sqlSum.sql

```sql

--✨Le langage de définition des données (LDD)
    CREATE
    ALTER
    DROP
--✨Le langage de manipulation des données (LMD)
    SELECT
    INSERT
    UPDATE
    DELETE
--✨Le langage de contrôle des données (LCD)
    GRANT
    REVOKE
    COMMIT
    ROLLBACK

--✨COMMENT IN SQL --------------------[]:
    ONE LINE
    --this is a comment

    MULTIPLE LINE
    /*
    multiple
    line
    comment :
    */

--✨ create ------------------------------------[]
    CREATE TABLE table_name (
        column1 datatype1,
        column2 datatype2,
        ...
    );
--✨CREATE INDEX:
    CREATE INDEX index_name
    ON table_name (column1, column2, ...);

--✨CREATE VIEW:
    CREATE VIEW view_name AS
    SELECT column1, column2, ...
    FROM table_name
    WHERE condition;
    [WITH CHECK OPTION] -- check where condition in any LMD operation

    --*exmaple :
        CREATE VIEW high_salary_employees AS
        SELECT employee_id, first_name, last_name
        FROM employees
        WHERE salary > 50000;


--  ✨check option --------------------------------------------------[]
    /*
    When you attempt to insert, update, or delete rows through a view
    created with WITH CHECK OPTION, Oracle checks whether the
    new or modified data satisfies the conditions specified in the view.
    */
    --*Example:
    CREATE VIEW high_salary_employees AS
    SELECT employee_id, first_name, last_name, salary
    FROM employees
    WHERE salary > 50000
    WITH CHECK OPTION;


--✨modes-----------------------------------------[]
    [<mode>::=[ON DELETE {CASCADE|SET DEFAULT|SET NULL}]
    | [ON UPDATE {CASCADE| SET DEFAULT| SET NULL} -- RESTRICT default
    -- *example :
        CREATE TABLE parent_table (
            parent_id INT PRIMARY KEY
        );
        CREATE TABLE child_table (
            child_id INT PRIMARY KEY,
            parent_id INT,
            FOREIGN KEY (parent_id)
                REFERENCES parent_table(parent_id)
                ON DELETE CASCADE
                ON UPDATE SET NULL
        );
     /*
     In this example, the ON DELETE CASCADE specifies that when a row
     in parent_table is deleted, all corresponding rows
     in child_table should also be deleted. The ON UPDATE
     SET NULL specifies that if the parent_id in parent_table is updated,
     the corresponding parent_id in child_table should be set to NULL.

     These clauses are essential for maintaining
     referential integrity in a relational database,
     ensuring that relationships between tables are consistent and valid.
     */

--  ✨Select from table:
    select attr from tableName;

--✨constraints :----------------------------------------[]
    [ CONSTRAINT <nom de la contrainte> ]
    [ NOT NULL |
    UNIQUE |
    PRIMARY KEY |
    CHECK (condition) |
    REFERENCES <nom de la table> (colonne)
    ]
    -- or:
    [ CONSTRAINT <nom de la contrainte>
    [
    UNIQUE (liste de colonnes) |
    PRIMARY KEY (liste de colonnes) |
    CHECK (condition) |
    FOREIGN KEY (liste de colonnes)
    REFERENCES <nom de la table> (liste colonnes) [<mode>]
    ]
    ]

-- ✨show all constraint -----------------------------------------[]
    SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE, TABLE_NAME
    FROM USER_CONSTRAINTS;

---✨Drop ---------[]:
    -- Drop the 'employee_view' view
        DROP VIEW employee_view;

    -- Drop the 'idx_salary' index
        DROP INDEX idx_salary;

    -- Drop the 'employees' table
        DROP TABLE employees;



--✨insert ---------------------------------------[]
    INSERT INTO table_name (column1, column2, ...)
    VALUES (value1, value2, ...);
    -- Inserting a single row
        INSERT INTO employees (employee_id, first_name, last_name, salary)
        VALUES (1, 'John', 'Doe', 50000);

    -- Inserting multiple rows (not compatible with oracle )
        INSERT INTO employees (employee_id, first_name, last_name, salary)
        VALUES (2, 'Jane', 'Smith', 60000),
            (3, 'Bob', 'Johnson', 55000);

--✨update -----------------------------------------------[]
    UPDATE table_name
    SET column1 = value1, column2 = value2, ...
    WHERE condition;

    -- Updating a single column for specific rows
        UPDATE employees
        SET salary = 55000
        WHERE department_id = 10;

    -- Updating multiple columns for a specific row
        UPDATE employees
        SET salary = 60000, job_id = 'MANAGER'
        WHERE employee_id = 1;


--✨delete -----------------------------------------------------[]
    DELETE FROM table_name
    WHERE condition;

--✨alter  main syntax :   ----------------------------------------------[]
    /*
        RESTRICT: pas de destruction si l’objet est référencé ou utilisé ailleurs
        CASCADE: propage la destruction
    */
    ALTER TABLE <nom de la Table>
    {
    ADD COLUMN <def Colonne> |
    DROP COLUMN <nom Colonne> [RESTRICT|CASCADE] |
