Databse test



**Câu 1**: Giải thích các hàm thông dụng sau

|  |  |  |
| --- | --- | --- |
| Stt | Tên Hàm | Mục đích sử dụng & nên sử dụng khi nào |
| 1 | Count() | -Là một hàm tổng hợp trả về số dòng trong một nhóm.  -Syntax: Count( [ ALL | DISTINCT | \* ] expression )  -Count() nhận một mệnh đề có thể là ALL, DISTINCT, hoặc \*:  +Count(\*) : bao gồm giá trị duplicate và giá trị null.  +Count(DISTINCT expression): bao gồm unique values và non-null values.  +Count(ALL expression): bao gồm duplicate values, và non-null values.  -Mặc định khi để trống là ALL. |
| 2 | Sum() | -Là một hàm tổng hợp trả về giá trị tổng của tất cả, hoặc của những giá trị phân biệt của một tập giá trị.  -Syntax: SUM( [ALL | DISTINCT] expression)  + DISTINCT : tính tổng của những giá trị phân biệt.  + ALL : tính tổng của tất cả giá trị, bao gồm cả duplicate values.  -Mặc định : ALL |
| 3 | MAX() | -Là hàm trả về giá trị lớn nhất của một tập các giá trị (không tính giá trị null). |
| 4 | MIN() | -Là hảm trả về giá trị nhỏ nhất của một tập giá trị(không tính giá trị null). |
| 5 | NVL() | -Để thay thế một giá trị null bằng một giá trị cụ thể khác.  -NVL(e1,e2). Nếu e1 = null, trả về e2 và ngược lại.  +Hai tham số e1 và e2 có thể cùng hoặc khác kiểu dữ liệu. Nếu khác kiểu sẽ được chuyển đổi dựa trên quy tắc sau:  \*Nếu e1 là kiểu kí tự thì e2 sẽ được chuyển đổi thành kiểu dữ liệu của e1 trước khi so sánh với null.  \*Nếu e1 là kiểu số thì Oracle sẽ kiểm tra xem kiểu dữ liệu nào sẽ có độ ưu tiên cao hơn để chuyển đổi ngầm định tham số còn lại về kiểu có độ ưu tiên cao hơn.  \*Nếu không thể chuyển đổi các kiểu dữ liệu thì sẽ phát sinh lỗi. |
| 6 | TO\_CHAR() | -Là hảm chuyển đổi dữ liệu có kiểu DATE hoặc INTERVAL thành dạng String có format được chỉ định.  -TO\_CHAR(expr [, date\_format] [, nslparam]);  +expr: giá trị kiểu Date hoặc Interval cần được chuyển đổi  +date\_format: (Optional) Nếu không truyền thì TO\_CHAR sẽ sử dụng các format mặc định của từng kiểu dữ liệu.  +nslparam: (Optional) chỉ định ngôn ngữ, tên cho ngày và tháng. Vd: Monday hay Mon, January hay Jan. |
| 7 | T0\_DATE() | -Chuyển định dạng của một chuỗi hoặc một dãy số sang định dạng ngày tháng.  -TO\_DATE (string, format, nls\_language);  +string: giá trị dạng chuỗi cần được chuyển đổi  +format: (Optional) format đích mà cần phải chuyền về. Mặc định là: DD-MON-YY (31-DEC-2000).  +nls\_language: (Optional) Chỉ định ngôn ngữ cho ngày và tháng. Mặc định là ngôn ngữ của phiên làm việc hiện tại. |
| 8 | TO\_NUMBER() | -Chuyển đổi dạng string thành dạng số.  -**TO\_NUMBER**(expr [, fmt [, 'nlsparam' ] ]);  +string: giá trị cần chuyển đổi.  +format: (Optional) format đích.  +nls\_language: (Optional) Chỉ định ngôn ngữ cho ngày và tháng. Mặc định là ngôn ngữ của phiên làm việc hiện tại. |
