES
FORD
MILLER

PL/SQL procedure successfully completed.
```

S-12 SLO-1 & SLO-2 : Stored Procedure, Functions Triggers and Exceptional Handling

Example : Display employee record

```
CREATE OR REPLACE PROCEDURE get_emp_rec (emp_number  IN emp.empno%TYPE) AS
emp_ret emp%ROWTYPE;
BEGIN
        SELECT empno, ename, job, mgr, hiredate, sal, comm, deptno
         INTO emp_ret
         FROM emp
         WHERE empno = emp_number;
         DBMS_OUTPUT.PUT_LINE(emp_ret.empno||'   '||emp_ret.ename||'   '||
         emp_ret.job||'   '||emp_ret.sal);
END;
```

Output

```
SQL> exec get_emp_rec (7499);
7499   ALLEN   SALESMAN   1600

PL/SQL procedure successfully completed.
```

S-12 SLO-1 & SLO-2 : Stored Procedure, Functions Triggers and Exceptional Handling

Functions

- ✓ Function is like procedure , but it should return a value and it returns only one value
- ✓ General syntax for Function creation

```
CREATE [OR REPLACE] FUNCTION function_name [(parameter_name
[IN | OUT | IN OUT] type [, ...])] RETURN return_datatype {IS | AS}
BEGIN
< function_body >
END [function_name];
```

S-12 SLO-1 & SLO-2 : Stored Procedure, Functions Triggers and Exceptional Handling

Functions

In the given syntax

- ✔ function-name specifies the name of the function.
- ✔ [OR REPLACE] option allows the modification of an existing function.
- ✔ The optional parameter list contains name, mode and types of the parameters.
  - IN parameter lets you pass a value to the subprogram. It is a read-only parameter.
  - OUT parameter returns a value to the calling program. Inside the subprogram, an OUT parameter acts like a variable.
  - IN OUT parameter passes an initial value to a subprogram and returns an updated value to the caller.
- ✔ The function must contain a return statement.
- ✔ The RETURN clause specifies the data type you are going to return from the function.
- ✔ function-body contains the executable part.
- ✔ The AS keyword is used instead of the IS keyword for creating a standalone function.

S-12 SLO-1 & SLO-2 : Stored Procedure, Functions Triggers and Exceptional Handling

Example : To count the number of employees in emp table

Creation of Function:

```
CREATE OR REPLACE FUNCTION
total_employee RETURN number IS
total number(2) := 0;
BEGIN
SELECT count(*) into total FROM emp;
RETURN total;
END;
```

Calling the Function

```
 DECLARE
      n number(2);
 BEGIN
       n := total_employee();
       dbms_output.put_line('Total no. of Employees: '
|| n);
 END;
```

Output

```
Total no. of Employees: 14

PL/SQL procedure successfully completed.
```

S-12 SLO-1 & SLO-2 : Stored Procedure, Functions Triggers and Exceptional Handling

Example : Function to find the maximum of given two numbers

```
DECLARE
   a number := &a;
   b number := &b;
   c number;
FUNCTION findMax(x IN number, y IN number)
RETURN number IS
    z number;
BEGIN
   IF x > y THEN
      z:= x;
   ELSE
      z:= y;
   END IF;
   RETURN z;
END;
BEGIN
   c := findMax(a, b);
   dbms_output.put_line(' Maximum of '||a||'  and '||b|| ' is: ' || c);
END;
```

Output

```
Enter value for a: 100
old  2:     a number := &a;
New 2:     a number := 100;
Enter value for b: 200
old   3:     b number := &b;
New 3:     b number := 200;
Maximum of 100  and 200 is: 200

PL/SQL procedure successfully completed.
```

S-12 SLO-1 & SLO-2 : Stored Procedure, Functions Triggers and Exceptional Handling

Recursive Function in PL/SQL : Find the Factorial of a number

