

sqlSum.sql

```
1
2 -- ✨ Le langage de définition des données (LDD)
3     CREATE
4     ALTER
5     DROP
6 -- ✨ Le langage de manipulation des données (LMD)
7     SELECT
8     INSERT
9     UPDATE
10    DELETE
11 -- ✨ Le langage de contrôle des données (LCD)
12    GRANT
13    REVOKE
14    COMMIT
15    ROLLBACK
16
17 -- ✨ COMMENT IN SQL -----[]:
18     ONE LINE
19     --this is a comment
20
21     MULTIPLE LINE
22     /*
23     multiple
24     line
25     comment :
26     */
27
28 -- ✨ create -----[]
29     CREATE TABLE table_name (
30         column1 datatype1,
31         column2 datatype2,
32         ...
33     );
34 -- ✨ CREATE INDEX:
35     CREATE INDEX index_name
36     ON table_name (column1, column2, ...);
37
38 -- ✨ CREATE VIEW:
39     CREATE VIEW view_name AS
40     SELECT column1, column2, ...
41     FROM table_name
42     WHERE condition;
43     [WITH CHECK OPTION] -- check where condition in any LMD operation
44
45     --*exmaple :
46         CREATE VIEW high_salary_employees AS
47         SELECT employee_id, first_name, last_name
48         FROM employees
49         WHERE salary > 50000;
50
51
52 -- ✨ check option -----[]
53     /*
```

```

54  When you attempt to insert, update, or delete rows through a view
55  created with WITH CHECK OPTION, Oracle checks whether the
56  new or modified data satisfies the conditions specified in the view.
57  */
58  --*Example:
59  CREATE VIEW high_salary_employees AS
60  SELECT employee_id, first_name, last_name, salary
61  FROM employees
62  WHERE salary > 50000
63  WITH CHECK OPTION;
64
65
66  -- ✨modes-----[ ]
67  [ <mode>::=[ON DELETE {CASCADE|SET DEFAULT|SET NULL}]
68  | [ON UPDATE {CASCADE| SET DEFAULT| SET NULL} -- RESTRICT default
69  -- *example :
70      CREATE TABLE parent_table (
71          parent_id INT PRIMARY KEY
72      );
73      CREATE TABLE child_table (
74          child_id INT PRIMARY KEY,
75          parent_id INT,
76          FOREIGN KEY (parent_id)
77              REFERENCES parent_table(parent_id)
78              ON DELETE CASCADE
79              ON UPDATE SET NULL
80      );
81  /*
82  In this example, the ON DELETE CASCADE specifies that when a row
83  in parent_table is deleted, all corresponding rows
84  in child_table should also be deleted. The ON UPDATE
85  SET NULL specifies that if the parent_id in parent_table is updated,
86  the corresponding parent_id in child_table should be set to NULL.
87
88  These clauses are essential for maintaining
89  referential integrity in a relational database,
90  ensuring that relationships between tables are consistent and valid.
91  */
92
93  -- ✨Select from table:
94  select attr from tableName;
95
96  -- ✨constraints :-----[ ]
97  [ CONSTRAINT <nom de la contrainte> ]
98  [ NOT NULL |
99  UNIQUE |
100  PRIMARY KEY |
101  CHECK (condition) |
102  REFERENCES <nom de la table> (colonne)
103  ]
104  -- or:
105  [ CONSTRAINT <nom de la contrainte>
106  [
107  UNIQUE (liste de colonnes) |
108  PRIMARY KEY (liste de colonnes) |
109  CHECK (condition) |

