

---

# DATABASE PROGRAMMING WITH SQL 1/2

---

ORACLE ACADEMY



27 DE OCTUBRE DE 2022  
UNIVERSIDAD POLITECNICA DE AGUASCALIENTES  
Juan Carlos Herrera Hernández

## Contenido

Section 1 – Introduction .....	3
1-2 Oracle Application Express.....	3
1-2 Relational Database Technology .....	3
1-3 Anatomy of a SQL Statement .....	5
Section 2 – SELECT and WHERE .....	6
2-1 Columns, Characters, and Rows .....	6
SELECT First_name    ' '    Last_name as "Complete Name" from employees; .....	6
2-2 Limit Rows Selected .....	6
2-3 Comparison Operators .....	6
Section 3 – WHERE, ORDER BY, and Intro to Functions .....	8
3-1 Logical Comparisons and Precedence Rules .....	8
3-2 Sorting Rows.....	8
3-3 Introduction to Functions.....	9
Section 4 – Single Row Functions Part I.....	10
4-1 Case and Character Manipulation.....	10
4-2 Number Functions.....	11
4-3 Date Functions.....	11
Section 5 – Single Row Functions Part II.....	12
5-1 Conversion Functions .....	12
5-2 NULL Functions.....	14
5-3 Conditional Expressions .....	15
Section 6 – JOINS Part I.....	16
6-1 Cross Joins and Natural Joins .....	16
6-2 Join Clauses .....	16
6-3 Inner versus Outer Joins.....	17
6-4 Self-Joins and Hierarchical Queries .....	17
Section 7 – JOINS Part II.....	19
7-1 Oracle Equijoin and Cartesian Product.....	19
7-2 Oracle Nonequijoins and Outer Joins.....	20

Section 8 – Group Functions Part I .....	21
8-1 Group Functions .....	21
8-2 COUNT, DISTINCT, NVL.....	21
Section 9 – Group Functions Part II .....	22
9-1 Using Group By and Having Clauses.....	22
9-2 Using Rollup and Cube Operations, and Grouping Sets .....	22
9-3 Using Set Operators .....	24
Section 10 – Subqueries .....	25
10-1 Fundamentals of Subqueries.....	25
10-2 Single-Row Subqueries: =,>,<,>=,<=,<> .....	26
10-3 Multiple-Row Subqueries (IN, ANY, ALL).....	26
10-4 Correlated Subqueries.....	27

## Section 1 – Introduction

### 1-2 Oracle Application Express

System software consists of low-level programs designed to interact with the computer hardware. Operating systems, compilers, and system utilities are examples of system software.

In contrast, application software includes programs for word processing, databases, gaming, email, and graphics.

Oracle Application Express (APEX) is the tool that we will use to allow you to build tables and retrieve information from an Oracle database.

Two components in Oracle Application Express are:

- SQL Workshop (to learn SQL)
- Application Builder (to design an application).

```
SELECT < * | [DISTINCT] column | expr [ AS alias], .....>
FROM <table name>
WHERE <condition>;
```

### 1-2 Relational Database Technology

Row	An entry in a table, consisting of values for each appropriate column. Data for one table instance.
Primary key	The set of mandatory columns within a table that is used to enforce uniqueness of rows and that is normally the most frequent means by which rows are accessed. Unique identifier for each row.
Table	An arrangement of data in rows and columns. basic storage structure.
Foreign key	A column or set of columns that refers to a primary key in the same table or another table.
Relational database	Collections of objects or relations, set of operators to act on those relations, and data integrity for accuracy and consistency
Field	Intersection of a row and column
Data manipulation language (DML)	Used to modify the table data by entering, changing, or removing rows
Data definition language (DDL)	Creates, changes, and removes data structures from the database
Transaction control (TCL)	Used to manage the changes made by DML statements
Data control language (DCL)	Used to give or remove access rights to the database and the structures within it

## Categories SQL Statements

CREATE  
ALTER  
DROP  
RENAME  
TRUNCATE  
COMMENT

Data definition language (DDL)

SELECT  
INSERT  
UPDATE  
DELETE  
MERGE

Data manipulation language (DML)

GRANT  
REVOKE

Data control language (DCL)

COMMIT  
ROLLBACK  
SAVEPOINT

Transaction control language (TCL)

### Properties of Tables

There are six properties of tables in a relational database:

Property 1: Entries in columns are single-valued

Property 2: Entries in columns are of the same kind

Property 3: Each row is unique

Property 4: Sequence of columns is insignificant

Property 5: Sequence of rows is insignificant

Property 6: Each column has a unique name

### Types of Data

MySQL	ORACLE	Description
VARCHAR(SIZE)	VARCHAR2(size)	Variable-length character data
CHAR(SIZE)	CHAR(size)	Fixed-length character data
INT[EGER]	NUMBER	Variable-length numeric data
Decimal(p,s)	NUMBER(p,s)	Float
DATE	DATE	Date and time values
TINYINT(1)	NUMBER(1,0)	boolean

### 1-3 Anatomy of a SQL Statement

Join	Display data from two or more related tables
Arithmetic operator	A symbol used to perform an operation on some values.
Column	An implementation of an attribute or relationship in a table
Projection	The capability in SQL to choose the <u>columns</u> in a table that you want returned from a query
NULL	A value that is unavailable, unassigned, <u>unknown</u> , or inapplicable. NULL is not the same as a zero or a space. Example: Null * 5 = Null
Column alias	Renames a column heading
Arithmetic expression	A mathematical equation
Selection	The capability in SQL to choose the <u>rows</u> in a table returned from a query
Select statement	Retrieves information from the database
Select clause	Specifies the columns to be displayed
From clause	Specifies the table containing the column listed in the select clause
Keyword	An individual SQL command
Clause	Part of a SQL statement
Statement	A combination of the two clauses

- double quotes in columns and tables.

```
SELECT department_id, first_name nombre, salary*12 "Annual Salary"
FROM employees
where job_id = 'SA_REP'
and department_id is not null;
```

