26-8-2022

Database Programming with SQL 1 / 2

JUAN CARLOS HERRERA HERNANDEZ
UNIVERSIDAD POLITECNICA DE AGUASCALIENTES

Contenido

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:

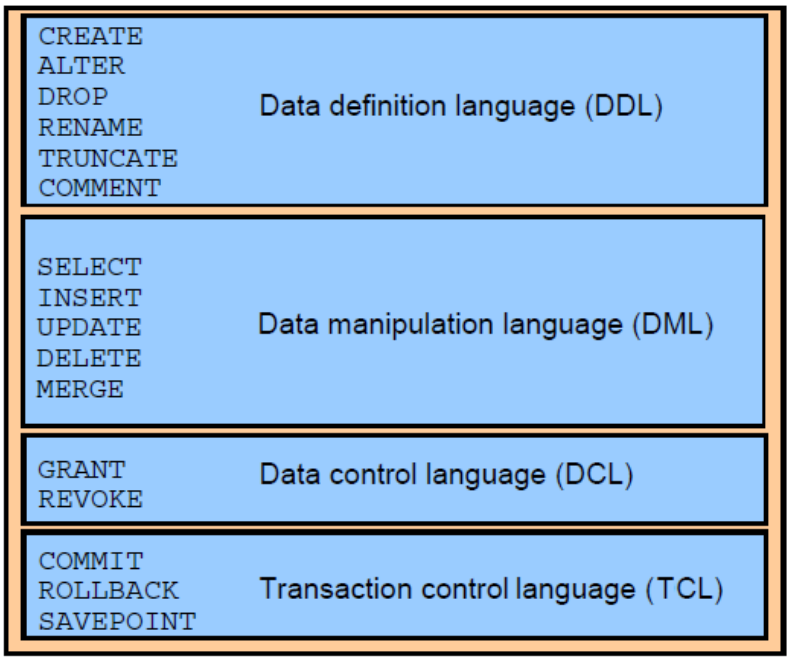
- SQL Workshop (to learn SQL)
- Application Builder (to design an application).

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



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 columns in a table that you want returned from a query
NULL	A value that is unavailable, unassigned, unknown, 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 rows 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 = Not equal to <> , !=	select sysdate from dual; -- Server select current_date from dual; -- Client	SELECT * from employees 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

`select employee_id, job_id`
`from employees`
`where job_id like '%\_P%' ESCAPE '\';`
	`select employee_id, job_id`
	`from employees`
	`where job_id like '%\_P%';`

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(50,80)
OR first_name LIKE 'C%'
AND last_name LIKE '%s%';
```

```
SELECT department_id, first_name, last_name
FROM employees
WHERE (department_id IN (50,80)
   OR first_name LIKE 'C%')
  AND last_name LIKE '%s%';
```

3-2 Sorting Rows

Ascending / ASC	Orders the rows in ascending order (the default order); A-Z
Descending / DESC	Orders the rows in descending order: Z-A
Sort	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

DUAL	Dummy table used to view results from functions and calculations. The DUAL table has one row called "X" and one column called "DUMMY"
Single-row functions	Functions that operate on single rows only and return one result per row
Character functions	Functions that accept character data as input and can return both character and numeric values
Input	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;
Output	Data that is processed into information
Format	The arrangement of data for storage or display
Expression	A symbol that represents a quantity or a relationship between quantities
INITCAP	Converts alpha character values to uppercase for the first letter of each word, all other letters in lowercase.
UPPER	Converts alpha characters to upper case
LOWER	Converts alpha character values to lowercase
TRIM	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;
CONCAT	Concatenates the first character value to the second character value; equivalent to concatenation operator (||).
LPAD	Pads the left side of a character, resulting in a right-justified value
SUBSTR	Returns specific characters from character value starting at a specific character position and going specified character positions long
REPLACE	Replaces a sequence of characters in a string with another set of characters. SELECT REPLACE('JACK and JUE', 'J', 'BL') FROM DUAL;
INSTR	Returns the numeric position of a named string.
LENGTH	Returns the number of characters in the expression
RPAD	Pads the right-hand side of a character, resulting in a left- justified value.

4-2 Number Functions

Number functions	These functions accept numeric input and return numeric values.

TRUNC	Used to terminate the column, expression, or value to a specified number of decimal places
MOD	Returns the remainder of a division. Even or odd (par o impar)
ROUND	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

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

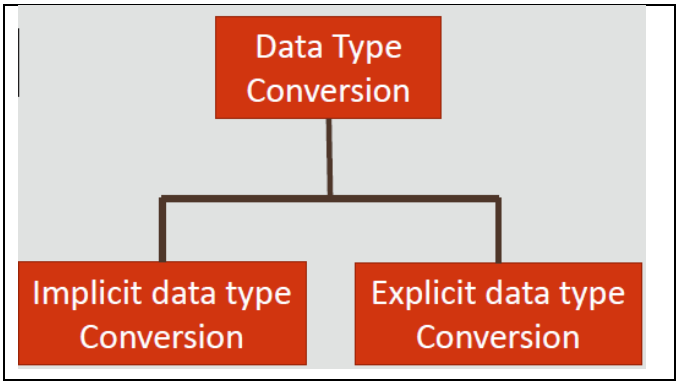
```
select * from nls_session_parameters
where parameter = 'NLS_DATE_LANGUAGE';

