Sql:

update table\_name

set cod\_user =

CASE

WHEN user\_rol = 'student' THEN '622057'

WHEN user\_rol = 'assistant' THEN '2913659'

WHEN user\_rol = 'admin' THEN '6160230'?

END,date = '12082014'

WHERE user\_rol IN ('student','assistant','admin')

AND cod\_office = '17389551';

--

INSERT INTO table\_users (cod\_user, date, user\_rol, cod\_office)

VALUES

('622057', '12082014', 'student', '17389551'),

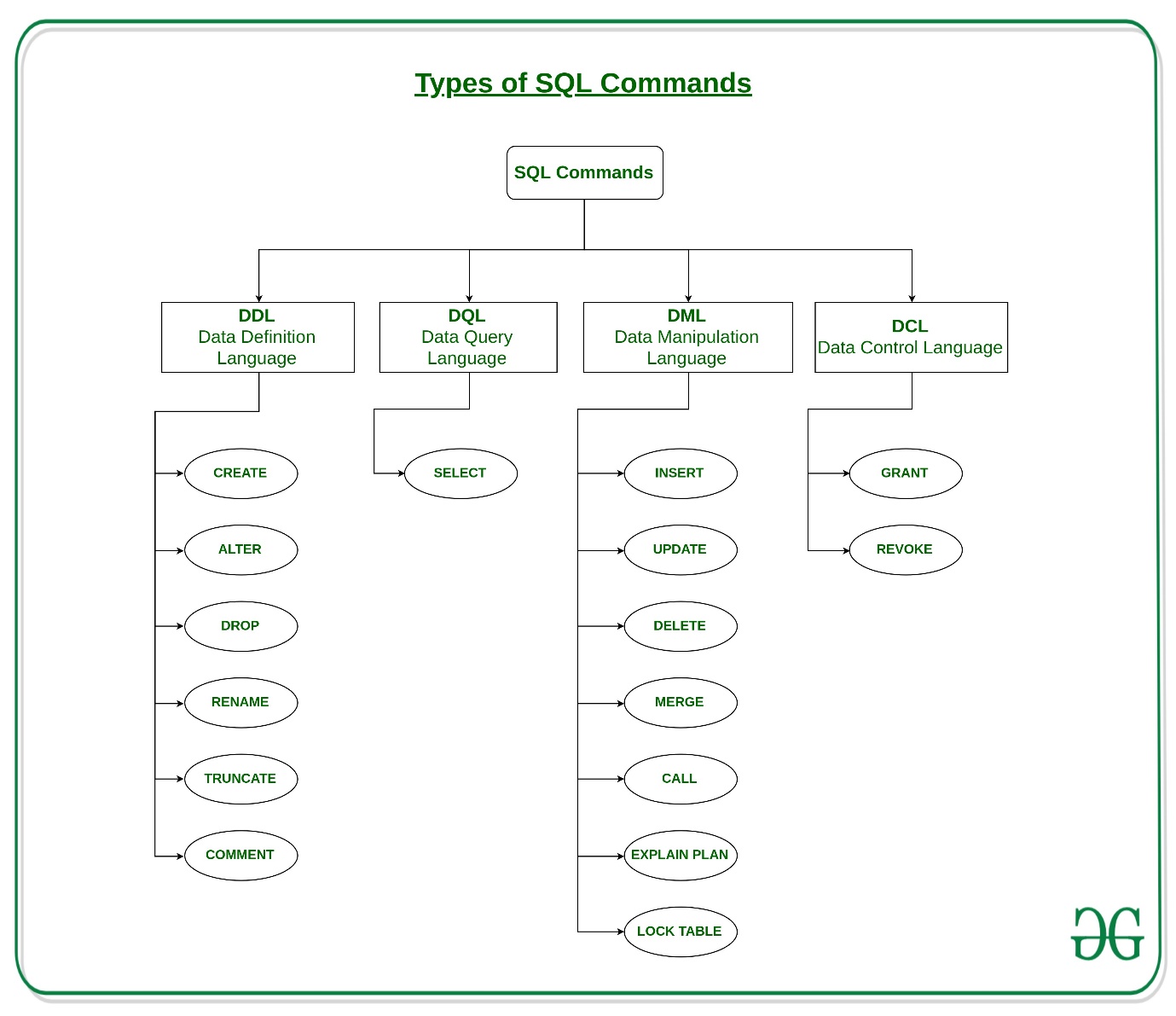
('2913659', '12082014', 'assistant','17389551'),