    ADD CONSTRAINT <def Contrainte> |
    DROP CONSTRAINT <nom Contrainte> [RESTRICT|CASCADE] |
    }


--✨table operations --------------------------------------------------[]
    -- Deleting specific rows based on a condition
        DELETE FROM employees
        WHERE department_id = 20;

    -- Deleting all rows from a table
        DELETE FROM employees;

    -- Adding a new column
        ALTER TABLE table_name
        ADD column_name datatype;

    -- Adding a new column with a default value
        ALTER TABLE table_name
        ADD column_name datatype DEFAULT default_value;

    -- Adding multiple columns
        ALTER TABLE table_name
        ADD (column1 datatype, column2 datatype, ...);

    -- Modifying the datatype of a column
        ALTER TABLE table_name
        MODIFY column_name new_datatype;

    -- Modifying the size of a VARCHAR2 column
        ALTER TABLE table_name
        MODIFY column_name VARCHAR2(new_size);

    -- Renaming a column
        ALTER TABLE table_name
        RENAME COLUMN old_column_name TO new_column_name;

    -- Adding or modifying a default value
        ALTER TABLE table_name
        MODIFY column_name DEFAULT new_default_value;

    -- Dropping a default value
        ALTER TABLE table_name
        MODIFY column_name DEFAULT NULL;

    -- Dropping a single column
        ALTER TABLE table_name
        DROP COLUMN column_name;

    -- Dropping multiple columns
```

```
        ALTER TABLE table_name
        DROP (column1, column2, ...);

    -- Adding a primary key constraint
        ALTER TABLE table_name
        ADD CONSTRAINT pk_constraint_name PRIMARY KEY (column1, column2, ...);

    -- Adding a unique constraint
        ALTER TABLE table_name
        ADD CONSTRAINT unique_constraint_name UNIQUE (column1, column2, ...);

    -- Adding a foreign key constraint
        ALTER TABLE child_table
        ADD CONSTRAINT fk_constraint_name
        FOREIGN KEY (column_name) REFERENCES parent_table (referenced_column);

    -- Dropping a constraint
        ALTER TABLE table_name
        DROP CONSTRAINT constraint_name;

    -- Renaming a table
        ALTER TABLE old_table_name
        RENAME TO new_table_name;

    -- Truncating a table (removing all rows)
        TRUNCATE TABLE table_name;

    -- Adding comments to a table
        COMMENT ON TABLE table_name
        IS 'This is a comment on the table.';



--✨create domain : ---------------------------------------------------[]
    CREATE DOMAIN <nom domaine> <type> [valeur]
    [CONSTRAINT nom_contrainte CHECK (condition) ]

    --example :
        CREATE DOMAIN TypeNomDOC IS VARCHAR2(20);
        CREATE DOMAIN DATE_RDV IS DATE
        DEFAULT (CURRENT_DATE)
        CHECK (VALUE >= CURRENT_DATE)
        NOT NULL

    -- example 2 :
        CREATE DOMAIN email_domain AS VARCHAR(255)
        CHECK (VALUE LIKE '%@%' AND VALUE LIKE '%.%');

        CREATE TABLE users (
        user_id INT PRIMARY KEY,
        username VARCHAR(50) NOT NULL,
        email email_domain NOT NULL
        );

--✨ full syntax select ------------------------------------[]
    SELECT column, group_fonction
    FROM tables
    [ WHERE condition ]
    [ GROUP BY group_by_expression
    [ HAVING group_condition ] ]
    [ ORDER BY column];

--✨select statement ----------------[]:
    specific COLUMNS :
    select atr1,atr2 .... from tableName;

--✨all COLUMNS :
    select * from TableName;

--✨relational operators :
    =
    <> or !=
    >
    <
    >=
    <=

--✨logical operators :
    and
    or
    not

--✨mathimatical operations  -----------[]:
    SELECT EMPLOYEE_ID,FIRST_NAME,LAST_NAME ,SALARY + 100
    from EMPLOYEES;

--✨null value ---------------[]:
    null value is a value that is unavallable unssigned ,unknown
    or inapplicable
    null is not the same as zero or a blank space
    null operand Value = null ;

--✨column alias -------------------------[]
    SELECT clm1  as aliasName, clm2 aliasName   from TableName;
    accepted character with as => $ #  _


--  ✨concatenation  OPERATOR : ---------------------[-]:
    --you can use it to cocatinate between multiple Column in the same time :
    example :
    select first_name||' '||last_name "full name " from EMPLOYEES;

--  ✨q keyword :----------------------------------[-]
    -- exmaple 1:
        select first_name || ' work in departement ' || department_id
        from  employees;
    -- example 2:
        select first_name || q'[ work in departement ]' || department_id
        from  employees;

--✨distinct keyworrd --------------------------------[-]
    to ignore the repetition;
    -- example:
        select distinct first_name || q'[ work in departement ]' || department_id
        from  employees;

--✨describe -----------------------------------[-]
    describe a table  columns  data type
    example: you can use describe or desc

   describe employees;