alter session set nls_language = english | american;
```

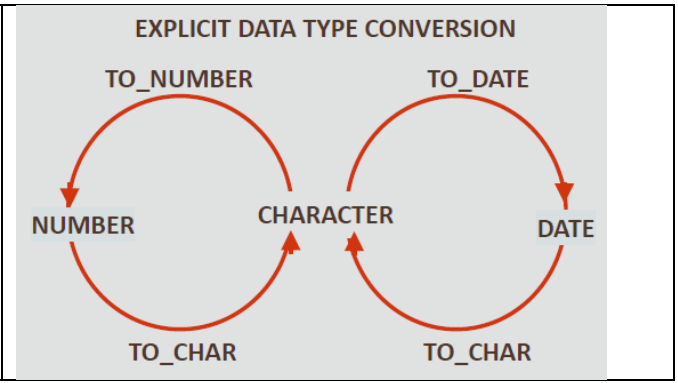
ORACLE	MySQL
select extract(year from sysdate) from dual;	select year(sysdate()) from dual;
select to_char(sysdate,'D') from dual; -- (1=dom 2=lun 7=Sab)	

Section 5 – Single Row Functions Part II

5-1 Conversion Functions

CHAR	Used for text and character data of fixed length, including numbers, dashes, and special characters.
fm	Used to remove padded blanks or to suppress leading zeros
Conversion function	Functions that convert a value from one datatype to another.
NUMBER	Used to store variable-length numeric data.
VARCHAR2	Used for character data of variable length, including numbers, special characters, and dashes.
DATE	Used for date and time values. DD-Mon-YYYY (for example, 23-Oct-2013)
TO_CHAR	Converts dates or numbers to character strings with optional formatting
RR date format	Century value depends on the specified year and the last two digits of the current year
TO_NUMBER	Converts a character string containing digits to a number with optional formatting
DD date format	Numeric day of the month
TO_DATE	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 date '2022-12-25' - SYSDATE FROM DUAL; SELECT SYSDATE - TO_DATE('25-DEC-2022', 'DD/MON/YYYY') 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 **RR** :	
		0-49	50-99
If the two first digits of the **current year** 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';

5-2 NULL Functions

NVL	Converts nulls to an actual value
	
	`Sintaxis: NVL(test_value, if_null)`
	`Example: NVL(commission_pct,0)`
	
	MySQL: SELECT NVL(commission_pct,0), **IFNULL**(commission_pct,0) FROM EMPLOYEES;

NVL2	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
	
	`Sintaxis: NVL2(test_value, if_not_null, if_null)`
	`Example: NVL2(commission_pct, 'SAL+COMM', 'SAL')`

COALESCE	Returns the first non-null expression in the list
	
	`Sintaxis: COALESCE (expr1, expr2, ..., exprn)`
	`Example: COALESCE(salary+commission_pct*salary, salary+2000)`

NULLIF	Compares two expressions; if they are equal, the function returns null; if they are not equal, the function returns the first expression
	
	`Sintaxis: NULLIF (Expr1, expr2)`
	`Example: NULLIF(LENGTH(first_name), LENGTH(last_name))`

5-3 Conditional Expressions

Conditional expression	An if-then-else expression whose value depends on the truth-value of a Boolean expression.

CASE	Implements conditional processing within a SQL statement; it meets the ANSI standard.
	`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') > 2 then 1 ELSE 0 end) Resto,`
	`count(*) total`
	`from employees`
	`group by EXTRACT(YEAR FROM hire_date);`

DECODE (Only Oracle)	Compares an expression to each of the search values
	`DECODE(columnl|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)`

IF (Only MySQL)	`IF(condition, true_value, false_value)`
	
	`SELECT sysdate(),`
	`if(MONTH(sysdate())= 1, 1, 0) as Ene`
	`FROM dual;`

Section 6 – JOINs Part I

6-1 Cross Joins and Natural Joins

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

NATURAL JOIN	**CROSS JOIN**
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

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

ON clause	**USING clause**

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 **non-equality** 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 **Three** 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

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

LEFT OUTER JOIN