| 9 | SUBSTR() | -Lấy một chuỗi con từ một chuỗi cho trước.  - SUBSTR( str, start\_position [, substring\_length] );  +str: chuỗi cho trước.  +start\_position: vị trí bắt đầu. Nếu >=0 tự đầu chuỗi, nếu <0 từ cuối chuỗi.  +substr\_length: độ dài của chuỗi con cần lấy. Mặc định nếu kh truyền vào thì là lấy hết. Nếu <1 thì trả về chuỗi rỗng. |
| 10 | REPLACE() | - REPLACE(string\_expression, string\_pattern [,string\_replacement]) : thay thế tất cả các lần xuất hiện thỏa string\_pattern thành string\_replacement  +string\_expression: giá trị dạng chuỗi.  +string\_pattern: chuỗi con cần được thay thế.  +string\_replacement: chuỗi thay thế. |
| 11 | REVERSE() | -Để đảo ngược một chuỗi. |
| 12 | DECODE() | - DECODE (expression , search1, result1[, search2, result2], ...,[,searchn,resultn] [, default]);  - Dùng để thực hiện cấu trúc rẽ nhánh (if else, case when)  +expression: biểu thức để so sánh.  +search: giá trị để so sánh với biểu thức.  +result: giá trị sẽ trả về nếu biểu thức khớp với giá trị so sánh.  +default: giá trị có thể trả về mặc định khi biểu thức và giá trị so sánh không khớp nhau. Với default là Optional 🡪 Decode sẽ trả về null nếu biểu thức không khớp với giá trị so sánh. |
| 13 | TRUNC() | -Dùng để thu gọn một số đến vị trí decimal\_places  - TRUNC(number [, decimal\_places]);  +number: số ban đầu  +decimal\_places: vị trí muốn lấy sau dấu phẩy, đây là một số nguyên âm hoặc dương, mặc định là 0. |
| 14 | LENGTH() | - Trả về độ dài của một chuỗi.  - LENGTH(string\_expression); |
| 15 | lPAD() | -Thềm vào bên trái của chuỗi với tập ký tự được xác định trong tham số của nó.  - LPAD(source\_string, target\_length [,pad\_string]);  +source\_string: chuỗi muốn áp dụng  +target\_length: độ dài của kết quá trả về. Nếu nhập vào số nhỏ hơn chuỗi gốc thì nó sẽ cắt ngắn chuỗi gốc.  +pad\_string là chuỗi cần chèn vào phía bên trái của chuỗi. |
| 16 | RPAD() | -Tương tự như lPAD() nhưng thêm vào bên phải. |
| 17 | TRIM() | -Để xóa ký tự khoảng trắng hoặc ký tự bất kì tại vị trí được quy ước.  - TRIM( [ [ LEADING | TRAILING | BOTH ] trim\_character FROM ] trim\_source);  +LEADING, TRAILING, BOTH : tham số chỉ định vị trí sẽ được xóa đầu, cuối và cả hai.  +trim\_character : các ký tự muốn xóa  +trim\_source: chuỗi gốc. |
| 18 | LTRIM() | -Xóa các ký tự từ ngoài cùng bên trái trong một tập ký tự được chỉ định.  - LTRIM(trim\_source,[set]);  +trim\_source: string cần được loại bỏ các ký tự không mong muốn.  +set: các ký tự cần xóa |
| 19 | RTRIM() | -Xóa bỏ tất cả ký tự xuất hiện trong một tập giá trị cụ thể từ đầu bên phải của một chuỗi.  - RTRIM(trim\_source,[set]); |
| 20 | ROUND() | -Làm tròn số.  - ROUND(number [,decimal\_places]):  +number: số cần được làm tròn  +decimal\_places: số thập phân cần làm tròn, nếu để trống sẽ mặc định loại bỏ phần thập phân. |
| 21 | ADD\_MONTHS() | -Dùng để thêm tháng vào giá trị Date.  - ADD\_MONTHS(date\_expression, month):  +date\_expression: giá trị Date.  +month: số tháng cần được thêm vào. Nếu truyền vào số âm có nghĩa là trừ. |