## Section 2 – SELECT and WHERE

### 2-1 Columns, Characters, and Rows

DISTINCT	A command that suppresses duplicates. SELECT DISTINCT department_id FROM employees;  SELECT department_id FROM employees group by department_id;
Concatenation	Links two columns together to form one character data column SELECT First_name    ' '    Last_name as "Complete Name" from employees;
String	A group of character data
DESCRIBE (DESC)	An SQL plus command that displays the structure of a table DESC <table_name>; Describe Employees;

### 2-2 Limit Rows Selected

WHERE clause	Restricts the rows returned by a select statement
Comparison conditions	Compares one expression to another value or expression

Equal to =	select sysdate from dual; -- Server	SELECT * from employees
Not equal to <> , !=	select current_date from dual; -- Client	WHERE hire_date < '01/1/1990'; WHERE hire_date < '01-Jan-1990';

### 2-3 Comparison Operators

ESCAPE	This option identifies that the escape characters should be interpreted literally
IS NULL	Condition tests for null values. Unknown value. Example: Where department_id is not null;
BETWEEN	Displays rows based on a range of values
Inclusive	Including the specified limits and the area between them; the numbers 1-10, inclusive
LIKE	Selects rows that match a character pattern using wildcard characters
IN	Tests for values in a specified list of values
wildcard characters % _	can be used to construct a search string. % is used to represent any sequence of zero or more characters. _ is used to represent a single character

SELECT last_name, salary FROM employees WHERE salary BETWEEN 9000 AND 11000; WHERE salary >= 9000 AND salary <=11000;	SELECT city, state_province, country_id FROM locations WHERE country_id IN ('UK', 'CA'); WHERE country_id = 'UK' OR country_id = 'CA';
--	---

we would need to use an escape character to say we are searching for an underscore, and not just any one character.

Example: Show job\_id containing \_R

ORACLE	MySQL
<pre>select employee_id, job_id from employees where job_id like '%\_%' ESCAPE '\';</pre>	<pre>select employee_id, job_id from employees where job_id like '%\_P%';</pre>



## Section 3 – WHERE, ORDER BY, and Intro to Functions

### 3-1 Logical Comparisons and Precedence Rules

NOT highest	Inverts the value of the condition
AND	Both conditions must be true for a record to be selected
OR lowest	Either condition can be true for a record to be selected
Precedence	Rules that determine the order in which expressions are evaluated and calculated

SELECT manager_id, location_id FROM departments WHERE manager_id=124 or location_id= 2500;	SELECT manager_id, location_id FROM departments WHERE manager_id=124 and location_id= 1500;
--	---

SELECT department_id, first_name, last_name FROM employees WHERE department_id IN(60,80) OR first_name LIKE 'C%' AND last_name LIKE '%s%';	SELECT department_id, first_name, last_name FROM employees WHERE (department_id IN (60,80) OR first_name LIKE 'C%') AND last_name LIKE '%s%';
--	---

### 3-2 Sorting Rows

<b>Ascending / ASC</b>	Orders the rows in ascending order (the default order); A-Z
<b>Descending / DESC</b>	Orders the rows in descending order: Z-A
<b>Sort</b>	To arrange according to class, kind, or size
	Null values are displayed last in ascending order and first in descending order ORDER BY department_id DESC, last_name;

### 3-3 Introduction to Functions

Single-row functions: affects only one row and return one result per row



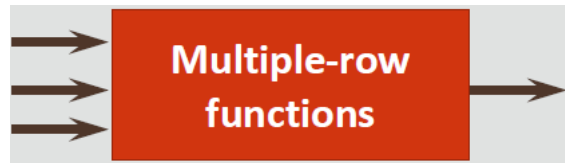
Number Functions:

Character Functions: upper, substr

Date Functions:

Conversion Functions:

Multiple-row functions: affects a set or group of rows and Return one result per group of rows, are known as group functions



Count

Sum

Avg: finds the average value in a group of rows

Min

Max

## Section 4 – Single Row Functions Part I

### 4-1 Case and Character Manipulation

<b>DUAL</b>	Dummy table used to view results from functions and calculations. The DUAL table has one row called "X" and one column called "DUMMY"
<b>Single-row functions</b>	Functions that operate on single rows only and return one result per row
<b>Character functions</b>	Functions that accept character data as input and can return both character and numeric values
<b>Input</b>	Raw data entered into the computer  Sustitution Variables: SELECT first_name, last_name, salary, department_id FROM employees WHERE department_id = &enter_dept_id; -- WHERE department_id=:enter_dept_id;
<b>Output</b>	Data that is processed into information
<b>Format</b>	The arrangement of data for storage or display
<b>Expression</b>	A symbol that represents a quantity or a relationship between quantities
<b>INITCAP</b>	Converts alpha character values to uppercase for the first letter of each word, all other letters in lowercase.
<b>UPPER</b>	Converts alpha characters to upper case
<b>LOWER</b>	Converts alpha character values to lowercase
<b>TRIM</b>	Removes all specified characters from either the beginning or the ending of a string. (Leading, Trailing, Both) SELECT TRIM(both 'a' FROM 'abcba') FROM DUAL; select trim(' hola amigos ') from dual;
<b>CONCAT</b>	Concatenates the first character value to the second character value; equivalent to concatenation operator (  ).
<b>LPAD</b>	Pads the left side of a character, resulting in a right-justified value
<b>SUBSTR</b>	Returns specific characters from character value starting at a specific character <u>position</u> and going specified character <u>positions long</u>
<b>REPLACE</b>	Replaces a sequence of characters in a string with another set of characters. SELECT REPLACE('JACK and JUE', 'J', 'BL') FROM DUAL;
<b>INSTR</b>	Returns the numeric position of a named string.
<b>LENGTH</b>	Returns the number of characters in the expression
<b>RPAD</b>	Pads the right-hand side of a character, resulting in a left- justified value.