```
SELECT e.last_name, d.department_id,
       d.department_name
FROM employees e LEFT OUTER JOIN
departments d
  ON (e.department_id= d.department_id);
```

RIGHT OUTER JOIN

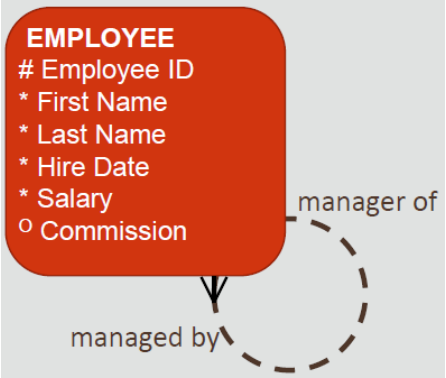
```
SELECT e.last_name, d.department_id,
       d.department_name
FROM employees e RIGHT OUTER JOIN
departments d
  ON e.department_id= d.department_id;
```

```
SELECT e.last_name, d.department_id,
       d.department_name
FROM employees e FULL OUTER JOIN departments d
  ON (e.department_id= d.department_id);
```

6-4 Self-Joins and Hierarchical Queries

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

An employee can also be a manager.



Workers (N = 20) Manager (N= 8)

EMPLOYEES (worker)

employee_id	last_name	manager_id
100	King	
101	Kochar	100
102	De Haan	100
103	Hunold	102
104	Ernst	103
107	Lorentz	103
124	Mourgos	100

EMPLOYEES (manager)

employee_id	last_name
100	King
101	Kochar
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;
```

```
select level, employee_id Trabajador, Last_name,  manager_id,
       lpad(employee_id, length(employee_id)+(level*2)-2,'-') as "Org Chart"
from employees
start with employee_id = 100   -- A partir de ahi, puede modificarse ej 101
connect by prior employee_id = manager_id;
```

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

Section 7 – JOINs Part II

7-1 Oracle Equijoin and Cartesian Product

Cartesian product	Results from an invalid or omitted join condition; all combinations of rows are displayed
Equijoin	Values in a column in one table are equal to a value in another table; also called an inner join or simple join
Proprietary join	Connection command exclusive to a specific company
Alias	Gives a table another name to simplify queries and improve performance
Join conditions	Display data from two or more related tables

Oracle Proprietary Join	**ANSI/ISO SQL: 1999 Equivalent**
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)

`SELECT last_name, e.job_id, job_title`
`FROM employees e, jobs j`
`WHERE e.job_id = j.job_id`
`  AND department_id = 80;`
	`SELECT e.last_name, d.department_name`
	`FROM employees e, departments d;`

join three tables

`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;`

7-2 Oracle Nonequijoins and Outer Joins

Nonequijoin
SELECT last_name, salary, grade_level, lowest_sal, highest_sal FROM employees, job_grades WHERE (salary **BETWEEN** lowest_sal **AND** highest_sal);

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

Section 8 – Group Functions Part I

8-1 Group Functions

COUNT	Returns the number of rows with non-null values for the expression Count(*) vs count(department_id)
SUM	Calculates the sum ignoring null values
AVG	Calculates average value excluding nulls
MIN	Returns minimum value with any data type ignoring nulls
MAX	Returns the maximum value with any data type ignoring nulls
STDDEV	For two sets of data with approximately the same mean, the greater the spread, the greater the standard deviation.
VARIANCE	Used with columns that store numeric data to calculate the spread of data around the mean
Group functions	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
Aggregate	To gather into a sum or whole