```
DECLARE
   num number;
   factorial number;
FUNCTION fact(x number) RETURN number IS
   f number;
BEGIN
   IF x=0 THEN
      f := 1;
   ELSE
      f := x * fact(x-1);
   END IF;
RETURN f;
END;
BEGIN
   num:= &num;
   factorial := fact(num);
   dbms_output.put_line(' Factorial '|| num || ' is ' || factorial);
END;
```

```
Output
Enter value for num: 5
old  17:    num:= &num;
new  17:    num:= 5;
Factorial 5 is 120

PL/SQL procedure successfully completed.
```

Triggers

- ✔ Triggers are event driven program
- ✔ Not necessary to execute manually
- ✔ Triggers are automatically executed when the event ocuurs
- ✔ There are 12 events are there in PL/SQL
  - Before insert
  - Before delete
  - Before update
  - After insert
  - After delete
  - After update
- ✔ The above mentioned events can be executed for each row or each statements
- ✔ Apart from these triggers, Instead of triggers are also available which is used to insert into the views which is created from more than one tables

S-12 SLO-1 & SLO-2 : Stored Procedure, Functions Triggers and Exceptional Handling

Syntax for Trigger

```
CREATE [OR REPLACE ] TRIGGER trigger_name {BEFORE | AFTER | INSTEAD OF } {INSERT
[OR] | UPDATE [OR] | DELETE} [OF col_name] ON table_name [REFERENCING OLD AS o NEW
AS n] [FOR EACH ROW] WHEN (condition)
DECLARE
<Declaration-statements>
BEGIN
<Executable-statements>
EXCEPTION
<Exception-handling-statements>
END;
```

S-12 SLO-1 & SLO-2 : Stored Procedure, Functions Triggers and Exceptional Handling

In the given syntax

- ✔ CREATE [OR REPLACE] TRIGGER trigger_name − Creates or replaces an existing trigger with the *trigger_name*.
- ✔ {BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
- ✔ {INSERT [OR] | UPDATE [OR] | DELETE} − This specifies the DML operation.
- ✔ [OF col_name] − This specifies the column name that will be updated.
- ✔ [ON table_name] − This specifies the name of the table associated with the trigger.
- ✔ [REFERENCING OLD AS o NEW AS n] − This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
- ✔ [FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- ✔ WHEN (condition) − This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

S-12 SLO-1 & SLO-2 : Stored Procedure, Functions Triggers and Exceptional Handling

Example : Display the salary difference in emp table

```
create or replace trigger display_salary_changes
before delete or insert or update on emp
for each row
when (new.empno > 0)
declare
    sal_diff number;
begin
    sal_diff := :new.sal  - :old.sal;
    dbms_output.put_line('old salary: ' || :old.sal);
    dbms_output.put_line('new salary: ' || :new.sal);
    dbms_output.put_line('salary difference: ' || sal_diff);
end;
```

```
Output
SQL> update emp set sal = 1111 where empno = 7900;
Old salary: 950
New salary: 1111
Salary difference: 161

1 row updated.

SQL> insert into emp values
    (1234,'NANTHA','MANAGER',7839,'23-MAR-83',4000,NULL,
    30);
Old salary:
New salary: 4000
Salary difference:

1 row created.

SQL> delete from emp where empno=1234;

1 row deleted.
```

S-12 SLO-1 & SLO-2 : Stored Procedure, Functions Triggers and Exceptional Handling

Example for Triggers

Voters_list and polling

Click the following link

C:\Users\Admin\Desktop\Academic Year 2021_2022 EVEN Semester\18CSC303J DBMS\BM_DBMS_PPT_18CSC303J\Triggers Example.docx

S-12 SLO-1 & SLO-2 : Stored Procedure, Functions Triggers and Exceptional Handling

Exception Handling

- ✔ Exception handling is an error handling mechanism
- ✔ It will handle the logical errors during the runtime
- ✔ While the occurrence of logical errors , exceptions are used to continue the program execution instead stop the execution.
- ✔ There are two types of exceptions are in PL/SQL
    - System defined Exception
        - No_data_found
        - Login_denied
        - Too_many_rows
        - Value_error
        - Zero_divide
    - User defined Exception

S-12 SLO-1 & SLO-2 : Stored Procedure, Functions Triggers and Exceptional Handling

Exception Handling

General Syntax