('6160230', '12082014', 'admin', '17389551')

ON DUPLICATE KEY UPDATE

cod\_user=VALUES(cod\_user), date=VALUES(date)

--



SELECT \* INTO backUpTable from GFG\_Employees;

--

CREATE or REPLACE PROCEDURE INC\_SAL(eno IN NUMBER, up\_sal OUT NUMBER)

IS

BEGIN

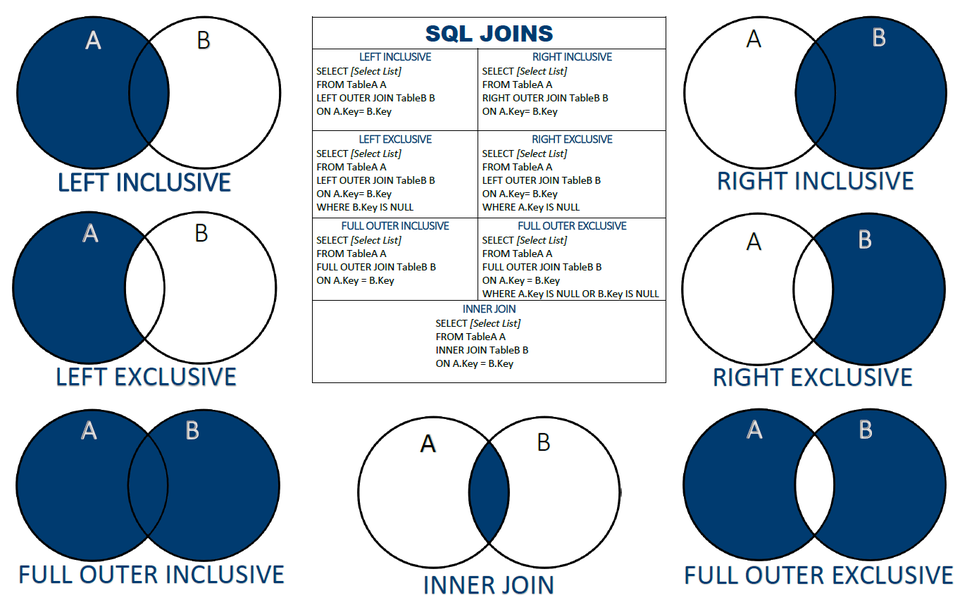
UPDATE emp\_table SET salary = salary+1000 WHERE emp\_no = eno;

COMMIT;

SELECT sal INTO up\_sal FROM emp\_table WHERE emp\_no = eno;

END;

--



INSERT INTO sales.promotions (

promotion\_name,

discount,

start\_date,

expired\_date

)

VALUES

(

'2019 Summer Promotion',

0.15,

'20190601',

'20190901'

),

(

'2019 Fall Promotion',

0.20,

'20191001',

'20191101'

),

(

'2019 Winter Promotion',

0.25,

'20191201',

'20200101'

);

CREATE TABLE employee(emp\_id varchar(5) NOT NULL,

emp\_name varchar(20) NULL,

dt\_of\_join date NULL,

emp\_supv varchar(5) NULL,

CONSTRAINT emp\_id PRIMARY KEY(emp\_id) ,

CONSTRAINT emp\_supv FOREIGN KEY(emp\_supv)

REFERENCESemployee(emp\_id));

Self join;

SELECT a.column\_name, b.column\_name...

