Datos 1

2021

Objetivos

- Revisión ejercicio de transacción
- Introducción a clausula SELECT

Validar conceptos de ejercicio de transacciones

- Deadlock
- Autonomous transaction
- Lost update
- Optimistic locking
- Pesimistic locking
- Stateful application
- Stateles application

Basic SELECT Statement

```
SELECT *|{[DISTINCT]
column [alias],...} FROM
table;
```

- SELECT identifies the columns to be displayed.
- FROM identifies the table containing those columns.

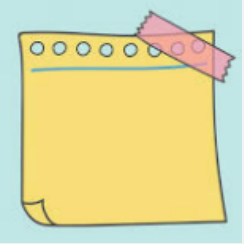
```
SELECT
department_id,
location_id
FROM departments;
```

	DEPARTMENT_ID	LOCATION_ID
1	10	1700
2	20	1800
3	50	1500
4	60	1400
5	80	2500
6	90	1700
7	110	1700
8	190	1700

```
SELECT * FROM departments;
```

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700

Writing SQL Statements



- SQL statements are not case-sensitive.
- SQL statements can be entered on one or more lines.
- Keywords cannot be abbreviated or split across lines.
- Clauses are usually placed on separate lines.
- Indents are used to enhance readability.
- In SQL Developer, SQL statements can be optionally terminated by a semicolon (;).
- Semicolons are required when you execute multiple SQL statements.
- In SQL*Plus, you are required to end each SQL statement with a semicolon (;).

Arithmetic Expressions

Create expressions with number and date data by using arithmetic operators (+, -, *, /).

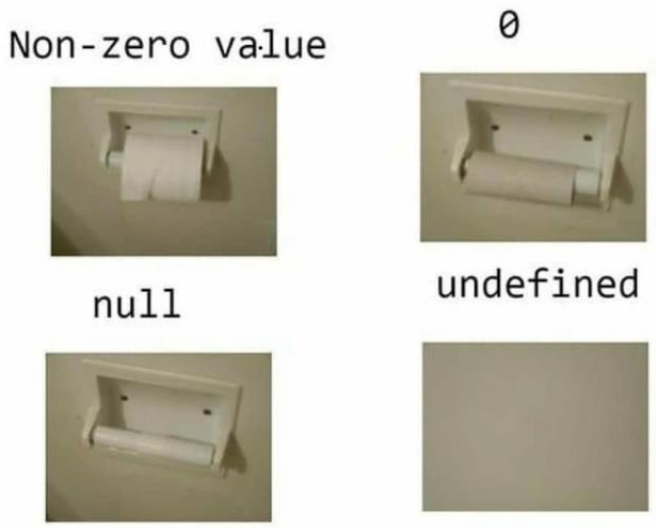
```
SELECT last_name, salary, 12*salary+100
FROM   employees;
```

	LAST_NAME	SALARY	12*SALARY+100
1	King	24000	288100
2	Kochhar	17000	204100
3	De Haan	17000	204100
4	Hunold	9000	108100

```
SELECT last_name, salary, 12*(salary+100)
FROM   employees;
```

	LAST_NAME	SALARY	12*(SALARY+100)
1	King	24000	289200
2	Kochhar	17000	205200
3	De Haan	17000	205200
4	Hunold	9000	109200

Defining a Null Value




```
SELECT last_name, job_id, salary,
commission_pct
FROM   employees;
```

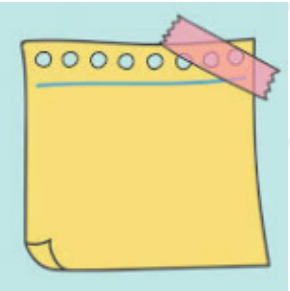
	LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
1	King	AD_PRES	24000	(null)
2	Kochhar	AD_VP	17000	(null)
3	De Haan	AD_VP	17000	(null)

...