**Câu 2**:

SELECT \*

FROM MDM\_CUSTOMER A

WHERE A.CUST\_LGL\_ENG\_NM LIKE ‘%\\_LOGISTICS%’ ESCAPE ‘\’

ORDER BY A.CUST\_LOCL\_LANG\_NM NULLS FIRST

A) Vui lòng giải thích ý nghĩa của câu SQL trên:

- SELECT \* : hiển thị tất cả các cột (fields)

-FROM MDM\_CUSTOMER A: từ bảng(object) MDM\_CUSTOMER và đổi tên thành A

-WHERE A.CUST\_LGL\_ENG\_NM LIKE ‘%\\_LOGISTICS’ ESCAPE ‘\’: điều kiện CUST\_LGL\_ENG\_NM chứa string có dạng \_LOGISTICS

-ORDER BY A.CUST\_LOCL\_LANG\_NM NULLS FIRST: sắp xếp tăng dần theo CUST\_LOCL\_LANG\_NM và giá trị NULL sẽ đứng trước giá trị non-null

🡪 Hiển thị tất cả các cột trong bảng MDM\_CUSTOMER theo điều kiện và sắp theo cột LOCL\_LANG\_NM với các giá trị NULL đứng trước giá trị non-null.

B) ý nghĩa của việc dùng ESCAPSE

- Để chỉ định ký tự sau ESCAPE là ký tự thông thường. Vd: ‘%\_LOGISTICS%’ nếu không dùng ESCAPE thì dấu ‘\_’ ở đây có nghĩa là cần ít nhất một ký tự trước chữ LOGISTICS, nhưng ở đây ý ta là dấu gạch dưới ‘\_’. Dấu gạch dưới (\_) là dấu gạch dưới(\_) chứ không phải là placeholder cho ký tự nào đó. 🡺 Trong trường hợp này ta cần dùng ESCAPE.

-Reference: [Oracle Escape Characters (dba-oracle.com)](http://www.dba-oracle.com/tips_oracle_escape_characters.htm)

C) Ý nghĩa của việc dung Nulls First.

-Sắp xếp các giá trị NULL trước các giá trị non-null

D) Ý nghĩa của việc dung alias, có nên dung alias trong mọi trường hợp không?

-Không cần thiết phải sử dụng alias trong mọi trường hợp. Alias hữu ích khi: Có nhiều bảng(table) trong một câu truy vấn, functions được sử dụng, tên cột dài hoặc khó đọc(không mang ý nghĩa), hai hoặc nhiều cột được kết hợp với nhau (combined).

->Để dễ đọc, dễ viết. Vì đối với tên cột có dạng viết tắt nên sẽ gây khó hiểu cho người đọc nên ta dùng alias để hiển thị tên cột đó dưới dạng tên mang ý nghĩa hơn.

->Bắt buộc phải dùng. Vd: khi sử dụng self-join. Giả sử ta có bảng Employee có cột managerId, mà managerId lại tham chiếu đến id của bảng Employee, lúc này thì self-join ta cần phải alias tên của 1 trong 2 bảng, để khi ta gọi tên cột như id hay managerId thì Oracle hiểu ta đang nói đến bảng nào.

-Reference: [SQL Aliases (w3schools.com)](https://www.w3schools.com/sql/sql_alias.asp)

**Câu 3**:

SELECT \*

FROM MDM\_CUSTOMER

WHERE 1 = 1

AND CUST\_CNT\_CD = 'JP'

AND TO\_CHAR(CUST\_SEQ) = '201791'

Theo bạn câu trên cách dung TO\_CHAR(CUST\_SEQ) = '201791' có hợp lý không, tại sao?