```

```

110 FOREIGN KEY (liste de colonnes)
111 REFERENCES <nom de la table> (liste colonnes) [<mode>]
112 ]
113 ]
114
115 -- ✨ show all constraint -----[]
116 SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE, TABLE_NAME
117 FROM USER_CONSTRAINTS;
118
119 --- ✨ Drop -----[]:
120 -- Drop the 'employee_view' view
121 DROP VIEW employee_view;
122
123 -- Drop the 'idx_salary' index
124 DROP INDEX idx_salary;
125
126 -- Drop the 'employees' table
127 DROP TABLE employees;
128
129
130
131 -- ✨ insert -----[]
132 INSERT INTO table_name (column1, column2, ...)
133 VALUES (value1, value2, ...);
134 -- Inserting a single row
135 INSERT INTO employees (employee_id, first_name, last_name, salary)
136 VALUES (1, 'John', 'Doe', 50000);
137
138 -- Inserting multiple rows (not compatible with oracle )
139 INSERT INTO employees (employee_id, first_name, last_name, salary)
140 VALUES (2, 'Jane', 'Smith', 60000),
141         (3, 'Bob', 'Johnson', 55000);
142
143 -- ✨ update -----[]
144 UPDATE table_name
145 SET column1 = value1, column2 = value2, ...
146 WHERE condition;
147
148 -- Updating a single column for specific rows
149 UPDATE employees
150 SET salary = 55000
151 WHERE department_id = 10;
152
153 -- Updating multiple columns for a specific row
154 UPDATE employees
155 SET salary = 60000, job_id = 'MANAGER'
156 WHERE employee_id = 1;
157
158
159 -- ✨ delete -----[]
160 DELETE FROM table_name
161 WHERE condition;
162
163 -- ✨ alter main syntax : -----[]
164 /*
165     RESTRICT: pas de destruction si l'objet est référencé ou utilisé ailleurs

```

```

166         CASCADE: propage la destruction
167     */
168     ALTER TABLE <nom de la Table>
169     {
170     ADD COLUMN <def Colonne> |
171     DROP COLUMN <nom Colonne> [RESTRICT|CASCADE] |
172     ADD CONSTRAINT <def Contrainte> |
173     DROP CONSTRAINT <nom Contrainte> [RESTRICT|CASCADE] |
174     }
175
176
177 -- ✨ table operations -----[]
178 -- Deleting specific rows based on a condition
179     DELETE FROM employees
180     WHERE department_id = 20;
181
182 -- Deleting all rows from a table
183     DELETE FROM employees;
184
185 -- Adding a new column
186     ALTER TABLE table_name
187     ADD column_name datatype;
188
189 -- Adding a new column with a default value
190     ALTER TABLE table_name
191     ADD column_name datatype DEFAULT default_value;
192
193 -- Adding multiple columns
194     ALTER TABLE table_name
195     ADD (column1 datatype, column2 datatype, ...);
196
197 -- Modifying the datatype of a column
198     ALTER TABLE table_name
199     MODIFY column_name new_datatype;
200
201 -- Modifying the size of a VARCHAR2 column
202     ALTER TABLE table_name
203     MODIFY column_name VARCHAR2(new_size);
204
205 -- Renaming a column
206     ALTER TABLE table_name
207     RENAME COLUMN old_column_name TO new_column_name;
208
209 -- Adding or modifying a default value
210     ALTER TABLE table_name
211     MODIFY column_name DEFAULT new_default_value;
212
213 -- Dropping a default value
214     ALTER TABLE table_name
215     MODIFY column_name DEFAULT NULL;
216
217 -- Dropping a single column
218     ALTER TABLE table_name
219     DROP COLUMN column_name;
220
221 -- Dropping multiple columns