```
DECLARE
<declarations section>
BEGIN
<executable command(s)>
EXCEPTION
<exception handling goes here >
WHEN exception1 THEN exception1-handling-statements
WHEN exception2 THEN exception2-handling-statements
WHEN exception3 THEN exception3-handling-statements
........
END;
```

S-12 SLO-1 & SLO-2 : Stored Procedure, Functions Triggers and Exceptional Handling

Example : System defined Exception

```
declare
   emp_number   number(10) := &empno;
   emp_name     varchar2(10);
begin
   select ename into emp_name from emp where
    empno = emp_number;
   dbms_output.put_line('employee name is ' ||
    emp_name);
exception
   when no_data_found then
      dbms_output.put_line('no such employee: ' ||
    emp_number);
end;
```

Output

```
Enter value for empno: 7499
old 2: emp_number   number(10) := &empno;
New 2: emp_number   number(10) := 7499;
Employee name is ALLEN

PL/SQL procedure successfully completed.

SQL> /
Enter value for empno: 1234
old  2: emp_number   number(10) := &empno;
New 2: emp_number   number(10) := 1234;
No such employee: 1234

PL/SQL procedure successfully completed.
```

S-12 SLO-1 & SLO-2 : Stored Procedure, Functions Triggers and Exceptional Handling

Example for User defined exception With Raise command

```
DECLARE
emp_name VARCHAR2(10);
emp_number NUMBER;
empno_out_of_range EXCEPTION;

BEGIN
emp_number := &empno;
IF    emp_number > 9999 OR emp_number < 1000 THEN
    RAISE empno_out_of_range;
    ELSE
    SELECT ename INTO emp_name FROM emp WHERE empno = emp_number;
    DBMS_OUTPUT.PUT_LINE('Employee name is ' || emp_name);
END IF;
EXCEPTION
    WHEN empno_out_of_range THEN
    DBMS_OUTPUT.PUT_LINE('Employee number ' || emp_number || ' is out of range.');
END;
```

S-12 SLO-1 & SLO-2 : Stored Procedure, Functions Triggers and Exceptional Handling

Output

```
SQL> /
Enter value for empno: 7499
old   6:     emp_number := &empno;
new   6:     emp_number := 7499;
Employee name is ALLEN

PL/SQL procedure successfully completed.

SQL> /
Enter value for empno: 900
old   6:     emp_number := &empno;
new   6:     emp_number := 900;
Employee number 900 is out of range.

PL/SQL procedure successfully completed.

SQL> /
Enter value for empno: 10000
old   6:     emp_number := &empno;
new   6:     emp_number := 10000;
Employee number 10000 is out of range.

PL/SQL procedure successfully completed.
```

S-12 SLO-1 & SLO-2 : Stored Procedure, Functions Triggers and Exceptional Handling

Example : With user defined and System defined Exceptions

```
DECLARE
   emp_name         VARCHAR2(10);
   emp_number        NUMBER;
   empno_out_of_range EXCEPTION;
BEGIN
   emp_number := &empno;
   IF emp_number > 9999 OR emp_number < 1000 THEN
      RAISE empno_out_of_range;
   ELSE
      SELECT ename INTO emp_name FROM emp
         WHERE empno = emp_number;
      DBMS_OUTPUT.PUT_LINE('Employee name is ' || emp_name);
END IF;
EXCEPTION
   WHEN empno_out_of_range THEN
      DBMS_OUTPUT.PUT_LINE('Employee number ' || emp_number ||
       ' is out of range.');
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE ('Employee number does not exist');
END;
```

```
Output

SQL> /
Enter value for empno: 7499
old   6:     emp_number := &empno;
new   6:     emp_number := 7499;
Employee name is ALLEN

PL/SQL procedure successfully completed.