-Không hợp lý vì trong trường hợp này truy vấn của ta sẽ phải lặp qua các cột để tìm cột CUST\_SEQ sau đó dùng TO\_CHAR() lên cột CUST\_SEQ. Thay vì như vậy thì so sánh thẳng luôn không cần phải To\_Char() vì CUST\_SEQ là dữ liệu số.

**Câu 4**: cho câu SQL và kết quả như hình bên dưới



A) Giải thích ý nghĩa COUNT(\*), COUNT(1), COUNT(CUST\_LOCL\_LANG\_NM)

-COUNT(\*): đếm tổng số hàng(records/rows) trong bảng và bao gồm các giá trị null và duplicate.

-COUNT(1): thêm một cột giả có giá trị là 1 và đếm tổng số hàng trong bảng bao gồm duplicate và null.

=> COUNT(\*) và COUNT(1) có kết quả trả về như nhau.

- COUNT(CUST\_LOCL\_LANG\_NM): đếm tất cả các hàng trong cột CUST\_LOCL\_LANG\_NM không bao gồm giá trị NULL.

-Reference: [sql - What does "select count(1) from table\_name" on any database tables mean? - Stack Overflow](https://stackoverflow.com/questions/181272/what-does-select-count1-from-table-name-on-any-database-tables-mean)

[sql server 2005 - Not getting the correct count in SQL - Stack Overflow](https://stackoverflow.com/questions/169784/not-getting-the-correct-count-in-sql)

B) Tại sao COUNT(CUST\_LOCL\_LANG\_NM) lại bằng 0

Vì cột CUST\_LOCL\_LANG\_NM chỉ có giá trị null.

Graphical user interface, text, application

Description automatically generated

**Câu 5**: có 2 cách như bên dưới, cách nào tốt tại sao

|  |  |  |
| --- | --- | --- |
|  | **Cách 1** | **Cách 2** |
|  | NVL(SUM(COL1),0) | SUM(NVL(COL1,0)) |

-C1: Gọi NVL() sau khi tính tổng xong bất kể có bao nhiêu hàng(rows).

-C2: Mỗi dòng đều gọi NVL()

🡪 C1 tốt hơn.

**Câu 6**: có 2 cách như bên dưới, cách nào tốt tại sao

|  |  |  |
| --- | --- | --- |
|  | **Cách 1** | **Cách 2** |
|  | NVL(SUM(COL1),0) + NVL(SUM(COL2),0) | Ex.1] SUM(NVL(COL1 + COL2,0))  Ex.2] NVL(SUM(COL1 + COL2),0) |

-

**Câu 7**: có 2 cách như bên dưới, cách nào tốt tại sao

|  |  |  |
| --- | --- | --- |
|  | **Cách 1** | **Cách 2** |
|  | SELECT A.CUST\_NO, A.ORD\_NO, A.PRO\_CD, B.PROD\_NM  FROM TB\_ORD A,  TB\_PROD B  WHERE 1 = 1  AND A.PRO\_CD = B.PROD\_CD  AND B.PROD\_CD IN (SELECT PROD\_CD FROM TB\_PROD D WHERE D.PROD\_CD = A.PRO\_CD AND PROD\_UNIT\_AMT < 800); | SELECT A.CUST\_NO, A.ORD\_NO, A.PRO\_CD, B.PROD\_NM  FROM TB\_ORD A,  TB\_PROD B  WHERE 1 = 1  AND A.PRO\_CD = B.PROD\_CD  AND EXISTS (SELECT D.PROD\_CD FROM TB\_PROD D WHERE D.PROD\_CD = A.PRO\_CD AND D.PROD\_UNIT\_AMT < 800); |

**IN** đc dung khi nào và EXISTS đc dung khi nào.

-C1: Dùng IN khi kết quả truy vấn phụ nhỏ. IN thì sẽ so sánh từng phần tử trong danh sách (nhiều phép OR gộp lại).

-C2: Dùng EXISTS khi kết quả truy vấn phụ lớn. EXISTS thì kiểm tra xem truy vấn phụ có trả về kết quả hay không (số dòng (records) có lớn hơn 1 hay không).