12	Zlotkey	SA_MAN	10500	0.2
13	Abel	SA_REP	11000	0.3
14	Taylor	SA_REP	8600	0.2
15	Grant	SA_REP	7000	0.15

- Null is a value that is unavailable, unassigned, unknown, or inapplicable.
- Null is not the same as zero or a blank space.
- If any column value in an arithmetic expression is null, the result is null.
- If you attempt to perform division by zero, you get an error. If you divide a number by null, the result is a null or unknown.

Column Alias



- A column Alias renames a column heading
- Is useful with calculations
- Immediately follows the column name (there can also be the optional `AS` keyword between the column name and the alias)
- Requires double quotation marks if it contains spaces or special characters, or if it is case-sensitive

```
SELECT last_name "Name" ,
salary*12 "Annual Salary"
FROM   employees;
```

	Name	Annual Salary
1	King	288000
2	Kochhar	204000
3	De Haan	204000
4	Hunold	108000

Concatenation Operator

- Links columns or character strings to other columns
- Is represented by two vertical bars (| |)
- Creates a resultant column that is a character expression

```
SELECT last_name||job_id
AS "Employees"
FROM   employees;
```

	Employees
1	AbelSA_REP
2	DaviesST_CLERK
3	De HaanAD_VP
4	ErnstIT_PROG
5	FayMK_REP
6	GietzAC_ACCOUNT
7	GrantSA_REP
8	HartsteinMK_MAN

Literal Character Strings

- A literal is a character, a number, or a date that is included in the `SELECT` statement.
- Date and character literal values must be enclosed within single quotation marks.
- Each character string is output once for each row returned.

```
SELECT last_name ||' is a '||job_id  AS "Employee Details"
FROM   employees;
```

	Employee Details
1	Abel is a SA_REP
2	Davies is a ST_CLERK
3	De Haan is a AD_VP
4	Ernst is a IT_PROG
5	Fay is a MK_REP
6	Gietz is a AC_ACCOUNT
7	Grant is a SA_REP
8	Hartstein is a MK_MAN
9	Higgins is a AC_MGR
10	Hunold is a IT_PROG
11	King is a AD_PRES

Duplicate Rows

- The default display of queries is all rows, including duplicate rows.
- To eliminate duplicate rows in the result, include the `DISTINCT` keyword in the `SELECT` clause immediately after the `SELECT` keyword
- You can specify multiple columns after the `DISTINCT` qualifier. The `DISTINCT` qualifier affects all the selected columns, and the result is every distinct combination of the columns.

```
SELECT department_id
FROM   employees;
```

	DEPARTMENT_ID
1	90
2	90
3	90
4	60
5	60
6	60
7	50
8	50

```
SELECT DISTINCT department_id
FROM   employees;
```

	DEPARTMENT_ID
1	(null)
2	90
3	20
4	110
5	50
6	80
7	60
8	10

Quiz

Identify the `SELECT` statements that execute successfully.

a.
```
SELECT first_name, last_name, job_id, salary*12  AS Yearly Sal
FROM   employees;
```

b.
```
SELECT first_name, last_name, job_id, salary*12 "yearly sal"
FROM   employees;
```

c.
```
SELECT first_name, last_name, job_id, salary AS "yearly sal"
FROM   employees;
```

d.
```
SELECT first_name+last_name AS name, job_Id, salary*12 yearly sal
FROM   employees;
```

Group Functions

Group functions operate on sets of rows to give one result per group.