    -----------------result----------------------------
    Name    Null?   Type
    EMPLOYEE_ID NOT NULL    NUMBER(6)
    FIRST_NAME      VARCHAR2(20)
    LAST_NAME   NOT NULL    VARCHAR2(25)
    EMAIL   NOT NULL    VARCHAR2(25)
    PHONE_NUMBER        VARCHAR2(20)
    HIRE_DATE   NOT NULL    DATE
    JOB_ID  NOT NULL    VARCHAR2(10)
    SALARY      NUMBER(8,2)
    COMMISSION_PCT      NUMBER(2,2)
    MANAGER_ID      NUMBER(6)
    DEPARTMENT_ID       NUMBER(4)
    ------------------------------------------------------
/*these you should know when using
 ✨the where character strings and date values are enclosed with single marks
 ✨character values are case-sensitive and date values are format-sensitive
 ✨the defualt data display format is DD-MON-RR
 ✨ the alis doesn't work directly with where close
*/

-- ✨where statement-----------------------------[-]
   --syntax :
    select * from Columns  where condition;

    -- example:
        select *
        from employees
        where department_id=90;
    -- example 2:
        select *
        from employees
        where department_id=90 and first_name='Steven';
    -- example 3:
        select *
        from employees
        where hire_date='17-oct-03';

-- ✨special comparison operators:------------------------[-]
    between ..and ... ,
    in(set)  ,
    like ,
    is null ,
    is not null,

    -- example 1:
        select *
```

```sql
        from employees
        where salary  between 10000 and 20000;
    -- example 2:
        select *
        from employees
        --his start with a character in this range Adel Basma...
        where first_name  between 'A' and 'C';
    -- example 3:
        select *
        from employees
        where salary in(10000,17000,20000);
    -- exmaple 4:
        select *
        from employees
        where first_name like 'S%'; --start wiht S
    -- example 5:
        select *
        from employees
        where first_name like '%s'; --finish with s;
    -- eample 6:
        select *
        from employees
        where first_name like '%am%'; --include am
    -- example 7:
        select *
        from employees
        where first_name like '_d%'; -- has d in second letter
    -- example 8:
        select *
        from employees
        where first_name like '__s%'; --has s in the third letter:

    -- example 9:
    -- special case when the data contain '_' or '%':
        /*
        if we hava a name that contain special character like
        _ or % we need to espace these ones to  prevent  oracle
        from consdering '_' and '%' as  keywords of sql
        */
        select *
        from employees
        where first_name like '/__s%' escape '/';
    -- example 10:
        select *
        from employees
        where commission_pct is null;
    -- example 11:
        select *
        from employees
        where commission_pct is not  null;
 --✨compare lists  :
    /*
    ✨IN : la condition est vraie si EXP appartient à la liste des valeurs retournées
            par la sous-requête
    ✨ANY : la condition est vraie si la comparaison est vraie pour AU MOINS une
            des valeurs retournées par la sous-requête
```

~~~
    ✨ALL : la condition est vraie si la comparaison est vraie pour TOUTES
            les valeurs retournées par la sous-requête
    ✨EXISTS (sous-requête)
        FAUX si Resultat(Sous-requête) = ∅
        VRAIE si Resultat(Sous-requête) ≠ ∅
    */

    -- ### 1. `IN` Operator (Appartenance):
    /*
        The `IN` operator is used to determine whether a specified
        value matches any value in a subquery or a list.
    */
        -- **Example:**
            ```sql
            SELECT column1, column2
            FROM table
            WHERE column1 IN (value1, value2, ...);
            ~~~

    -- ### 2. `ALL` Operator (À Tous):
    /*
        The `ALL` operator compares a value to all values in a set or
        returned by a subquery.
    */
        -- **Example:**
            ```sql
            SELECT column1, column2
            FROM table
            WHERE column1 > ALL (SELECT other_column FROM another_table);
            ```

    -- ### 3. `ANY` Operator (Au Moins Un):
        /*
            The `ANY` operator compares a value to any value in a set or
            returned by a subquery.
        */
        -- **Example:**
        ```sql
        SELECT column1, column2
        FROM table
        WHERE column1 > ANY (SELECT other_column FROM another_table);
        ```

    -- ### 4. `EXISTS` Operator (Non Vide):
        /*
            The `EXISTS` operator is used to test for the existence of rows
            returned by a subquery.
        */
        -- **Example:**
        ```sql
        SELECT column1, column2
        FROM table
        WHERE EXISTS (SELECT 1 FROM another_table WHERE condition);
        ```
        In these examples:
        /*
```
```