#### 4-2 Number Functions

<b>Number functions</b>	These functions accept numeric input and return numeric values.
<b>TRUNC</b>	Used to terminate the column, expression, or value to a specified number of decimal places
<b>MOD</b>	Returns the remainder of a division. Even or odd (par o impar)
<b>ROUND</b>	Rounds the column, expression, or value to a set number of decimal places. ROUND(column   expression, decimal places) ROUND(45.926, 0)      46 ROUND(45.926, -1)    50

#### 4-3 Date Functions

<b>SYSDATE</b>	A function that returns the current date and time of the database server. DD-Mon-YYYY
<b>ADD_MONTHS</b>	Add calendar months to date
<b>LAST_DAY</b>	Last day of the month
<b>NEXT_DAY</b>	Next day of the date specified
<b>MONTHS_BETWEEN</b>	Number of months between due dates
<b>Year, month, day</b>	Number of the: Subtracting two dates returns a number

Tools -> Preferences -> Database -> NLS

```
select * from nls_session_parameters;
```

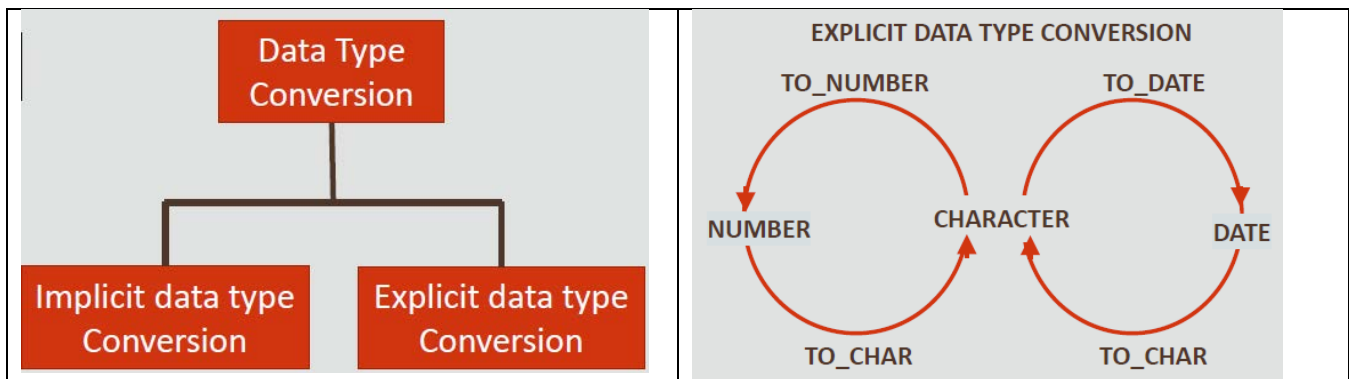
```
alter session set nls_language = 'AMERICAN';
```

ORACLE	MySQL
select extract(year from sysdate) from dual;	select year(sysdate()) from dual;
select to_char(sysdate,'D') from dual; -- (1=dom 2=lun 7=Sab)	

## Section 5 – Single Row Functions Part II

### 5-1 Conversion Functions

<b>CHAR</b>	Used for text and character data of fixed length, including numbers, dashes, and special characters.
<b>fm</b>	Used to remove padded blanks or to suppress leading zeros
<b>Conversion function</b>	Functions that convert a value from one datatype to another.
<b>NUMBER</b>	Used to store variable-length numeric data.
<b>VARCHAR2</b>	Used for character data of variable length, including numbers, special characters, and dashes.
<b>DATE</b>	Used for date and time values. DD-Mon-YYYY (for example, 23-Oct-2013)
<b>TO_CHAR</b>	Converts dates or numbers to character strings with optional formatting
<b>RR date format</b>	Century value depends on the specified year and the last two digits of the current year
<b>TO_NUMBER</b>	Converts a character string containing digits to a number with optional formatting
<b>DD date format</b>	Numeric day of the month
<b>TO_DATE</b>	Converts a character string representing a date to a date value with optional formatting



Use an **fm** element to remove padded blanks or remove leading zeroes from the output

Use **sp** to spell out a number

Use **th** to have the number appear as an ordinal – (1st, 2nd, 3rd, and so on)

### Date Conversion to Character Data

TO\_CHAR (date column name, 'format model you specify')

YYYY	Full year in numbers
YEAR	Year spelled out
MM	Two-digit value for month
MONTH	Full name of the month
MON	Three-letter abbreviation of the month
DY	Three-letter abbreviation of the day of the week
DAY	Full name of the day of the week
DD	Numeric day of the month
DDspth	FOURTEENTH
Ddspth	Fourteenth
ddspth	fourteenth
DDD or DD or D	Day of year, month or week
HH24:MI:SS AM	15:45:32 PM
DD "of" MONTH	12 of October

```
SELECT TO_CHAR(hire_date, 'fmMonth ddth, YYYY')
FROM employees;
```

### Number Conversion to Character Data

TO\_CHAR(number, 'format model')

ELEMENT	DESCRIPTION	EXAMPLE	RESULT
9	Numeric position (# of 9's determine width)	999999	1234
0	Display leading zeros	099999	001234
\$	Floating dollar sign	\$999999	\$1234
L	Floating local currency symbol	L999999	FF1234
.	Decimal point in position specified	999999.99	1234.00
,	Comma in position specified	999,999	1,234
MI	Minus signs to right (negative values)	999999MI	1234-
PR	Parenthesize negative numbers	999999PR	<1234>
EEEE	Scientific notation ( must have four EEEE)	99.999EEEE	1,23E+03
V	Multiply by 10 n times (n= number of 9's after V)	9999V99	9999V99
B	Display zero values as blank, not 0	B9999.99	1234.00

```
SELECT TO_CHAR(salary, '$99,999') AS "Salary"
FROM employees;
```

select sysdate, to_char(sysdate, 'fmMonth DD, RRRR') from dual;
SELECT trunc(date '2023-07-20' - SYSDATE, 1) dias FROM DUAL;
SELECT round(SYSDATE - TO_DATE('20-JUL-2023', 'DD/MON/YYYY'),1) dias FROM DUAL;