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	110	12000
5	110	8300
6	90	24000
7	90	17000
8	90	17000
9	60	9000
10	60	6000
...		
18	80	11000
19	80	8600
20	(null)	7000

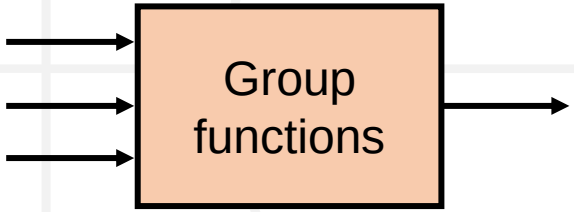
Maximum salary in `EMPLOYEES` table

MAX(SALARY)
24000

Group Functions

Types of groups functions

- AVG
- COUNT
- MAX
- MIN
- SUM
- LISTAGG
- STDDEV
- VARIANCE



Group Functions

Syntax of groups functions

```
SELECT
group_function([DISTINCT|ALL]column|expr), ...
FROM   table
[WHERE  condition];
```

- DISTINCT makes the function consider only nonduplicate values; ALL makes it consider every value, including duplicates. The default is ALL and, therefore, does not need to be specified.
- The data types for the functions with an expr argument may be CHAR, VARCHAR2, NUMBER, or DATE.
- All group functions ignore null values. To substitute a value for null values, use the NVL, NVL2, COALESCE, CASE, or DECODE functions.

Using Group Functions

```
SELECT AVG(salary), MAX(salary),MIN(salary), SUM(salary)
FROM   employees
WHERE  job_id LIKE '%REP%';
```

	AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
1	8150	11000	6000	32600

```
SELECT MIN(hire_date), MAX(hire_date)
FROM    employees;
```

	MIN(HIRE_DATE)	MAX(HIRE_DATE)
1	13-JAN-01	29-JAN-08

Using Group Functions

```
SELECT COUNT(*)
FROM   employees
WHERE  department_id = 50;
```

	COUNT(*)
1	5

```
SELECT COUNT(commission_pct)
FROM   employees
WHERE  department_id = 50;
```

	COUNT(COMMISSION_PCT)
1	0

```
SELECT COUNT(DISTINCT department_id)
FROM   employees;
```

	COUNT(DISTINCTDEPARTMENT_ID)
1	7

Groups of Data

	DEPARTMENT_ID	SALARY	
1	10	4400	4400
2	20	13000	9500
3	20	6000	
4	50	2500	3500
5	50	2600	
6	50	3100	
7	50	3500	
8	50	5800	
9	60	9000	6400
10	60	6000	
11	60	4200	
12	80	11000	10033
13	80	8600	
…			
18	110	8300	
19	110	12000	
20	(null)	7000	

Average salary in the `EMPLOYEES` table for each department

	DEPARTMENT_ID	AVG(SALARY)
1	(null)	7000
2	20	9500
3	90	19333.333333333333...
4	110	10150
5	50	3500
6	80	10033.333333333333...
7	10	4400
8	60	6400

GROUP BY …

Syntax of GROUP BY clause

```
SELECT      column, group_function(column)
FROM        table
[WHERE      condition]
[GROUP BY   group_by_expression]
[ORDER BY   column];
```

- If you include a group function in a `SELECT` clause, you cannot select individual column as well, *unless* the individual column appears in the `GROUP BY` clause. You receive an error message if you fail to include the column list in the `GROUP BY` clause.
- Using a `WHERE` clause, you can exclude rows before dividing them into groups.
- You can substitute *`column`* with an expression in the `SELECT` statement.
- You must include the *columns* in the `GROUP BY` clause.
- You cannot use a column alias in the `GROUP BY` clause.

Using GROUP BY clause

```
SELECT   department_id,AVG(salary)
FROM     employees
GROUP BY department_id ;
```

	DEPARTMENT_ID	AVG(SALARY)
1	(null)	7000
2	90	19333.333333333333333333333333333333
3	20	9500
4	110	10154
5	50	3500
6	80	10033.333333333333333333333333333333
7	60	6400
8	10	4400

```
SELECT   AVG(salary)
FROM     employees
GROUP BY department_id ;
```

	AVG(SALARY)
1	7000
2	19333.333333333333333333333333333333
3	9500
4	10154
5	3500
6	10033.333333333333333333333333333333
7	6400
8	4400