```
        - **Subquery:** A subquery is a query nested within another query.
        It can be used with `IN`, `ALL`, `ANY`, or `EXISTS` operators.

        - **List:** A list of values can be used with the `IN` operator.

        - **Comparison:** The `ALL` and `ANY`
        operators are often used in comparison expressions.

        These operators provide flexibility and efficiency in constructing
        complex queries to filter and compare data in various ways.
        */


-- ✨order by ----------------[]
 /*
     ✨asc  : ascending order,default
     ✨desc :descending order
     by default null come last in ascending order :
     by default null come first in descending order;
     you can change by adding : nulls first(or last)
 */
    syntaxe :
    order by ColumnName orderByWhat(asc desc)
    -- example 1:
        select *
        from employees
        order by hire_date;

    -- example 2:
        select *
        from employees
        order by hire_date desc;

    --example 3: --usngin aliases in ordering :
        select salary as n,first_name
        from employees
        order by   n;

    --example 4: --sort by expression
        select salary +100 as n,first_name
        from employees
        order by   n;

    --example 5:-- sort by column not slected
        select first_name,salary
        from employees
        order by   department_id; -- but this is not logical

    --example 6: --sort using multiple Columns
        select department_id,first_name,salary
        from employees
        order by  department_id,first_name ;-- you specfiy the first_name order asc desc

    --example 7: --sort by column Number
        select department_id,first_name,salary
        from employees
```

```sql
        order by  3; --3=salary

    -- ✨fetch statement ----------------[]
        --show just the N rows
        fetch first N rows only;
    -- example :
        select employee_id,first_name
        from employees
        order by employee_id
        fetch first 5 rows only;

    --✨show N% rows only:
        fetch first n percent rows only;
    -- example:
        select employee_id,first_name
        from employees
        order by employee_id
        fetch first 50 percent rows only ;
    --show from an offest to specific Number show(from offest to  end: dpend on offset)
        offset start rows fetch next end rows only;
    -- example:
        select employee_id,first_name
        from employees
        order by employee_id
        offset 5 rows fetch next 5 rows only;

-- ✨ties -------------------------------------[-]
  --with ties means add  also  rows with same ordered  value