### Character Conversion to Number

TO\_NUMBER(character string, 'format model')

```
SELECT TO_NUMBER('5,320', '9,999') AS "Number" FROM dual;
```

## Character Conversion to Date

TO\_DATE('character string', 'format model')

```
Select TO_DATE('November 3, 2001', 'Month dd, yyyy') from dual;
SELECT TO_DATE('27-Oct-95', 'DD-Mon-YY') AS "Date" FROM dual;
```

to_char(hire_date, 'RR')		If the specified two-digit year is <b>RR</b> :	
		0-49	50-99
If the two first digits of the <b>current year</b> are:	0-49	The return date is in the current century =	The return date is in the century before the current one -
	50-99	The return date is in the century after the current one +	The return date is in the current century =

The default date display format is DD-MON-RR.

```
alter session set nls_date_format='dd-mm-yyyy';
```

```
SELECT
  TO_CHAR(TO_DATE('20-Dec-81', 'DD-Mon-RR'), 'YYYY') AS "RR",
  TO_CHAR(TO_DATE('20-Dec-81', 'DD-Mon-YY'), 'YYYY') AS "YY"
FROM DUAL;
Result:
  RR   YY
1981  2081
```

## 5-2 NULL Functions

<b>NVL</b>	<p>Converts nulls to an actual value</p> <p>Sintaxis: NVL(test_value, if_null)</p> <p>Example: NVL(commission_pct,0)</p> <p>MySQL: SELECT NVL(commission_pct,0), <b>IFNULL</b>(commission_pct,0) FROM EMPLOYEES;</p>
<b>NVL2</b>	<p>Examines the first expression; if the first expression is not null, it returns the second expression; if the first expression is null, it returns the third expression</p> <p>Sintaxis: NVL2(test_value, if_not_null, if_null)</p> <p>Example: NVL2(commission_pct, 'SAL+COMM', 'SAL')</p>
<b>COALESCE</b>	<p>Returns the first non-null expression in the list</p> <p>Sintaxis: COALESCE (expr1, expr2, ..., exprn)</p> <p>Example: COALESCE(salary+commission_pct*salary, salary+2000)</p>
<b>NULLIF</b>	<p>Compares two expressions; if they are equal, the function returns null; if they are not equal, the function returns the first expression</p> <p>Sintaxis: NULLIF (Expr1, expr2)</p> <p>Example: NULLIF(LENGTH(first_name), LENGTH(last_name))</p>

### 5-3 Conditional Expressions

<b>Conditional expression</b>	An if-then-else expression whose value depends on the truth-value of a Boolean expression.
<b>CASE</b>	<p>Implements conditional processing within a SQL statement; it meets the ANSI standard.</p> <pre> CASE expr WHEN comparison_expr1 THEN return_expr1         [WHEN comparison_expr2 THEN return_expr2         WHEN comparison_exprn THEN return_exprn         ELSE else_expr]  END  SELECT last_name, job_id, salary,        CASE WHEN job_id = 'IT_PROG' THEN 1.10*salary             WHEN job_id = 'ST_CLERK' THEN 1.15*salary             WHEN job_id = 'SA_REP' THEN 1.20*salary             ELSE salary        END "REVISED_SALARY" FROM employees;  select EXTRACT(YEAR FROM hire_date) anio,        sum(case TO_CHAR(hire_date, 'MM') when 1 then 1 ELSE 0 end) "Ene",        sum(case when TO_CHAR(hire_date, 'MM') = '02' then 1 ELSE 0 end) Feb,        sum(case when TO_CHAR(hire_date, 'MM') &gt; 2 then 1 ELSE 0 end) Resto,        count(*) total from employees group by EXTRACT(YEAR FROM hire_date); </pre>
<b>DECODE (Only Oracle)</b>	<p>Compares an expression to each of the search values</p> <pre> DECODE(column1 expression, search1, result1         [, search2, result2,...,]         [, default])  DECODE(job_id, 'IT_PROG', 1.10*salary,             'ST_CLERK', 1.15*salary,             'SA_REP', 1.20*salary,             salary) </pre>
<b>IF (Only MySQL)</b>	<p>IF(condition, true_value, false_value)</p> <pre> SELECT sysdate(),        if(MONTH(sysdate())= 1, 1, 0) as Ene FROM dual; </pre>



## Section 6 – JOINS Part I

### 6-1 Cross Joins and Natural Joins

<b>CROSS JOIN</b>	Returns the Cartesian product from two tables.
<b>NATURAL JOIN</b>	Joins two tables based on the same column name. The names and data types of both columns must be the same.

<b>NATURAL JOIN</b>	<b>CROSS JOIN</b>
SELECT last_name, job_id, job_title FROM employees NATURAL JOIN jobs WHERE department_id > 80;	SELECT last_name, department_name FROM employees CROSS JOIN departments; -- 20 X 8 = 160 rows

Is it possible to apply a NATURAL JOIN with Employees and Departments?