Process: SQL engine compares all values in the IN condition. While, the SQL engine stops the process as soon as finding a single positive condition in EXISTS.

References: [What is the Difference Between IN and EXISTS in Oracle - Pediaa.Com](https://pediaa.com/what-is-the-difference-between-in-and-exists-in-oracle/#:~:text=IN%20is%20a%20clause%20or,and%20create%20subquery%20in%20Oracle.)

**Câu 8**: có 2 cách như bên dưới, cách nào tốt tại sao

|  |  |  |
| --- | --- | --- |
|  | **Cách 1** | **Cách 2** |
|  | SELECT A.CUST\_NO, A.ORD\_NO, A.PRO\_CD, B.PROD\_NM  FROM TB\_ORD A,  TB\_PROD B  WHERE 1 = 1  AND A.PRO\_CD = B.PROD\_CD  AND B.PROD\_CD IN ('00001','00002'); | SELECT A.CUST\_NO, A.ORD\_NO, A.PRO\_CD  , (SELECT B.PROD\_NM FROM TB\_PROD B WHERE B.PROD\_CD = A.PRO\_CD) AS PROD\_NM  FROM TB\_ORD A  WHERE 1 = 1  AND A.PRO\_CD IN ('00001','00002'); |

-C2 tốt hơn vì bảng TB\_ORD đã được bỏ bớt các dòng dữ liệu vi phạm và khi đó ta thực hiện subquery ở SELECT sẽ ít lần. Nếu ta JOIN từ 2 bảng như cách 1 thì trường hợp 2 bảng lớn sẽ kiến cho chi phí khi JOIN vô cùng lớn mà bảng sau khi JOIN lại còn tồn tại vi phạm, ta lại cần WHERE để loại bỏ.

**Câu 9**: cho số 8988.80 vui lòng xuất ra định dạng $8,988.800

SELECT to\_char(9888.80, ‘$9,999.999’) num

FROM dual;

**Câu 10**: cho số 8988.80, 820988.80 vui lòng xuất ra định dạng $8,000.000, $820,000.000

SELECT

to\_char(trunc(8988.80, -3), '$9,999.999') num1,

to\_char(trunc(820988.80, -3), '$999,999.999') num2

FROM dual;

**Câu 11**: Cho cấu SQL và kết quả như sau:



Như hình trên cả 2 A và B điêu substr từ 1, đến 3 tại sao kết quả lại khác nhau.

-Nếu không có ‘fm’ thì sẽ xuất hiện khoảng trắng (khoảng trắng dùng để thể hiện dấu +/-) ở kết quả trả về. Vd: length(to\_char(98765, ‘fm00000’)) là 5, còn length(to\_char(98765, ‘00000’)) là 6.

🡪 Do đó khi dùng hàm substr() ta thu được 2 kết quả khác nhau.

Graphical user interface, text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Câu 12**: Viết Câu SQL xuất ra, Ngày hiện tại, này hôm qua, ngày mai

SELECT

sysdate - 1 yesterday,

sysdate today,

sysdate + 1 tomorrow

FROM dual;

**Câu 13**: ta có table (**TB\_ORD**), yêu cầu viết câu SQL để generate ORD\_NO có đô dài 10 tự với format sau: yyyymmdd000Seq, ví dụ hnay là 20191028 và chưa có seq nào thì ORD\_NO sẽ là 201910280001, và nếu đã tồn tại ORD\_NO 201910280001 thì nó sẽ là 201910280002

Select to\_char(sysdate, 'YYYYMMDD') || to\_char(count(\*) +1, ‘fm0000’) as seq

From tb\_ord

Where ord\_dttm like to\_char(sysdate, 'YYYYMMDD') || '%'

**Câu 14**: ta có table (**MDM\_CUSTOMER**) và dữ liệu như bên dưới



Các field liên quan: CUST\_CNT\_CD, CUST\_SEQ, CUST\_GRP\_HRCHY\_CD, CUST\_GRP\_ID