    --without rows that have the  same ordred value
        select employee_id,first_name,salary
        from employees
        order by salary desc
        fetch  first 2 rows only;
    --result :
        EMPLOYEE_ID,FIRST_NAME,SALARY
            100    ,   Steven ,24000
            101    ,   Neena  ,17000

    --with ties:
        select employee_id,first_name,salary
        from employees
        order by salary desc
        fetch   first 2 rows with ties;
    --result:
        EMPLOYEE_ID,FIRST_NAME,SALARY
        EMPLOYEE_ID,FIRST_NAME,SALARY
            100   ,   Steven ,24000
            101   ,   Neena  ,17000
            102   ,   Lex    ,17000

--✨substitution variables :  -------------------------------------[-]
    /*
        temporarily store values with single-ampersand (&) and double-ampersand (&&)
substitution
        1- & :the variable will discareded after is used
        2- use '' when using varchar
        3- &&  + defining of the variable
        user it to supplement the following:
        where rule_expression_conditions;
        order by clauses
        column expressions
        table names
        entire select statements
    */
    -- example 1:
    /*
        in this code we declare a variable that will contains the value that user
        will enter in the prompt  that will pop-up  on the screen
    */
        select employee_id,first_name
        from employees
        where employee_id=&UserChoiceId;
        -- assum that user entered 100 :
        -- result :
        EMPLOYEE_ID,FIRST_NAME
            100    ,  Steven

    -- example 2: with varchar
        select employee_id,first_name,salary
        from employees
        where first_name='&FristName';
    -- example 3:
        select employee_id,last_name,job_id,&column_name
        from employees
        order by &order_column;


--✨define and undefine ---------------------------------------------[-]
    /*
        use the define command to create ans assign a value to a variable
        use the undefine commant to remove a variable
    */

    -- example1:
        -- the prompt no will pop-up because employee_num defined
        define employee_num=100;
        select employee_id, last_name, salary
        from  employees
        where employee_id=&employee_num;

    -- example 2:
        define employee_num=100;
        undefine employee_num; -- remove the variable then it will pop-up the prompt
        select   employee_id,last_name,salary
        from employees
        where employee_id=&employee_num;

--✨`change the prompt message -----------------------[-]:
    /*
        you can change the prompt as follow
        but it should executed as a script
```

```sql
    */
    syntaxe :
    ACCEPT  Variable_name PROMPT 'Messgae' -- accept =define + prompt change message
    -- this message it will just assoscited with only this variable

    -- example 1:
        ACCEPT User_id PROMPT 'Please enter the user id :';
        select employee_id,first_name,last_name,salary
        from employees
        where employee_id=&User_id;

    -- example 2:
        select first_name,last_name,&&User_column
        from employees
        order by  &User_column;
    -- example 3:
        ACCEPT User_column PROMPT 'Pelase enter the Column  :';
        select first_name,last_name,&User_column
        from employees
        order by  &User_column;

--✨verify -----------------------------------------------[-]
    /*
    Use the command to toggle the display of the substitution variable,
    both before and after sql developer repalces substiution variables with values :
    */

    -- example:
        set verify on  --add the stat before edit getting value from user and after
        select first_name,last_name,&User_column
        from employees
        order by  &User_column;

    -- using set define off =>turn off the prompt
    -- example:
        set define off; -- will  stop the prompt
        select *
        from departments
        where department_name like '%&t%' ;  -- &t it's not will consirted as a variable

--✨character function --------------------------[-]
    /*
        there are 2 types for character functions:

        1- case conversion functions (upper ,lower ,initcap)
          initcap : camle case
          i can use them in select,where,order by,

        2- character manipulation functions
    */
    --✨1- case conversion functions (upper ,lower ,initcap)
        -- example 1:
        select employee_id,first_name,upper(first_name),lower(first_name),
initcap(first_name)
        from employees;
        -- example 2:
```

```
        select employee_id,first_name,upper(first_name),lower(first_name),
initcap(first_name)
        from employees
        where upper(first_name) ='PATRICK';

    -- ✨2- character manipulation functions;

        -- ✨a- contact function
            concat(column1,column2);
            -- example :
                select employee_id,first_name,concat(first_name,last_name)
                from employees;