### 6-2 Join Clauses

<b>ON clause</b>	Allows a natural join based on an arbitrary condition or two columns with different names.
<b>USING clause</b>	Performs an equijoin based on one specified column name. Don't use an alias in this column

<b>ON clause</b>	<b>USING clause</b>
SELECT last_name, j.job_id, job_title FROM employees e JOIN jobs j ON (e.job_id = j.job_id) WHERE last_name LIKE 'H%';	SELECT last_name, job_id, job_title FROM employees e JOIN jobs j using (job_id) WHERE last_name LIKE 'H%';

ON Clause with <b>non-equality</b> operator
SELECT last_name, salary, grade_level, lowest_sal, highest_sal FROM employees JOIN job_grades ON (salary BETWEEN lowest_sal AND highest_sal);

Joining <b>Three</b> Tables
SELECT last_name, department_name "Department", city FROM employees JOIN departments USING (department_id) JOIN locations USING (location_id);

### 6-3 Inner versus Outer Joins

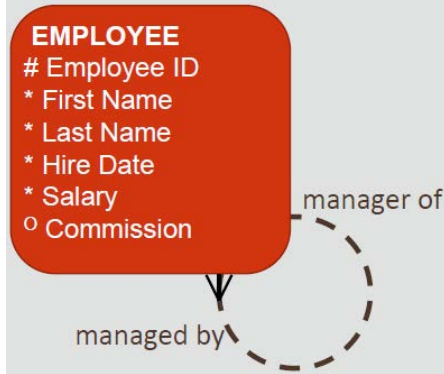
<b>FULL OUTER JOIN</b>	Performs a join on two tables, retrieves all the rows in the Left table, even if there is no match in the Right table. It also retrieves all the rows in the Right table, even if there is no match in the Left table.
<b>Outer join</b>	A join that returns the unmatched rows as well as matched rows
<b>LEFT OUTER JOIN</b>	Performs a join on two tables, retrieves all the rows in the Left table even if there is no match in the Right table
<b>RIGHT OUTER JOIN</b>	Performs a join on two tables, retrieves all the rows in the Right table even if there is no match in the Left table
<b>Inner join</b>	A join of two or more tables that returns only matched rows

LEFT OUTER JOIN	RIGHT OUTER JOIN
<pre>SELECT e.last_name, d.department_id,        d.department_name FROM employees e <b>LEFT</b> OUTER JOIN departments d   ON (e.department_id= d.department_id);</pre>	<pre>SELECT e.last_name, d.department_id,        d.department_name FROM employees e <b>RIGHT</b> OUTER JOIN departments d   ON e.department_id= d.department_id;</pre>
<pre>SELECT e.last_name, d.department_id,        d.department_name FROM employees e <b>FULL</b> OUTER JOIN departments d   ON (e.department_id= d.department_id);</pre>	

### 6-4 Self-Joins and Hierarchical Queries

<b>Self join</b>	Joins a table to itself
<b>Hierarchical Query</b>	Retrieves data based on a natural hierarchical relationship between rows in a table
<b>Level</b>	Determines the number of steps down from the beginning row that should be returned by a hierarchical query
<b>Start with</b>	Identifies the beginning row for a hierarchical query
<b>Connect By Prior</b>	Specifies the relationship between parent rows and child rows of a hierarchical query

An employee can also be a manager.



Workers ( N = 20)

EMPLOYEES (worker)		
employee_id	last_name	manager_id
100	King	
101	Kochhar	100
102	De Haan	100
103	Hunold	102
104	Ernst	103
107	Lorentz	103
124	Mourgos	100

Manager ( N= 8)

EMPLOYEES (manager)	
employee_id	last_name
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst
107	Lorentz
124	Mourgos

```
select count(*) from employees;
select count(distinct manager_id) from employees;
```

With hierarchical queries, we can also see who that manager works for, and so on

```
SELECT worker.Employee_id Trabajador,
       manager.employee_id Manager
FROM employees worker JOIN employees manager
  ON worker.manager_id = manager.employee_id
order by 2, 1;
```

```
select level, employee_id Emp_ID,last_name "Empleado",
       prior employee_id Mng_ID , prior last_name "Manager",
       lpad(employee_id, length(employee_id)+(level*2)-2,'-') as "Org Chart"
from employees
start with employee_id = 100
connect by prior employee_id = manager_id;
```

LEVEL	EMP_ID	Empleado	MNG_ID	Manager	Org Chart
1	1	100 King	(null)	(null)	100
2	2	101 Kochhar	100 King		--101
3	3	200 Whalen	101 Kochhar		----200
4	3	205 Higgins	101 Kochhar		----205
5	4	206 Gietz	205 Higgins		-----206
6	2	102 De Haan	100 King		--102
7	3	103 Hunold	102 De Haan		----103
8	4	104 Ernst	103 Hunold		-----104
9	4	107 Lorentz	103 Hunold		-----107
10	2	124 Mourgos	100 King		--124
11	3	141 Rais	124 Mourgos		----141
12	3	142 Davies	124 Mourgos		----142
13	3	143 Matos	124 Mourgos		----143
14	3	144 Vargas	124 Mourgos		----144
15	2	149 Zlotkey	100 King		--149
16	3	174 Abel	149 Zlotkey		----174
17	3	176 Taylor	149 Zlotkey		----176
18	3	178 Grant	149 Zlotkey		----178
19	2	201 Hartstein	100 King		--201
20	3	202 Fav	201 Hartstein		----202

## Section 7 – JOINS Part II

### 7-1 Oracle Equijoin and Cartesian Product

<b>Cartesian product</b>	Results from an invalid or omitted join condition; all combinations of rows are displayed
<b>Equijoin</b>	Values in a column in one table are equal to a value in another table; also called an inner join or simple join
<b>Proprietary join</b>	Connection command exclusive to a specific company
<b>Alias</b>	Gives a table another name to simplify queries and improve performance
<b>Join conditions</b>	Display data from two or more related tables
<b>Oracle Proprietary Join</b>	<b>ANSI/ISO SQL: 1999 Equivalent</b>
Cartesian Product	Cross Join
Equijoin	NATURAL JOIN  JOIN USING clause  JOIN ON clause (if the equality operator is used)
Non-equijoin	ON clause

Without JOIN (Oracle Proprietary Join)	Cross Join (Cartesian Product)
<pre>SELECT last_name, e.job_id, job_title FROM employees e, jobs j WHERE e.job_id      = j.job_id       AND department_id = 80;</pre>	<pre>SELECT e.last_name, d.department_name FROM employees e, departments d;</pre>

join three tables
<pre>SELECT last_name, city FROM employees e, departments d, locations l WHERE e.department_id = d.department_id       AND d.location_id  = l.location_id;</pre>