Dữ liệu cột CUST\_GRP\_HRCHY\_CD có thể có(**I**: individual, **C**: Country, **G**: Global)

A) Viết câu SQL tìm CUST\_GRP\_ID sao cho: CUST\_GRP\_HRCHY\_CD có **I** hoặc **C** nhưng không có **G**

**SELECT cust\_grp\_id**

**FROM (**

**SELECT cust\_grp\_id, cust\_grp\_hrchy\_cd**

**FROM mdm\_customer**

**WHERE cust\_grp\_id NOT IN (**

**SELECT cust\_grp\_id**

**FROM mdm\_customer**

**WHERE cust\_grp\_hrchy\_cd LIKE 'G'**

**)**

**GROUP BY cust\_grp\_id, cust\_grp\_hrchy\_cd**

**) GROUP BY cust\_grp\_id;**

B) Viết câu SQL tìm CUST\_GRP\_ID sao cho: CUST\_GRP\_HRCHY\_CD có **G** và có **I** nhưng không có **C**

**Câu 15**: ta có table (**TB\_PROD**) và dữ liệu như bên dưới



Viets cấu SQL để suất ra kêt quả như sau:

1. Lấy max(PROD\_UNIT\_AMT)
2. Lấy giá trị min(PROD\_UNIT\_AMT)
3. Lấy giá trị trung bình PROD\_UNIT\_AMT
4. Lấy tên của sản phẩm có PROD\_UNIT\_AMT lớn nhất

SELECT

MAX(prod\_unit\_amt) max\_amt,

MAX(prod\_unit\_amt) min\_amt,

AVG(prod\_unit\_amt) avg\_amt,

FROM tb\_prod

WHERE prod\_unit\_amt IS NOT NULL;

Kết quả phải ra đc như sau:



**Câu 16**: ta có table (**TB\_ORD**) và dữ liệu như bên dưới



A) viết cấu SQL lấy ra top3 sản phẩm đc bán nhiều nhất**.**

SELECT \* FROM (

SELECT pro\_cd,

DENSE\_RANK()

OVER(

ORDER BY COUNT(\*) DESC

) as rank

FROM tb\_ord

GROUP BY pro\_cd ) temp

WHERE temp.rank <=3;

B) Viết cấu SQL lấy ra cái ORD\_DT, ORD\_TM, PROD\_CD gần nhất theo CUST\_NO

Kết quả mong đợi như sau:



SELECT cust\_no,

ord\_dttm,

ord\_no,

pro\_cd

FROM (

SELECT cust\_no,

ord\_dttm,

ord\_no,

pro\_cd,

ROW\_NUMBER()

OVER(PARTITION BY cust\_no ORDER BY ord\_dttm DESC) rank

FROM tb\_ord

) temp

WHERE temp.rank = 1;

C) viết cấu SQL report xem trong tháng 06, 07, 08, 09 cảu 2019 sản phẩm có mã code là 00001bán đc bao nhiêu cái.

Kết quả mong đợi nhưu sau: left outer join partition - 130



D) giả sư lúc đầu sản phẩn 00001 có 100 cái, viết report để tính số lương remain theo tháng 06, 07, 08, 09



WITH report as(

SELECT '201906' as dt from dual

UNION ALL

SELECT '201907' as dt from dual

UNION ALL

SELECT '201908' as dt from dual

UNION ALL

SELECT '201909' as dt from dual

)

SELECT report.dt,

nvl(ord.total, 0) total,

100 - nvl(SUM(ord.total)

OVER(PARTITION BY ord.pro\_cd ORDER BY report.dt), 0) AS remain

FROM report

left join(

select pro\_cd,

substr(ord\_dttm, 1, 6) as ord\_dttm,

count(\*) total

from tb\_ord

where pro\_cd = '00001'

group by pro\_cd,

substr(ord\_dttm, 1, 6)

) ord ON report.dt = ord.ord\_dttm;