```

```

222     ALTER TABLE table_name
223     DROP (column1, column2, ...);
224
225 -- Adding a primary key constraint
226     ALTER TABLE table_name
227     ADD CONSTRAINT pk_constraint_name PRIMARY KEY (column1, column2, ...);
228
229 -- Adding a unique constraint
230     ALTER TABLE table_name
231     ADD CONSTRAINT unique_constraint_name UNIQUE (column1, column2, ...);
232
233 -- Adding a foreign key constraint
234     ALTER TABLE child_table
235     ADD CONSTRAINT fk_constraint_name
236     FOREIGN KEY (column_name) REFERENCES parent_table (referenced_column);
237
238 -- Dropping a constraint
239     ALTER TABLE table_name
240     DROP CONSTRAINT constraint_name;
241
242 -- Renaming a table
243     ALTER TABLE old_table_name
244     RENAME TO new_table_name;
245
246 -- Truncating a table (removing all rows)
247     TRUNCATE TABLE table_name;
248
249 -- Adding comments to a table
250     COMMENT ON TABLE table_name
251     IS 'This is a comment on the table.';
252
253
254
255 -- ✨ create domain : -----[ ]
256     CREATE DOMAIN <nom domaine> <type> [valeur]
257     [CONSTRAINT nom_contrainte CHECK (condition) ]
258
259 --example :
260     CREATE DOMAIN TypeNomDOC IS VARCHAR2(20);
261     CREATE DOMAIN DATE_RDV IS DATE
262     DEFAULT (CURRENT_DATE)
263     CHECK (VALUE >= CURRENT_DATE)
264     NOT NULL
265
266 -- example 2 :
267     CREATE DOMAIN email_domain AS VARCHAR(255)
268     CHECK (VALUE LIKE '%@%' AND VALUE LIKE '%.%.');
269
270     CREATE TABLE users (
271     user_id INT PRIMARY KEY,
272     username VARCHAR(50) NOT NULL,
273     email email_domain NOT NULL
274     );
275
276 -- ✨ full syntax select -----[ ]
277     SELECT column, group_fonction

```

```

278     FROM tables
279     [ WHERE condition ]
280     [ GROUP BY group_by_expression
281     [ HAVING group_condition ] ]
282     [ ORDER BY column];
283
284 -- ✨select statement -----[:
285     specific COLUMNS :
286     select atr1,atr2 .... from tableName;
287
288 -- ✨all COLUMNS :
289     select * from TableName;
290
291 -- ✨relational operators :
292     =
293     <> or !=
294     >
295     <
296     >=
297     <=
298
299 -- ✨logical operators :
300     and
301     or
302     not
303
304 -- ✨mathimatical operations -----[:
305     SELECT EMPLOYEE_ID,FIRST_NAME,LAST_NAME ,SALARY + 100
306     from EMPLOYEES;
307
308 -- ✨null value -----[:
309     null value is a value that is unavallable unsigned ,unknown
310     or inapplicable
311     null is not the same as zero or a blank space
312     null operand Value = null ;
313
314 -- ✨column alias -----[]
315     SELECT clm1 as aliasName, clm2 aliasName from TableName;
316     accepted character with as => $ # _
317
318
319 -- ✨concatenation OPERATOR : -----[-]:
320     --you can use it to cocatinate between multiple Column in the same time :
321     example :
322     select first_name||' '||last_name "full name " from EMPLOYEES;
323
324 -- ✨q keyword :-----[-]
325     -- exmaple 1:
326     select first_name || ' work in departement ' || department_id
327     from employees;
328     -- example 2:
329     select first_name || q'[ work in departement ]' || department_id
330     from employees;
331
332 -- ✨distinct keyworrd -----[-]
333     to ignore the repetition;

```

```

334 -- example:
335     select distinct first_name || q'[ work in departement ]' || department_id
336     from employees;
337
338 -- ✨ describe -----[-]
339     describe a table columns data type
340     example: you can use describe or desc
341
342     describe employees;
343     -----result-----
344     Name      Null?   Type
345     EMPLOYEE_ID NOT NULL   NUMBER(6)
346     FIRST_NAME          VARCHAR2(20)
347     LAST_NAME    NOT NULL   VARCHAR2(25)
348     EMAIL      NOT NULL     VARCHAR2(25)
349     PHONE_NUMBER          VARCHAR2(20)
350     HIRE_DATE   NOT NULL     DATE
351     JOB_ID     NOT NULL     VARCHAR2(10)
352     SALARY      NUMBER(8,2)
353     COMMISSION_PCT      NUMBER(2,2)
354     MANAGER_ID      NUMBER(6)
355     DEPARTMENT_ID      NUMBER(4)
356     -----
357 /*these you should know when using
358 ✨ the where character strings and date values are enclosed with single marks
359 ✨ character values are case-sensitive and date values are format-sensitive
360 ✨ the default data display format is DD-MON-RR
361 ✨ the alis doesn't work directly with where close
362 */
363
364 -- ✨ where statement-----[-]
365 --syntax :
366     select * from Columns where condition;
367
368 -- example:
369     select *
370     from employees
371     where department_id=90;
372 -- example 2:
373     select *
374     from employees
375     where department_id=90 and first_name='Steven';
376 -- example 3:
377     select *
378     from employees
379     where hire_date='17-oct-03';
380
381 -- ✨ special comparison operators:-----[-]
382     between ..and ... ,
383     in(set) ,
384     like ,
385     is null ,
386     is not null,
387
388 -- example 1:
389     select *