## 7-2 Oracle Nonequijoins and Outer Joins

Nonequijoin
<pre>SELECT last_name, salary, grade_level, lowest_sal, highest_sal FROM employees, job_grades WHERE (salary BETWEEN lowest_sal AND highest_sal);</pre>

ANSI/ISO SQL	Only Oracle Syntax
<pre>SELECT e.last_name, d.department_name FROM employees e LEFT JOIN departments d ON e.department_id = d.department_id;</pre>	<pre>SELECT e.last_name, d.department_name FROM employees e, departments d WHERE e.department_id = d.department_id(+);</pre>
<pre>SELECT e.last_name, d.department_name FROM employees e RIGHT JOIN departments d ON e.department_id = d.department_id;</pre>	<pre>SELECT e.last_name, d.department_name FROM employees e, departments d WHERE e.department_id(+) = d.department_id;</pre>
<pre>(ORACLE only) SELECT e.last_name, d.department_name FROM employees e FULL JOIN departments d ON e.department_id = d.department_id;</pre>	No direct equivalent.

## Section 8 – Group Functions Part I

### 8-1 Group Functions

<b>COUNT</b>	Returns the number of rows with non-null values for the expression Count(*) vs count(department_id)
<b>SUM</b>	Calculates the sum ignoring null values
<b>AVG</b>	Calculates average value excluding nulls
<b>MIN</b>	Returns minimum value with any data type ignoring nulls
<b>MAX</b>	Returns the maximum value with any data type ignoring nulls
<b>STDDEV</b>	For two sets of data with approximately the same mean, the greater the spread, the greater the standard deviation.
<b>VARIANCE</b>	Used with columns that store numeric data to calculate the spread of data around the mean
<b>Group functions</b>	Operate on sets of rows to give one result per group. Group functions cannot be used in the WHERE clause Group functions ignore NULL values
<b>Aggregate</b>	To gather into a sum or whole

<pre>SELECT group_function(column), .. FROM table WHERE condition;</pre>	<pre>SELECT column, group_function(column), .. FROM table WHERE condition GROUP BY column;</pre>
--	--

### 8-2 COUNT, DISTINCT, NVL

<b>COUNT(*)</b>	Returns the number of rows in a table
<b>COUNT(expression)</b>	Returns the number of non-null values in the expression column
<b>DISTINCT</b>	The keyword used to return only nonduplicate values or combinations of nonduplicate values in a query.
<b>COUNT(DISTINCT expression)</b>	Returns the number of unique non-null values in the expression column.

Select count(*) from employees	Select count(department_id) from employees
Select department_id from employees	Select distinct department_id from employees
Select count(nvl(department_id,0)) from employees	Select count( distinct department_id) from employees

SELECT AVG(commission_pct) FROM employees;	SELECT AVG(NVL(commission_pct, 0)) FROM employees;
--	--

## Section 9 – Group Functions Part II

### 9-1 Using Group By and Having Clauses

<b>GROUP BY</b>	<p>Divides the rows in a table into groups</p> <p>Group functions require that any column listed in the SELECT clause that is not part of a group function must be listed in a GROUP BY clause.</p> <p>You cannot use a column alias in the GROUP BY clause</p>
<b>HAVING</b>	Used to specify which groups are to be displayed; restricts groups that do not meet group criteria
	<p>If there is having, then there is a group.</p> <p>If there is group, does not necessarily having</p>
<b>WHERE</b>	excludes rows <b>before</b> they are divided into groups

Correct	Find the Error
<pre>SELECT department_id, MAX(salary) FROM employees WHERE last_name!= 'King' GROUP BY department_id HAVING COUNT(*) &gt; 1 order by department_id;</pre>	<pre>SELECT department_id, job_id, AVG(salary) FROM employees GROUP BY department_id order by 1;</pre>
<pre>( ORACLE Only) SELECT max(avg(salary)) FROM employees GROUP by department_id;</pre>	<pre>SELECT department_id, MAX(salary) FROM employees WHERE COUNT(*) &gt; 1 GROUP BY department_id;</pre>

### 9-2 Using Rollup and Cube Operations, and Grouping Sets

<b>ROLLUP</b>	Used to create <b>subtotals</b> that roll up from the most detailed level to a grand total, following a grouping list specified in the clause
<b>CUBE</b>	An extension to the GROUP BY clause like ROLLUP that produces <b>cross-tabulation</b> reports
<b>GROUPING SETS</b>	<p>Used to specify multiple groupings of data</p> <p>Use GROUPING SETS to produce a single result set.</p> <p>Use the GROUPING function to identify the extra row values created by either a ROLLUP or CUBE operation.</p>

ORACLE	MySQL
SELECT department_id, job_id, SUM(salary) FROM employees where department_id is not null GROUP BY <b>ROLLUP</b> (department_id, job_id) order by department_id, job_id ;	SELECT department_id, job_id, SUM(salary) FROM employees where department_id is not null GROUP BY department_id, job_id with rollup
SELECT department_id, job_id, SUM(salary) FROM employees where department_id is not null GROUP BY <b>CUBE</b> (department_id, job_id) order by 1, 2 ;	does not exist

Grouping Set (ORACLE Only)
SELECT department_id, job_id, manager_id, SUM(salary) FROM employees WHERE department_id < 50 GROUP BY GROUPING SETS ( (job_id, manager_id), (department_id, job_id), (department_id, manager_id) );
Grouping Functions (ORACLE Only)
SELECT department_id, job_id, SUM(salary), <b>GROUPING</b> (department_id) AS "Dept sub total", -- Return 1 if it is null <b>GROUPING</b> (job_id) AS "Job sub total" -- Return 0 if it has a value FROM employees WHERE department_id < 50 GROUP BY CUBE (department_id, job_id);
SELECT department_name AS "Department Name", job_id AS "Job Title", SUM(salary) AS "Monthly Cost", <b>DECODE</b> (GROUPING(department_name), 0, 'Yes', 'No') AS "Department Used", <b>CASE</b> WHEN GROUPING(job_id)= 0 THEN 'Yes' ELSE 'No' END AS "Job Used" FROM employees JOIN departments using(department_id) GROUP BY CUBE(department_name, job_id) ORDER BY department_name;