Grouping by more than one column

```
SELECT
department_id, job_id,
sum(salary)
FROM employees
GROUP BY
department_id, job_id
ORDER BY job_id;
```

	DEPARTMENT_ID	JOB_ID	SALARY
1	10	AD_ASST	4400
2	20	MK_MAN	13000
3	20	MK_REP	6000
4	50	ST_CLERK	2500
5	50	ST_CLERK	2600
6	50	ST_CLERK	3100
7	50	ST_CLERK	3500
8	50	ST_MAN	5800
9	60	IT_PROG	9000
10	60	IT_PROG	6000
11	60	IT_PROG	4200
12	80	SA_REP	11000
13	80	SA_REP	8600
14	80	SA_MAN	10500

...

19	110	AC_MGR	12000
20	(null)	SA_REP	7000

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	110	AC_ACCOUNT	8300
2	110	AC_MGR	12008
3	10	AD_ASST	4400
4	90	AD_PRES	24000
5	90	AD_VP	34000
6	60	IT_PROG	19200
7	20	MK_MAN	13000
8	20	MK_REP	6000
9	80	SA_MAN	10500
10	80	SA_REP	19600
11	(null)	SA_REP	7000
12	50	ST_CLERK	11700
13	50	ST_MAN	5800

Quiz - Identify valid SQL statements.

```
a) SELECT department_id, COUNT(last_name)
   FROM employees;

b) SELECT department_id, job_id, COUNT(last_name)
   FROM employees
   GROUP BY department_id;

c) SELECT   department_id, AVG(salary)
   FROM employees
   WHERE    AVG(salary) > 8000
   GROUP BY department_id;
```

Restricting Group Results

The maximum salary per department when it is greater than $10,000

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	50	2500
5	50	2600
6	50	3100
7	50	3500
8	50	5800
9	60	9000
10	60	6000
11	60	4200
12	80	11000
13	80	8600
...		
18	110	8300
19	110	12000
20	(null)	7000

	DEPARTMENT_ID	MAX(SALARY)
1	20	13000
2	90	24000
3	110	12000
4	80	11000

HAVING clause …

Syntax of HAVING clause

```
SELECT      column, group_function
FROM        table
[WHERE      condition]
[GROUP BY   group_by_expression]
[HAVING     group_condition]
[ORDER BY   column];
```

Note
The `WHERE` clause restricts rows, whereas the `HAVING` clause restricts groups.

Oracle Server Restricts rows as follows:

1. Rows are grouped.
2. The group function is applied.
3. Groups matching the `HAVING` clause are displayed.

Using HAVING clause

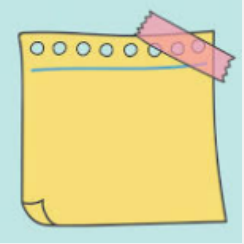
```
SELECT   department_id, MAX(salary)
FROM     employees
GROUP BY department_id
HAVING   MAX(salary)>10000 ;
```

	DEPARTMENT_ID	MAX(SALARY)
1	90	24000
2	20	13000
3	110	12008
4	80	11000

```
SELECT   job_id, SUM(salary) PAYROLL
FROM     employees
WHERE    job_id NOT LIKE '%REP%'
GROUP BY job_id
HAVING   SUM(salary) > 13000
ORDER BY SUM(salary);
```

	JOB_ID	PAYROLL
1	IT_PROG	19200
2	AD_PRES	24000
3	AD_VP	34000

Summary



- There are several group functions available in SQL
- You can create subgroups by using the `GROUP BY` clause. Further, groups can be restricted using the `HAVING` clause.
- Place the `HAVING` and `GROUP BY` clauses after the `WHERE` clause in a statement. The order of the `GROUP BY` and `HAVING` clauses following the `WHERE` clause is not important. You can have either the `GROUP BY` clause or the `HAVING` clause first as long as they follow the `WHERE` clause. Place the `ORDER BY` clause at the end.