```

```

390         from employees
391         where salary between 10000 and 20000;
392 -- example 2:
393         select *
394         from employees
395         --his start with a character in this range Adel Basma...
396         where first_name between 'A' and 'C';
397 -- example 3:
398         select *
399         from employees
400         where salary in(10000,17000,20000);
401 -- exmaple 4:
402         select *
403         from employees
404         where first_name like 'S%'; --start wiht S
405 -- example 5:
406         select *
407         from employees
408         where first_name like '%s'; --finish with s;
409 -- eample 6:
410         select *
411         from employees
412         where first_name like '%am%'; --include am
413 -- example 7:
414         select *
415         from employees
416         where first_name like '_d%'; -- has d in second letter
417 -- example 8:
418         select *
419         from employees
420         where first_name like '__s%'; --has s in the third letter:
421
422 -- example 9:
423 -- special case when the data contain '_' or '%':
424 /*
425  if we hava a name that contain special character like
426  _ or % we need to espace these ones to prevent oracle
427  from consdering '_' and '%' as keywords of sql
428  */
429         select *
430         from employees
431         where first_name like '/__s%' escape '/';
432 -- example 10:
433         select *
434         from employees
435         where commission_pct is null;
436 -- example 11:
437         select *
438         from employees
439         where commission_pct is not null;
440 -- ♦ compare lists :
441 /*
442  ♦ IN : La condition est vraie si EXP appartient à la liste des valeurs retournées
443         par la sous-requête
444  ♦ ANY : La condition est vraie si la comparaison est vraie pour AU MOINS une
445         des valeurs retournées par la sous-requête

```



```

446 ✨ ALL : La condition est vraie si la comparaison est vraie pour TOUTES
447      les valeurs retournées par la sous-requête
448 ✨ EXISTS (sous-requête)
449      FAUX si Resultat(Sous-requête) = ∅
450      VRAIE si Resultat(Sous-requête) ≠ ∅
451 */
452
453 -- ### 1. `IN` Operator (Appartenance):
454 /*
455      The `IN` operator is used to determine whether a specified
456      value matches any value in a subquery or a list.
457 */
458 -- **Example:**
459 ```sql
460 SELECT column1, column2
461 FROM table
462 WHERE column1 IN (value1, value2, ...);
463 ```
464
465 -- ### 2. `ALL` Operator (À Tous):
466 /*
467      The `ALL` operator compares a value to all values in a set or
468      returned by a subquery.
469 */
470 -- **Example:**
471 ```sql
472 SELECT column1, column2
473 FROM table
474 WHERE column1 > ALL (SELECT other_column FROM another_table);
475 ```
476
477 -- ### 3. `ANY` Operator (Au Moins Un):
478 /*
479      The `ANY` operator compares a value to any value in a set or
480      returned by a subquery.
481 */
482 -- **Example:**
483 ```sql
484 SELECT column1, column2
485 FROM table
486 WHERE column1 > ANY (SELECT other_column FROM another_table);
487 ```
488
489 -- ### 4. `EXISTS` Operator (Non Vide):
490 /*
491      The `EXISTS` operator is used to test for the existence of rows
492      returned by a subquery.
493 */
494 -- **Example:**
495 ```sql
496 SELECT column1, column2
497 FROM table
498 WHERE EXISTS (SELECT 1 FROM another_table WHERE condition);
499 ```
500 In these examples:
501 /*