        --✨b-substr function
            substr(column,start,distance);
            -- if you didn't specify the distance value it will be take to the end (all
string )
            -- if you pass a negative value .then the count start from the end
            -- example :
                select employee_id,
                first_name,
                substr(first_name,1,3),
                substr(first_name,2,4),
                substr(first_name,2),
                substr(first_name,-3)
                from employees

        --✨c-length(Column);
            -- example:
                select first_name,length(first_name)
                from employees;

--✨to_char function ------------------------------------[]
  /*
    value: The value to be converted. It can be a date, timestamp, or number.
    format: The format mask that defines how the value should be converted.
    For dates and timestamps, it specifies the desired date or time format.
    For numbers, it specifies the number format.

    ✔YYYY  :year
    ✔MM    :month
    ✔DD    :day
    ✔Day   :dayName
    ✔hh24  :hours
    ✔MI    :minutes
    ✔SS    :seconds

  */
    TO_CHAR(value, format)
    -- example (select with  year ) :
        select with year
        SELECT *
        FROM your_table
        WHERE TO_CHAR(your_date_column, 'YYYY') = '2023';
    -- convert date to heour :
        to_number(to_char(date,'hh24'));

```

```sql
    -- extract HH:mm::ss
        SELECT
        TO_NUMBER(TO_CHAR(SYSDATE, 'HH24')) AS current_hour,
        TO_NUMBER(TO_CHAR(SYSDATE, 'MI')) AS current_minute,
        TO_NUMBER(TO_CHAR(SYSDATE, 'SS')) AS current_second
        FROM DUAL;

    -- extract YYYY::MM::dd
        SELECT
        TO_CHAR(SYSDATE, 'YYYY') AS current_year,
        TO_CHAR(SYSDATE, 'MM') AS current_month,
        SELECT TO_CHAR(SYSDATE, 'DD') AS current_day FROM DUAL;
        FROM DUAL;

    -- now day name :
        select TO_CHAR(SYSDATE, 'Day') AS day_name
        from dual;


    -- change date format :
        Alter session set nls_date_format='dd/mm/yyyy';

--  group functions : ------------------------------------[]
    --*Count(*|[ DISTINCT|ALL] expr)
    /*
        Le nombre de ligne de expr
    */
    --*Avg( [ DISTINCT | ALL] expr)
    /*
        Valeur moyenne de expr, en ignorant les valeurs
        NULL
    */
    --*Min( [ DISTINCT | ALL] expr)
    /*
        Valeur minimale de expr, en ignorant les valeurs
        NULL
    */
    --*Max( [ DISTINCT | ALL] expr)
    /*
        Valeur maximale de expr, en ignorant les valeurs
        NULL
    */
    --*Sum( [ DISTINCT | ALL] expr)
    /*
        Somme des valeurs de expr, en ignorant les valeurs
        NULL
    */
--✨GROUP BY: ------------------------------------[]
    /*
    The `GROUP BY` and `HAVING` clauses in SQL are used together to perform aggregate
    functions on groups of rows and filter the results based on the grouped data.
     ✔️ - The `GROUP BY` clause is used to group rows that have the same values in specified
            columns into summary rows, like a summary table.
     ✔️ - It is often used with aggregate functions (e.g., `COUNT`, `SUM`, `AVG`, `MAX`,
`MIN`)
            to perform calculations on each group of rows.
```

~~~
    ✔ - The columns listed in the `GROUP BY` clause are the grouping columns, and each
            unique combination of values in these columns forms a group.
    */
    -- *Example:
        ```sql
        SELECT department_id, AVG(salary) as avg_salary
        FROM employees
        GROUP BY department_id;
        ~~~

-- ✨HAVING: ------------------------------------[]
    /*
    ✔ - The `HAVING` clause is used in combination with `GROUP BY` to filter the results based
            on the result of aggregate functions applied to the groups.
    ✔ - It is similar to the `WHERE` clause but is specifically designed for filtering
            results after the grouping has been applied.
    ✔ - It allows you to specify conditions on the results of aggregate functions.
    */
    --*Example:
        ```sql
        SELECT department_id, AVG(salary) as avg_salary
        FROM employees
        GROUP BY department_id
        HAVING AVG(salary) > 50000;
        ```