SQL> /
Enter value for empno: 1234
old   6:     emp_number := &empno;
new   6:     emp_number := 1234;
Employee number does not exist

PL/SQL procedure successfully completed.

SQL> /
Enter value for empno: 500
old   6:     emp_number := &empno;
new   6:     emp_number := 500;
Employee number 500 is out of range.

PL/SQL procedure successfully completed.

SQL> /
Enter value for empno: 20000
old   6:     emp_number := &empno;
new   6:     emp_number := 20000;
Employee number 20000 is out of range.

PL/SQL procedure successfully completed.
```

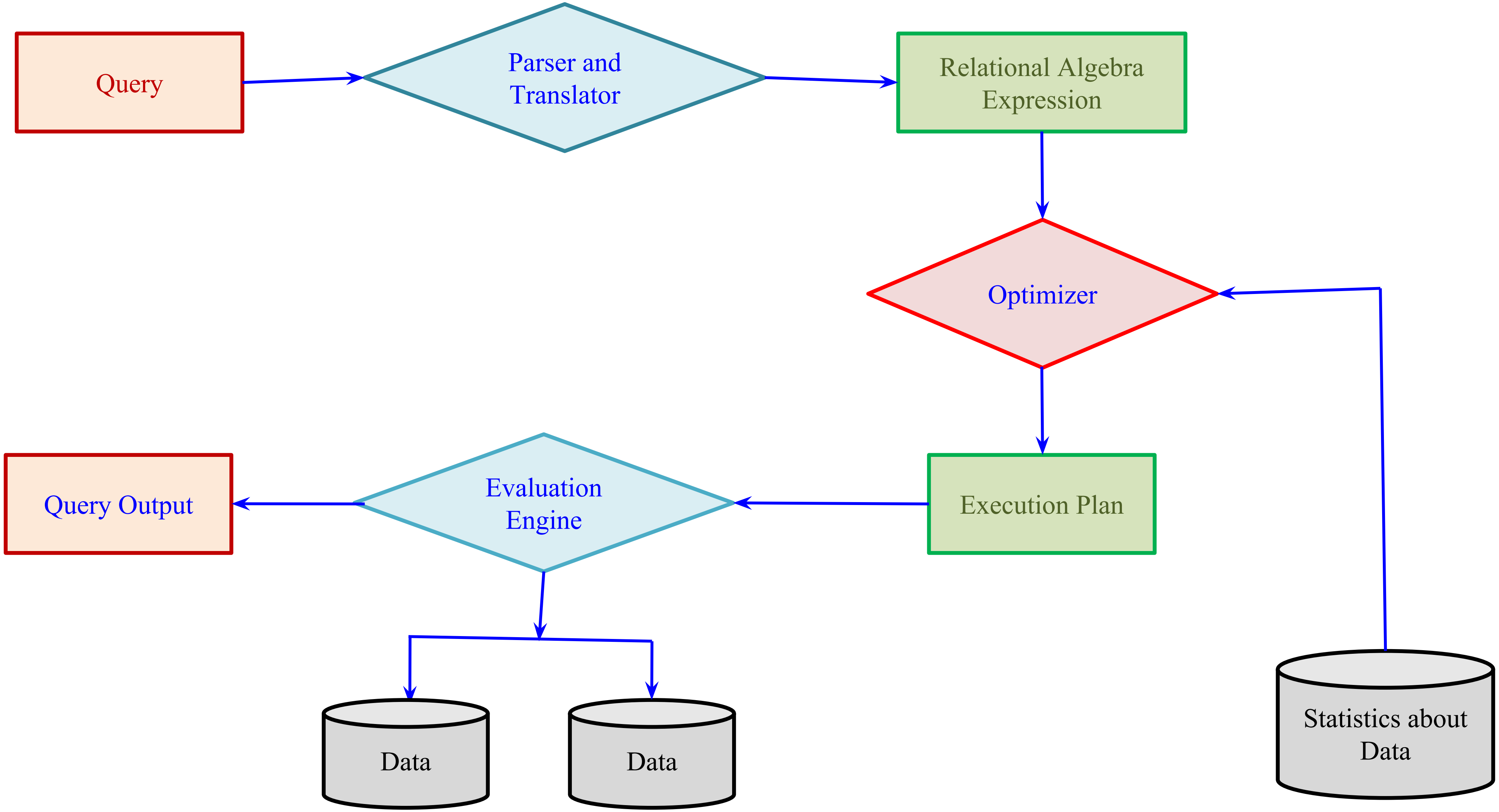
S-13 SLO-1 & SLO-2 : Query Processing

- ✓ Query processing is the process of identifying the fessible solution to execute the query.

- ✓ There are number of ways to execute the query , from that optimal solution will be identified and executed.

- ✓ The basic steps in Query processing are
  - Parsing and translation
  - Optimization
  - Evaluation

S-13 SLO-1 & SLO-2 : Query Processing



S-13 SLO-1 & SLO-2 : Query Processing

Basic Steps in Query Processing

- ✔ Parsing and translation
  - Translate the query into its internal form.
  - This is then translated into relational algebra.
  - Parser checks syntax, verifies relations
- ✔ Evaluation
  - The query-execution engine takes a query-evaluation plan, executes that plan, and returns the answers to the query.

S-13 SLO-1 & SLO-2 : Query Processing

Basic steps in Query Optimization

- ✔ A relational algebra expression may have many equivalent expressions
  - E.g., $\sigma_{salary<75000}(\prod_{salary}(instructor))$ is equivalent to $\prod_{salary}(\sigma_{salary<75000}(instructor))$
- ✔ Each relational algebra operation can be evaluated using one of several different algorithms
  - Correspondingly, a relational-algebra expression can be evaluated in many ways.
- ✔ Annotated expression specifying detailed evaluation strategy is called an evaluation-plan.
  - E.g., can use an index on salary to find instructors with salary < 75000,
  - or can perform complete relation scan and discard instructors with salary ≥ 75000

S-13 SLO-1 & SLO-2 : Query Processing

Basic steps in Query Optimization

- ✔ Query Optimization: Amongst all equivalent evaluation plans choose the one with lowest cost.
  - Cost is estimated using statistical information from the database catalog
    - e.g. number of tuples in each relation, size of tuples, etc.
- ✔ In this chapter we study
  - How to measure query costs
  - Algorithms for evaluating relational algebra operations
  - How to combine algorithms for individual operations in order to evaluate a complete expression

S-13 SLO-1 & SLO-2 : Query Processing

Measures of Query Cost

- ✔ Cost is generally measured as total elapsed time for answering query