```

```

502      - **Subquery:** A subquery is a query nested within another query.
503      It can be used with `IN`, `ALL`, `ANY`, or `EXISTS` operators.
504
505      - **List:** A list of values can be used with the `IN` operator.
506
507      - **Comparison:** The `ALL` and `ANY`
508      operators are often used in comparison expressions.
509
510      These operators provide flexibility and efficiency in constructing
511      complex queries to filter and compare data in various ways.
512      */
513
514
515  -- 💎order by -----[]
516  /*
517      💎asc : ascending order,default
518      💎desc :descending order
519      by default null come last in ascending order :
520      by default null come first in descending order;
521      you can change by adding : nulls first(or last)
522  */
523  syntaxe :
524  order by ColumnName orderByWhat(asc desc)
525  -- example 1:
526      select *
527      from employees
528      order by hire_date;
529
530  -- example 2:
531      select *
532      from employees
533      order by hire_date desc;
534
535  --example 3: --usngin aliases in ordering :
536      select salary as n,first_name
537      from employees
538      order by n;
539
540  --example 4: --sort by expression
541      select salary +100 as n,first_name
542      from employees
543      order by n;
544
545  --example 5:-- sort by column not slected
546      select first_name,salary
547      from employees
548      order by department_id; -- but this is not logical
549
550  --example 6: --sort using multiple Columns
551      select department_id,first_name,salary
552      from employees
553      order by department_id,first_name ;-- you specfiy the first_name order asc desc
554
555  --example 7: --sort by column Number
556      select department_id,first_name,salary
557      from employees

```

```

558         order by 3; --3=salary
559
560 -- ✨ fetch statement -----[]
561     --show just the N rows
562     fetch first N rows only;
563 -- example :
564     select employee_id,first_name
565     from employees
566     order by employee_id
567     fetch first 5 rows only;
568
569 -- ✨ show N% rows only:
570     fetch first n percent rows only;
571 -- example:
572     select employee_id,first_name
573     from employees
574     order by employee_id
575     fetch first 50 percent rows only ;
576 --show from an offset to specific Number show(from offset to end: depend on offset)
577     offset start rows fetch next end rows only;
578 -- example:
579     select employee_id,first_name
580     from employees
581     order by employee_id
582     offset 5 rows fetch next 5 rows only;
583
584 -- ✨ ties -----[-]
585 --with ties means add also rows with same ordered value
586
587 --without rows that have the same ordered value
588     select employee_id,first_name,salary
589     from employees
590     order by salary desc
591     fetch first 2 rows only;
592 --result :
593     EMPLOYEE_ID,FIRST_NAME,SALARY
594         100      , Steven ,24000
595         101      , Neena ,17000
596
597 --with ties:
598     select employee_id,first_name,salary
599     from employees
600     order by salary desc
601     fetch first 2 rows with ties;
602 --result:
603     EMPLOYEE_ID,FIRST_NAME,SALARY
604     EMPLOYEE_ID,FIRST_NAME,SALARY
605         100      , Steven ,24000
606         101      , Neena ,17000
607         102      , Lex ,17000
608
609 -- ✨ substitution variables : -----[-]
610 /*
611     temporarily store values with single-ampersand (&) and double-ampersand (&&)
612 substitution
613     1- & :the variable will be discarded after it is used