--✨change column format : --[]
    COLUMN example_column FORMAT A100;

--✨joins : --[]
    /*
        Equijointure (jointure naturelle)
        Requêtes sur plusieurs tables: la jointure
        Autojointure (jointure sur la même table)
        Non-équijointure (jointure par non égalité, théta jointure)
    */
    -- *1. INNER JOIN:
        /*
            The INNER JOIN keyword selects records that have matching values
            in both tables. It returns only the rows where there is
            a match in the specified columns.
        */
        SELECT columns
        FROM table1
        INNER JOIN table2 ON table1.column_name = table2.column_name;

    -- *2. LEFT (OUTER) JOIN:
        /*
            The LEFT JOIN keyword returns all records from the left table
            (table1), and the matched records from the right table (table2).
            The result is NULL from the right side if there is no match
        */
        SELECT columns
```
```

```
        FROM table1
        LEFT OUTER JOIN table2 ON table1.column_name = table2.column_name;
        -- or
            SELECT columns
            FROM table1,table2
            where  table1.column_name = table2.column_name(+);

    -- *3. RIGHT (OUTER) JOIN
    /*
        The RIGHT JOIN keyword returns all records from the right
        table (table2), and the matched records from the left table
        (table1). The result is NULL from the left side when there
        is no match.
    */
        SELECT columns
        FROM table1
        RIGHT OUTER JOIN table2 ON table1.column_name = table2.column_name;
        -- or :
            SELECT columns
            FROM table1,table2
            where  table1.column_name(+) = table2.column_name;

    -- *4. FULL (OUTER) JOIN:
        /*
            The FULL JOIN keyword returns all records when there is a
            match in either the left (table1) or the right (table2)
            table records.
        */
        SELECT columns
        FROM table1
        FULL OUTER JOIN table2 ON table1.column_name = table2.column_name;


    --  *5. SELF JOIN:
        /*
            A self-join is a regular join, but the table is joined with
            itself. This is useful for hierarchical structures or when
            relating records within the same table.
        */
        SELECT columns
        FROM table1 alias1
        JOIN table1 alias2 ON alias1.column_name = alias2.column_name;

--✨Requêtes imbriquées --------------------------------------------------[]
    SELECT colonnes_de_projection
    FROM table
    WHERE expr operator (
    SELECT colonnes_de_projection
    FROM table
    WHERE …..
    );

--✨list all constraints :  --------------------------------------------------[]
    SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE, TABLE_NAME
    FROM USER_CONSTRAINTS;

```

~~~
--✨Operators: --------------------------------------------------[]

    -- *### 1. `UNION` Operator:
        /*
            The `UNION` operator is used to combine the result sets of two or more SELECT
            statements into a single result set. It removes duplicate rows from the
            combined result set.
        */

        -- **Syntax:**
            ```sql
            SELECT column1, column2 FROM table1
            UNION
            SELECT column1, column2 FROM table2;
            ```

    -- *### 2. `UNION ALL` Operator:
        /*
            The `UNION ALL` operator is similar to `UNION`, but it does not remove duplicate
            rows. It combines all rows from the result sets of multiple SELECT statements,
            including duplicates.
        */

        -- **Syntax:**
            ```sql
            SELECT column1, column2 FROM table1
            UNION ALL
            SELECT column1, column2 FROM table2;
            ```

    -- *### 3. `INTERSECT` Operator:
        /*
            The `INTERSECT` operator returns the common rows between the result sets
            of two SELECT statements. It returns only the rows that appear in both result sets.
        */
        -- **Syntax:**
            ```sql
            SELECT column1, column2 FROM table1
            INTERSECT
            SELECT column1, column2 FROM table2;
            ```

    -- *### 4. `MINUS` Operator (or `EXCEPT` in some databases):

        /*
            The `MINUS` operator returns the rows that appear in the result set of
            the first SELECT statement but not in the result set of the second
            SELECT statement. It is often used to find the set difference between
            two result sets.
        */
        -- **Syntax:**
            ```sql
            SELECT column1, column2 FROM table1
            MINUS
            SELECT column1, column2 FROM table2;
~~~

~~~
            ```
        /*
            Remember, for these set operations to work, the number
            of columns and their data types in the corresponding positions in the SELECT
            statements must match.
        */
~~~