  - Many factors contribute to time cost
    - disk accesses, CPU, or even network communication
- ✔ Typically disk access is the predominant cost, and is also relatively easy to estimate. Measured by taking into account
  - Number of seeks * average-seek-cost
  - Number of blocks read * average-block-read-cost
  - Number of blocks written * average-block-write-cost
    - Cost to write a block is greater than cost to read a block
      - data is read back after being written to ensure that the write was successful

S-13 SLO-1 & SLO-2 : Query Processing

Measures of Query Cost

- ✔ For simplicity we just use the number of block transfers from disk and the number of seeks as the cost measures
  - t_T – time to transfer one block
  - t_S – time for one seek
  - Cost for b block transfers plus S seeks

    $b * t_T + S * t_S$
- ✔ We ignore CPU costs for simplicity
  - Real systems do take CPU cost into account
- ✔ We do not include cost to writing output to disk in our cost formulae
- ✔ Several algorithms can reduce disk IO by using extra buffer space
  - Amount of real memory available to buffer depends on other concurrent queries and OS processes, known only during execution
    - We often use worst case estimates, assuming only the minimum amount of memory needed for the operation is available
- ✔ Required data may be buffer resident already, avoiding disk I/O
  - But hard to take into account for cost estimation

S-13 SLO-1 & SLO-2 : Query Processing

Selection Operation

File scan

- ✔ Algorithm A1 (linear search). Scan each file block and test all records to see whether they satisfy the selection condition.
  - Cost estimate = b_r block transfers + 1 seek
    - b_r denotes number of blocks containing records from relation r
  - If selection is on a key attribute, can stop on finding record
    - cost = $(b_r/2)$ block transfers + 1 seek
  - Linear search can be applied regardless of
    - selection condition or
    - ordering of records in the file, or
    - availability of indices
- ✔ Note: binary search generally does not make sense since data is not stored consecutively
  - except when there is an index available,
  - and binary search requires more seeks than index search

S-13 SLO-1 & SLO-2 : Query Processing

Selection using Indices

- ✔ Index scan – search algorithms that use an index
  - selection condition must be on search-key of index.
- ✔ A2 (primary index, equality on key). Retrieve a single record that satisfies the corresponding equality condition
  - $Cost = (h_i + 1) * (t_T + t_S)$
- ✔ A3 (primary index, equality on nonkey) Retrieve multiple records.
  - Records will be on consecutive blocks
    - ▪ Let b = number of blocks containing matching records
  - $Cost = h_i * (t_T + t_S) + t_S + t_T * b$