FROM table1 a, table1 b

WHERE a.common\_filed = b.common\_field;

--

SELECT a.emp\_id AS "Emp\_ID",a.emp\_name AS "Employee Name",

b.emp\_id AS "Supervisor ID",b.emp\_name AS "Supervisor Name"

FROM employee a, employee b

WHERE a.emp\_supv = b.emp\_id;

--

The SQL UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

Each SELECT statement within UNION must have the same number of columns

The columns must also have similar data types

The columns in each SELECT statement must also be in the same order

--

SELECT 'Customer' As Type, ContactName, City, Country

FROM Customers

UNION

SELECT 'Supplier', ContactName, City, Country

FROM Suppliers;

--

SELECT City, Country FROM Customers

WHERE Country='Germany'

UNION ALL

SELECT City, Country FROM Suppliers

WHERE Country='Germany'

ORDER BY City;

--

View:

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

--

CREATE VIEW view\_name AS

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

--

CREATE VIEW [Products Above Average Price] AS

SELECT ProductName, Price

FROM Products

WHERE Price > (SELECT AVG(Price) FROM Products);

--

Updating a view

CREATE OR REPLACE VIEW [Brazil Customers] AS

SELECT CustomerName, ContactName, City

FROM Customers

WHERE Country = "Brazil";

--

Subquery:

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

There are a few rules that subqueries must follow −

Subqueries must be enclosed within parentheses.

A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.

An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.

Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.

The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.

A subquery cannot be immediately enclosed in a set function.

The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

--

SELECT column\_name [, column\_name ]

FROM table1 [, table2 ]

WHERE column\_name OPERATOR

(SELECT column\_name [, column\_name ]

FROM table1 [, table2 ]

[WHERE])

--

SQL> SELECT \*

FROM CUSTOMERS

WHERE ID IN (SELECT ID

FROM CUSTOMERS

WHERE SALARY > 4500) ;

--

Subquery with insert:

INSERT INTO table\_name [ (column1 [, column2 ]) ]

SELECT [ \*|column1 [, column2 ]

FROM table1 [, table2 ]

[ WHERE VALUE OPERATOR ]

INSERT INTO CUSTOMERS\_BKP

SELECT \* FROM CUSTOMERS

WHERE ID IN (SELECT ID

FROM CUSTOMERS) ;

Subquery with update:

UPDATE table

SET column\_name = new\_value

[ WHERE OPERATOR [ VALUE ]

(SELECT COLUMN\_NAME

FROM TABLE\_NAME)

[ WHERE) ]

UPDATE CUSTOMERS

SET SALARY = SALARY \* 0.25

WHERE AGE IN (SELECT AGE FROM CUSTOMERS\_BKP

WHERE AGE >= 27 );

Subquery with delete:

DELETE FROM TABLE\_NAME

[ WHERE OPERATOR [ VALUE ]

(SELECT COLUMN\_NAME

FROM TABLE\_NAME)

[ WHERE) ]

--

DELETE FROM CUSTOMERS

WHERE AGE IN (SELECT AGE FROM CUSTOMERS\_BKP

WHERE AGE >= 27 );

--

What is a Stored Procedure?

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

Stored Procedure Syntax

CREATE PROCEDURE procedure\_name

AS

sql\_statement

GO;

--

CREATE PROCEDURE SelectAllCustomers

AS

SELECT \* FROM Customers

GO;

EXEC SelectAllCustomers;

REATE PROCEDURE SelectAllCustomers @City nvarchar(30), @PostalCode nvarchar(10)

AS

SELECT \* FROM Customers WHERE City = @City AND PostalCode = @PostalCode

GO;

--

EXEC SelectAllCustomers @City = "London", @PostalCode = "WA1 1DP";

The SQL IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

SELECT \* FROM Customers

WHERE Country IN ('Germany', 'France', 'UK');

--

Sql function

--

CREATE [OR REPLACE] FUNCTION function\_name

[(parameter\_name [IN | OUT | IN OUT] type [, ...])]

RETURN return\_datatype

{IS | AS}

BEGIN

< function\_body >

END [function\_name];

--

Where,

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 represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.

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.

--

CREATE OR REPLACE FUNCTION totalCustomers

RETURN number IS

total number(2) := 0;

BEGIN

SELECT count(\*) into total

FROM customers;

RETURN total;

END;

--

Calling a function:

DECLARE

c number(2);

BEGIN

c := totalCustomers();

dbms\_output.put\_line('Total no. of Customers: ' || c);

END;

--

Example

The following example demonstrates Declaring, Defining, and Invoking a Simple PL/SQL Function that computes and returns the maximum of two values.

DECLARE

a number;

b number;

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

a:= 23;

b:= 45;

c := findMax(a, b);

dbms\_output.put\_line(' Maximum of (23,45): ' || c);

END;

/

When the above code is executed at the SQL prompt, it produces the following result −

Maximum of (23,45): 45

PL/SQL procedure successfully completed.