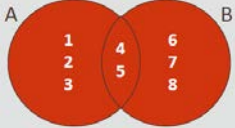

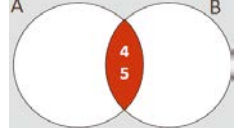
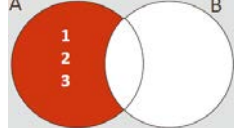


### 9-3 Using Set Operators

<b>SET operators</b>	used to combine results into one single result from multiple SELECT statements
<b>UNION</b>	operator that returns all rows from both tables and eliminates duplicates
<b>UNION ALL</b>	operator that returns all rows from both tables, including duplicates
<b>INTERSECT</b>	operator that returns rows common to both tables
<b>MINUS</b>	operator that returns rows that are unique to each table
<b>TO_CHAR(null)</b>	columns that were made up to match queries in another table that are not in both tables

Rules to remember when using SET operators:

- The number of columns and the data types of the columns must be identical in all of the SELECT statements used in the query
- The names of the columns need not be identical
- Column names in the output are taken from the column names in the first SELECT statement

	UNION	UNION ALL	INTERSECT	MINUS
SELECT a_id, C FROM a UNION SELECT b_id, D FROM b;	{1, 2, 3, 4, 5, 6, 7, 8} 	{1, 2, 3, 4, 5, 4, 5, 6, 7, 8} 	{4, 5} 	{1, 2, 3} 

SELECT hire_date, employee_id, job_id FROM employees WHERE employee_id = 200 UNION SELECT TO_DATE(NULL), employee_id, job_id FROM job_history WHERE employee_id = 200 ORDER BY hire_date;
--

## Section 10 – Subqueries

### 10-1 Fundamentals of Subqueries

<b>Subquery</b>	An inner query that is nested within an outer query
<b>Inner query</b>	Another name for a subquery A subquery executes once before the main query
<b>Outer query</b>	It accepts a value from the inner query to complete its SELECT statement.
<b>Pair- wise multiple column subquery</b>	An inner query that compares multiple columns at the same time
<b>Non-pair-wise multiple column subquery</b>	An inner query that compares the multiple columns one at a time in different subqueries
<b>Two Types of Subqueries:</b>	
<b>Single-row subquery ( =, &lt;&gt;, !=, &lt;, &gt;)</b>	An inner query that returns only one row to the outer query
<b>Multiple-row subquery (IN, ANY, ALL)</b>	An inner query that returns one or more rows to the outer query

Subqueries can be placed in a number of SQL clauses, including the **WHERE** clause, the **HAVING** clause, and the **FROM** clause. The **SELECT** statement in **parentheses** is the inner query or 'subquery'. It executes first, before the outer query. A subquery cannot have its own **ORDER BY** clause, it must be the last clause in the main **SELECT** statement.

<pre>SELECT select_list FROM table WHERE expression operator       (SELECT select_list        FROM table);</pre>	<pre>SELECT select_list FROM table1, (SELECT select_list               FROM table2) alias Where table1.key = alias.key</pre>
<pre>SELECT last_name, job_id, d.department_id FROM employees e inner join departments d   on e.department_id = d.department_id where department_name= 'Marketing' ORDER BY job_id;</pre>	<pre>SELECT last_name, job_id, department_id FROM employees WHERE department_id=       (SELECT department_id FROM departments        WHERE department_name= 'Marketing') ORDER BY job_id;</pre>
	<pre>SELECT last_name, hire_date FROM employees WHERE hire_date&gt;       (SELECT hire_date        FROM employees        WHERE last_name= 'Vargas');</pre>

## 10-2 Single-Row Subqueries: =,>,<,>=,<=,<>

Example: Which departments have the highest salary that is greater than the highest salary in department 60?

SELECT max(salary) FROM employees WHERE department_id= 60	<table><tr><th></th><th>MAX(SALARY)</th></tr><tr><td>1</td><td>9000</td></tr></table>		MAX(SALARY)	1	9000																							
	MAX(SALARY)																											
1	9000																											
SELECT department_id, max(salary) FROM employees GROUP BY department_id order by max(salary)	<table><tr><th></th><th>DEPARTMENT_ID</th><th>MAX(SALARY)</th></tr><tr><td>1</td><td>10</td><td>4400</td></tr><tr><td>2</td><td>50</td><td>5800</td></tr><tr><td>3</td><td>(null)</td><td>7000</td></tr><tr><td>4</td><td>60</td><td>9000</td></tr><tr><td>5</td><td>80</td><td>11000</td></tr><tr><td>6</td><td>110</td><td>12000</td></tr><tr><td>7</td><td>20</td><td>13000</td></tr><tr><td>8</td><td>90</td><td>24000</td></tr></table>		DEPARTMENT_ID	MAX(SALARY)	1	10	4400	2	50	5800	3	(null)	7000	4	60	9000	5	80	11000	6	110	12000	7	20	13000	8	90	24000
	DEPARTMENT_ID	MAX(SALARY)																										
1	10	4400																										
2	50	5800																										
3	(null)	7000																										
4	60	9000																										
5	80	11000																										
6	110	12000																										
7	20	13000																										
8	90	24000																										
SELECT department_id, max(salary) FROM employees GROUP BY department_id HAVING max(salary) > (SELECT max(salary) FROM employees WHERE department_id = 60) order by max(salary);	<table><tr><th></th><th>DEPARTMENT_ID</th><th>MAX(SALARY)</th></tr><tr><td>1</td><td>80</td><td>11000</td></tr><tr><td>2</td><td>110</td><td>12000</td></tr><tr><td>3</td><td>20</td><td>13000</td></tr><tr><td>4</td><td>90</td><td>24000</td></tr></table>		DEPARTMENT_ID	MAX(SALARY)	1	80	11000	2	110	12000	3	20	13000	4	90	24000												
	DEPARTMENT_ID	MAX(SALARY)																										
1	80	11000																										
2	110	12000																										
3	20	13000																										
4	90	24000																										