S-13 SLO-1 & SLO-2 : Query Processing

Selection using Indices

✔ A4 (secondary index, equality on nonkey).

- Retrieve a single record if the search-key is a candidate key
  - $Cost = (h_i + 1) * (t_T + t_S)$
- Retrieve multiple records if search-key is not a candidate key
  - each of *n* matching records may be on a different block
  - Cost = $(h_i + n) * (t_T + t_S)$
    - ○ Can be very expensive!

S-13 SLO-1 & SLO-2 : Query Processing

Selection using Indices

- ✔ Can implement selections of the form $\sigma_{A \le V}(r)$ or $\sigma_{A \ge V}(r)$ by using
  - a linear file scan,
  - or by using indices in the following ways:
- ✔ A5 (primary index, comparison). (Relation is sorted on A)
  - For $\sigma_{A \ge V}(r)$ use index to find first tuple $\ge v$ and scan relation sequentially from there
  - For $\sigma_{A \le V}(r)$ just scan relation sequentially till first tuple $> v$; do not use index
- ✔ A6 (secondary index, comparison).
  - For $\sigma_{A \ge V}(r)$ use index to find first index entry $\ge v$ and scan index sequentially from there, to find pointers to records.
  - For $\sigma_{A \le V}(r)$ just scan leaf pages of index finding pointers to records, till first entry $> v$
  - In either case, retrieve records that are pointed to
    - requires an I/O for each record
    - Linear file scan may be cheaper

S-13 SLO-1 & SLO-2 : Query Processing

Conjunction: $\sigma_{\theta 1} \wedge _{\theta 2} \wedge \ldots _{\theta n}(r)$

- ✔ A7 (conjunctive selection using one index).
  - Select a combination of θ_i and algorithms A1 through A7 that results in the least cost for $\sigma_{\theta i}(r)$.
  - Test other conditions on tuple after fetching it into memory buffer.
- ✔ A8 (conjunctive selection using composite index).
  - Use appropriate composite (multiple-key) index if available.
- ✔ A9 (conjunctive selection by intersection of identifiers).
  - Requires indices with record pointers.
  - Use corresponding index for each condition, and take intersection of all the obtained sets of record pointers.
  - Then fetch records from file
  - If some conditions do not have appropriate indices, apply test in memory.

S-13 SLO-1 & SLO-2 : Query Processing

Disjunction: $\sigma_{\theta 1} \vee {}_{\theta 2} \vee \ldots {}_{\theta n} (r)$.

- ✔ A10 (disjunctive selection by union of identifiers).
  - Applicable if *all* conditions have available indices.
    - Otherwise use linear scan.
  - Use corresponding index for each condition, and take union of all the obtained sets of record pointers.
  - Then fetch records from file
- ✔ Negation: $\sigma_{\neg\theta}(r)$
  - Use linear scan on file
  - If very few records satisfy ¬θ, and an index is applicable to θ
    - Find satisfying records using index and fetch from file