```

```

613      2- use '' when using varchar
614      3- && + defining of the variable
615      user it to supplement the following:
616      where rule_expression_conditions;
617      order by clauses
618      column expressions
619      table names
620      entire select statements
621  */
622  -- example 1:
623  /*
624      in this code we declare a variable that will contains the value that user
625      will enter in the prompt that will pop-up on the screen
626  */
627      select employee_id,first_name
628      from employees
629      where employee_id=&UserChoiceId;
630      -- assum that user entered 100 :
631      -- result :
632      EMPLOYEE_ID,FIRST_NAME
633      100      , Steven
634
635  -- example 2: with varchar
636      select employee_id,first_name,salary
637      from employees
638      where first_name='&FristName';
639  -- example 3:
640      select employee_id,last_name,job_id,&column_name
641      from employees
642      order by &order_column;
643
644
645  -- ✨ define and undefine -----[-]
646  /*
647      use the define command to create ans assign a value to a variable
648      use the undefine commant to remove a variable
649  */
650
651  -- example1:
652      -- the prompt no will pop-up because employee_num defined
653      define employee_num=100;
654      select employee_id, last_name, salary
655      from employees
656      where employee_id=&employee_num;
657
658  -- example 2:
659      define employee_num=100;
660      undefine employee_num; -- remove the variable then it will pop-up the prompt
661      select employee_id,last_name,salary
662      from employees
663      where employee_id=&employee_num;
664
665  -- ✨ `change the prompt message -----[-]:
666  /*
667      you can change the prompt as follow
668      but it should executed as a script

```

```

669 */
670 syntaxe :
671 ACCEPT Variable_name PROMPT 'Messgae' -- accept =define + prompt change message
672 -- this message it will just assoscited with only this variable
673
674 -- example 1:
675     ACCEPT User_id PROMPT 'Please enter the user id :';
676     select employee_id,first_name,last_name,salary
677     from employees
678     where employee_id=&User_id;
679
680 -- example 2:
681     select first_name,last_name,&&User_column
682     from employees
683     order by &User_column;
684 -- example 3:
685     ACCEPT User_column PROMPT 'Pelase enter the Column :';
686     select first_name,last_name,&User_column
687     from employees
688     order by &User_column;
689
690 -- ✨ verify -----[-]
691 /*
692 Use the command to toggle the display of the substitution variable,
693 both before and after sql developer repalces substiution variables with values :
694 */
695
696 -- example:
697     set verify on --add the stat before edit getting value from user and after
698     select first_name,last_name,&User_column
699     from employees
700     order by &User_column;
701
702 -- using set define off =>turn off the prompt
703 -- example:
704     set define off; -- will stop the prompt
705     select *
706     from departments
707     where department_name like '%&t%' ; -- &t it's not will consirted as a variable
708
709 -- ✨ character function -----[-]
710 /*
711     there are 2 types for character functions:
712
713     1- case conversion functions (upper ,lower ,initcap)
714         initcap : camle case
715         i can use them in select,where,order by,
716
717     2- character manipulation functions
718 */
719 -- ✨ 1- case conversion functions (upper ,lower ,initcap)
720 -- example 1:
721     select employee_id,first_name,upper(first_name),lower(first_name),
initcap(first_name)
722     from employees;
723 -- example 2:

```

```

724         select employee_id,first_name,upper(first_name),lower(first_name),
initcap(first_name)
725         from employees
726         where upper(first_name) = 'PATRICK';
727
728     -- ✨2- character manipulation functions;
729
730     -- ✨a- concat function
731         concat(column1,column2);
732     -- example :
733         select employee_id,first_name,concat(first_name,last_name)
734         from employees;
735
736     -- ✨b-substr function
737         substr(column,start,distance);
738     -- if you didn't specify the distance value it will be take to the end (all
string )
739     -- if you pass a negative value .then the count start from the end
740     -- example :
741         select employee_id,
742         first_name,
743         substr(first_name,1,3),
744         substr(first_name,2,4),
745         substr(first_name,2),
746         substr(first_name,-3)
747         from employees
748
749     -- ✨c-length(Column);
750     -- example:
751         select first_name,length(first_name)
752         from employees;
753
754     -- ✨to_char function -----[ ]
755     /*
756         value: The value to be converted. It can be a date, timestamp, or number.
757         format: The format mask that defines how the value should be converted.
758         For dates and timestamps, it specifies the desired date or time format.
759         For numbers, it specifies the number format.
760
761         ✓YYYY :year
762         ✓MM :month
763         ✓DD :day
764         ✓Day :dayName
765         ✓hh24 :hours
766         ✓MI :minutes
767         ✓SS :seconds
768
769     */
770     TO_CHAR(value, format)
771     -- example (select with year ) :
772         select with year
773         SELECT *
774         FROM your_table
775         WHERE TO_CHAR(your_date_column, 'YYYY') = '2023';
776     -- convert date to heour :
777         to_number(to_char(date,'hh24'));
778