## 10-3 Multiple-Row Subqueries (IN, ANY, ALL)

SELECT department_id, last_name, salary FROM employees WHERE salary IN (SELECT salary FROM employees WHERE department_id = 20) and department_id != 20;	SELECT department_id, last_name, salary FROM employees WHERE salary = <b>-- Solo un registro</b> (SELECT salary FROM employees WHERE department_id = 20) and department_id != 20;
SELECT department_id, last_name, salary FROM employees WHERE salary > <b>ANY</b> (SELECT salary FROM employees WHERE department_id = 20) and department_id != 20 order by salary;	SELECT department_id, last_name, salary FROM employees WHERE salary > (SELECT <b>MIN</b> (salary) FROM employees WHERE department_id = 20) and department_id != 20 order by salary;
SELECT department_id, last_name, salary FROM employees WHERE salary > <b>ALL</b> (SELECT salary FROM employees WHERE department_id = 20) and department_id != 20 order by salary;	SELECT department_id, last_name, salary FROM employees WHERE salary > (SELECT <b>MAX</b> (salary) FROM employees WHERE department_id = 20) and department_id != 20 order by salary;

Pairwise
<pre>SELECT column1.. FROM table 1 WHERE (column2, column3) = (SELECT column2, column3                              FROM table 1                              WHERE column 4 = expression);</pre>
Non-pairwise
<pre>SELECT column1.. FROM table 1 WHERE column2 = (SELECT column2 FROM table 1                  WHERE column 4 = expression)   AND column3 = (SELECT column3 FROM table 2                  WHERE column 4 = expression);</pre>

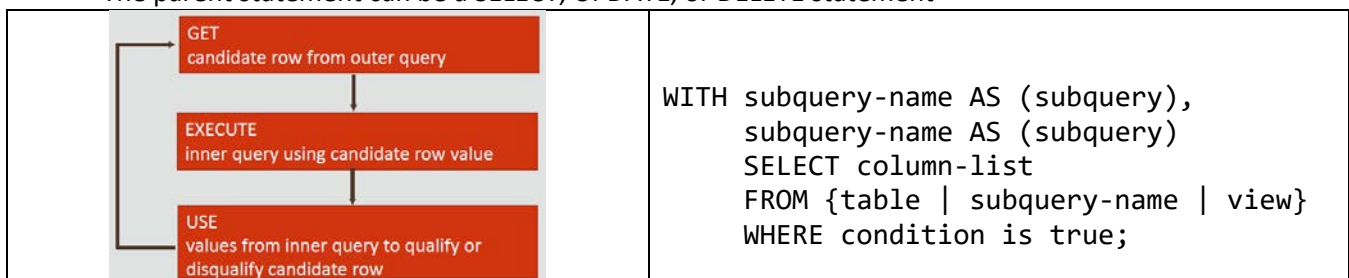
## 10-4 Correlated Subqueries

Correlated and non-correlated subqueries:

They need to include that the inner query in correlated subqueries executes once for each in the outer query. The outer query will run first.

Correlated Subqueries:

- The Oracle server performs a correlated subquery when the subquery references a column from a table referred to in the parent statement.
- A correlated subquery is evaluated once for each row processed by the parent statement.
- The parent statement can be a SELECT, UPDATE, or DELETE statement



Example: Whose salary is higher than the average salary of their department?

<pre> Select e.department_id ,e.last_name, e.salary from employees e       inner join (select department_id, avg(salary) promedio                   from employees                   group by department_id) t                   on e.department_id = t.department_id where e.salary &gt; promedio order by e.department_id; </pre>	Not Correlated Subqueries:
---	----------------------------

<pre> SELECT e.department_id ,e.last_name, e.salary FROM employees e WHERE e.salary &gt;       (SELECT AVG(sc.salary) promedio        FROM employees sc        WHERE sc.department_id = e.department_id) order by e.department_id; </pre>	Correlated Subqueries:
---	------------------------

<pre> WITH t as (select department_id, avg(salary) promedio            from employees            group by department_id) Select e.department_id ,e.last_name, e.salary from employees e, t where e.department_id = t.department_id       and e.salary &gt; promedio order by e.department_id; </pre>	<b>WITH</b>
--	-------------

EXISTS & NOT EXISTS in Subqueries. NOT EXISTS vs NOT IN

Correlated Subqueries:	Not Correlated Subqueries:
<pre> SELECT last_name "Not a Manager" FROM employees emp WHERE NOT EXISTS       (SELECT *        FROM employees mgr        WHERE mgr.manager_id = emp.employee_id); </pre>	<pre> SELECT last_name AS "Not a Manager" FROM employees emp WHERE emp.employee_id NOT IN       (SELECT distinct mgr.manager_id        FROM employees mgr        where mgr.manager_id is not null); </pre>

## Joining a Table to Itself

**EMPLOYEES (WORKER)**

	EMPLOYEE_ID	LAST_NAME	MANAGER_ID
1	100	King	(null)
2	101	Kochhar	100
3	102	De Haan	100
4	103	Hunold	102
5	104	Ernst	103
6	107	Lorentz	103
7	124	Mourgos	100
8	141	Rajs	124
9	142	Davies	124
10	143	Matos	124

...

**EMPLOYEES (MANAGER)**

EMPLOYEE_ID	LAST_NAME
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst
107	Lorentz
124	Mourgos
141	Rajs
142	Davies
143	Matos

...

**MANAGER\_ID in the WORKER table is equal to  
EMPLOYEE\_ID in the MANAGER table.**