--

CHECK Constraint in table scope:

ALTER TABLE employees

ADD CONSTRAINT check\_last\_name

CHECK (last\_name IN ('Smith', 'Anderson', 'Jones'));

--

create or replace trigger school\_biu

before insert or update

on school

for each row

begin

if :new.id is null then

:new.id := to\_number(sys\_guid(), 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX');

end if;

end school\_biu;

--

Alter table school modify (postal\_code number);

Trigger:

SQL Trigger | Student Database

Trigger: A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

Syntax:

create trigger [trigger\_name]

[before | after]

{insert | update | delete}

on [table\_name]

[for each row]

[trigger\_body]

--

create trigger [trigger\_name]: Creates or replaces an existing trigger with the trigger\_name.

[before | after]: This specifies when the trigger will be executed.

{insert | update | delete}: This specifies the DML operation.

on [table\_name]: This specifies the name of the table associated with the trigger.

[for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.

[trigger\_body]: This provides the operation to be performed as trigger is fired

BEFORE and AFTER of Trigger:

BEFORE triggers run the trigger action before the triggering statement is run.

AFTER triggers run the trigger action after the triggering statement is run.

create trigger stud\_marks

before INSERT

on

Student

for each row

set Student.total = Student.subj1 + Student.subj2 + Student.subj3, Student.per = Student.total \* 60 / 100;

GITHUB (Version control system)

1. Working directory b) Staging area – pre-commit holding area, c) Commit – GIT repository (History)
2. Remote Repository (GITHUB), Master branch (where committing changes)

Version control is the management of changes to the documents, programs, large websites and other collection of information.

1. CVCS (Centralaised version control system) – uses centralized server. Users access and commit from client machines.
2. DVCS (Distributed version control system) – GIT – Central repository -> <- local repository push and pull. Commit and update to and from local repository.

GIT is developed by Linus Torwalds (Linux Guy)

https://gist-github.com

Debian based linux install command: sudo apt-get update, sudo apt-get upgrade, sudo apt-get install git

Windows: git-scm.com // in windows, type gid bash to open the git prompt

Git version: 2.25.1 // git –version // git config -- global user.name ‘Jaison Jacob’ // user.name ‘email-id’

Git config –list (to list the configuration) // git help <command>

To initialize an empty directory as a git repository, inside the directory, enter ‘git init’. ‘ it creates a ‘.git’ inside.

‘ls –la’ – to display the hidden files // git status // git add <file name> // git commit –m ‘message’

‘git add ‘.’ (no apostrophe realtime) adds everything in the directory

‘git log’ to view the logs. // git log –author=<user\_name>

Workflow: **work, add, commit**

git config --global core.editor 'subl -n -w' to change editor to sublime.

git config --global core.editor 'path to notepad++' to change editor to notepa++. // add notepad++ path to environment variable

Git config –global –list (lists all the config files)

Notepad++ .bash\_profile // alias npp=’notepad++.exe –multIst –nosession

Git diff (to see the differences between files)

Git diff –staged (to see the difference between repository and ‘staged’ files)

Github is a code hosting platform. It’s a repository hosting service. It’s a central repository.

Github.com // “git remote add origin ‘github repolink’” // git pull // git push //

Git pull origin master // git push origin master // --allow-unrelated-histories

Git branch <branch name> // git checkout <branch name> // git merge <branch>

Git reset HEAD <file name> = to revert to to working area from staging area.

Git reset HEAD~ (tilde symbol) = will revert from commit to stage area.

Clone a repository into a new directory. **git clone**, **git fetch** and **git pull**, but not **git push**, will also accept a suitable bundle file

Git checkout .. <file name> = to revert the changes made to the file

Git rm <filename> = to delete a file // git reset HEAD <file name> = to reset.

Git mv <filne name> < new filename> =to rename the file

To ignore unwanted files, create a file as ‘.gitignore’ and add unwanted file names into that.

Git rbase and git merge are same. // git stash is used to save uncommit changes.

Git tags are references that point to specific points in GIT history. Tagging is generally used to capture a point in history that is used for a marked version release.

Git tag <tag name>