```

```

779 -- extract HH:mm:ss
780     SELECT
781         TO_NUMBER(TO_CHAR(SYSDATE, 'HH24')) AS current_hour,
782         TO_NUMBER(TO_CHAR(SYSDATE, 'MI')) AS current_minute,
783         TO_NUMBER(TO_CHAR(SYSDATE, 'SS')) AS current_second
784     FROM DUAL;
785
786 -- extract YYYY::MM::dd
787     SELECT
788         TO_CHAR(SYSDATE, 'YYYY') AS current_year,
789         TO_CHAR(SYSDATE, 'MM') AS current_month,
790         SELECT TO_CHAR(SYSDATE, 'DD') AS current_day FROM DUAL;
791     FROM DUAL;
792
793 -- now day name :
794     select TO_CHAR(SYSDATE, 'Day') AS day_name
795     from dual;
796
797
798 -- change date format :
799     Alter session set nls_date_format='dd/mm/yyyy';
800
801 -- group functions : -----[]
802 --*Count(*|[ DISTINCT|ALL] expr)
803 /*
804     Le nombre de ligne de expr
805 */
806 --*Avg( [ DISTINCT | ALL] expr)
807 /*
808     Valeur moyenne de expr, en ignorant les valeurs
809     NULL
810 */
811 --*Min( [ DISTINCT | ALL] expr)
812 /*
813     Valeur minimale de expr, en ignorant les valeurs
814     NULL
815 */
816 --*Max( [ DISTINCT | ALL] expr)
817 /*
818     Valeur maximale de expr, en ignorant les valeurs
819     NULL
820 */
821 --*Sum( [ DISTINCT | ALL] expr)
822 /*
823     Somme des valeurs de expr, en ignorant les valeurs
824     NULL
825 */
826 -- ✨ GROUP BY: -----[]
827 /*
828     The `GROUP BY` and `HAVING` clauses in SQL are used together to perform aggregate
829     functions on groups of rows and filter the results based on the grouped data.
830     ✓ - The `GROUP BY` clause is used to group rows that have the same values in specified
831         columns into summary rows, like a summary table.
832     ✓ - It is often used with aggregate functions (e.g., `COUNT`, `SUM`, `AVG`, `MAX`,
833     `MIN`)
834         to perform calculations on each group of rows.

```

```

834      ✓ - The columns listed in the `GROUP BY` clause are the grouping columns, and each
835          unique combination of values in these columns forms a group.
836      */
837      -- *Example:
838      ```sql
839      SELECT department_id, AVG(salary) as avg_salary
840      FROM employees
841      GROUP BY department_id;
842      ```
843
844      -- ✨ HAVING: -----[]
845      /*
846      ✓ - The `HAVING` clause is used in combination with `GROUP BY` to filter the results
      based
847          on the result of aggregate functions applied to the groups.
848      ✓ - It is similar to the `WHERE` clause but is specifically designed for filtering
849          results after the grouping has been applied.
850      ✓ - It allows you to specify conditions on the results of aggregate functions.
851      */
852      --*Example:
853      ```sql
854      SELECT department_id, AVG(salary) as avg_salary
855      FROM employees
856      GROUP BY department_id
857      HAVING AVG(salary) > 50000;
858      ```
859
860      -- ✨ change column format : -----[]
861      COLUMN example_column FORMAT A100;
862
863
864      -- ✨ joins : -----[]
865      /*
866          Equijointure ( jointure naturelle)
867          Requêtes sur plusieurs tables: la jointure
868          Autojointure (jointure sur la même table)
869          Non-équijointure (jointure par non égalité, théta jointure)
870      */
871      -- *1. INNER JOIN:
872      /*
873          The INNER JOIN keyword selects records that have matching values
874          in both tables. It returns only the rows where there is
875          a match in the specified columns.
876      */
877      SELECT columns
878      FROM table1
879      INNER JOIN table2 ON table1.column_name = table2.column_name;
880
881
882      -- *2. LEFT (OUTER) JOIN:
883      /*
884          The LEFT JOIN keyword returns all records from the left table
885          (table1), and the matched records from the right table (table2).
886          The result is NULL from the right side if there is no match
887      */
888      SELECT columns

```



```

889      FROM table1
890      LEFT OUTER JOIN table2 ON table1.column_name = table2.column_name;
891      -- or
892          SELECT columns
893          FROM table1,table2
894          where table1.column_name = table2.column_name(+);
895
896      -- *3. RIGHT (OUTER) JOIN
897      /*
898          The RIGHT JOIN keyword returns all records from the right
899          table (table2), and the matched records from the left table
900          (table1). The result is NULL from the left side when there
901          is no match.
902      */
903      SELECT columns
904      FROM table1
905      RIGHT OUTER JOIN table2 ON table1.column_name = table2.column_name;
906      -- or :
907          SELECT columns
908          FROM table1,table2
909          where table1.column_name(+) = table2.column_name;
910
911      -- *4. FULL (OUTER) JOIN:
912      /*
913          The FULL JOIN keyword returns all records when there is a
914          match in either the left (table1) or the right (table2)
915          table records.
916      */
917      SELECT columns
918      FROM table1
919      FULL OUTER JOIN table2 ON table1.column_name = table2.column_name;
920
921
922      -- *5. SELF JOIN:
923      /*
924          A self-join is a regular join, but the table is joined with
925          itself. This is useful for hierarchical structures or when
926          relating records within the same table.
927      */
928      SELECT columns
929      FROM table1 alias1
930      JOIN table1 alias2 ON alias1.column_name = alias2.column_name;
931
932      -- ✨ Requêtes imbriquées -----[ ]
933      SELECT colonnes_de_projection
934      FROM table
935      WHERE expr operator (
936      SELECT colonnes_de_projection
937      FROM table
938      WHERE ....
939      );
940
941      -- ✨ list all constraints : -----[ ]
942      SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE, TABLE_NAME
943      FROM USER_CONSTRAINTS;
944

```

```

945 -- ♦ Operators: -----[]
946
947 -- *### 1. `UNION` Operator:
948 /*
949     The `UNION` operator is used to combine the result sets of two or more SELECT
950     statements into a single result set. It removes duplicate rows from the
951     combined result set.
952 */
953
954 -- **Syntax:**
955 ```sql
956 SELECT column1, column2 FROM table1
957 UNION
958 SELECT column1, column2 FROM table2;
959 ```
960
961 -- *### 2. `UNION ALL` Operator:
962 /*
963     The `UNION ALL` operator is similar to `UNION`, but it does not remove duplicate
964     rows. It combines all rows from the result sets of multiple SELECT statements,
965     including duplicates.
966 */
967
968 -- **Syntax:**
969 ```sql
970 SELECT column1, column2 FROM table1
971 UNION ALL
972 SELECT column1, column2 FROM table2;
973 ```
974
975 -- *### 3. `INTERSECT` Operator:
976 /*
977     The `INTERSECT` operator returns the common rows between the result sets
978     of two SELECT statements. It returns only the rows that appear in both result
979     sets.
980 */
981 -- **Syntax:**
982 ```sql
983 SELECT column1, column2 FROM table1
984 INTERSECT
985 SELECT column1, column2 FROM table2;
986 ```
987
988 -- *### 4. `MINUS` Operator (or `EXCEPT` in some databases):
989 /*
990     The `MINUS` operator returns the rows that appear in the result set of
991     the first SELECT statement but not in the result set of the second
992     SELECT statement. It is often used to find the set difference between
993     two result sets.
994 */
995 -- **Syntax:**
996 ```sql
997 SELECT column1, column2 FROM table1
998 MINUS
999 SELECT column1, column2 FROM table2;

```

```
1000      ```\n1001      /*\n1002          Remember, for these set operations to work, the number\n1003          of columns and their data types in the corresponding positions in the SELECT\n1004          statements must match.\n1005      */
```