`SELECT group_function(column), .. FROM table WHERE condition;`	`SELECT column, group_function(column), .. FROM table WHERE condition GROUP BY column;`

8-2 COUNT, DISTINCT, NVL

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

GROUP BY	Divides the rows in a table into groups 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. You cannot use a column alias in the GROUP BY clause
HAVING	Used to specify which groups are to be displayed; restricts groups that do not meet group criteria
	If there is having, then there is a group. If there is group, does not necessarily having
WHERE	excludes rows **before** they are divided into groups

Correct	Find the Error

`SELECT department_id, MAX(salary)`
`FROM employees`
`WHERE last_name!= 'King'`
`GROUP BY department_id`
`HAVING COUNT(*) > 1`
`order by department_id;`
	`SELECT department_id, job_id, AVG(salary)`
	`FROM employees`
	`GROUP BY department_id`
	`order by 1;`

`( ORACLE Only)`
`SELECT max(avg(salary))`
`FROM employees`
`GROUP by department_id;`
	`SELECT department_id, MAX(salary)`
	`FROM employees`
	`WHERE COUNT(*) > 1`
	`GROUP BY department_id;`

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

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

ORACLE	MySQL
SELECT department_id, job_id, SUM(salary) FROM employees where department_id is not null GROUP BY **ROLLUP**(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 **CUBE**(department_id, job_id) order by 1, 2 ;	does not exist

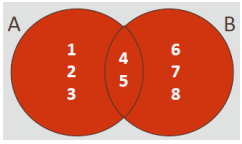
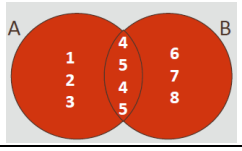
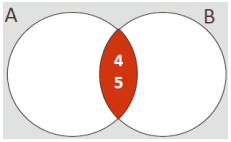
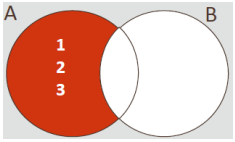
Grouping Set (ORACLE Only)
SELECT department_id, job_id, manager_id, SUM(salary) FROM employees WHERE department_id < 50 GROUP BY GROUPING SETS ((job_id, manager_id), (department_id, job_id), (department_id, manager_id));
Grouping Functions (ORACLE Only)
SELECT department_id, job_id, SUM(salary), GROUPING(department_id) AS "Dept sub total", GROUPING(job_id) AS "Job sub total" FROM employees WHERE department_id < 50 GROUP BY CUBE (department_id, job_id);

9-3 Using Set Operators

SET operators	used to combine results into one single result from multiple SELECT statements
UNION	operator that returns all rows from both tables and eliminates duplicates
UNION ALL	operator that returns all rows from both tables, including duplicates
INTERSECT	operator that returns rows common to both tables
MINUS	operator that returns rows that are unique to each table
TO_CHAR(null)	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

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

```
SELECT select_list
FROM table
WHERE expression operator
   (SELECT select_list
    FROM table);
```

```
SELECT select_list
FROM table1, (SELECT select_list
              FROM table2) alias
Where table1.key = alias.key
```

```
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;
```

```
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;
```

```
SELECT last_name, hire_date
FROM employees
WHERE hire_date>
   (SELECT hire_date
    FROM employees
    WHERE last_name= 'Vargas');
```

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
```

	MAX(SALARY)
1	9000

```
SELECT department_id, max(salary)
FROM employees
GROUP BY department_id
order by max(salary)
```

	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);
```

	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 =       -- Solo un registro
   (SELECT salary
    FROM employees
    WHERE department_id = 20)
and department_id != 20;
```

```
SELECT department_id, last_name, salary
FROM employees
WHERE salary > ANY
      (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 MIN(salary) FROM employees
       WHERE department_id = 20)
  and department_id != 20
order by salary;
```

```
SELECT department_id, last_name, salary
FROM employees
WHERE salary > ALL
      (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 MAX(salary) FROM employees
       WHERE department_id = 20)
  and department_id != 20
order by salary;
```

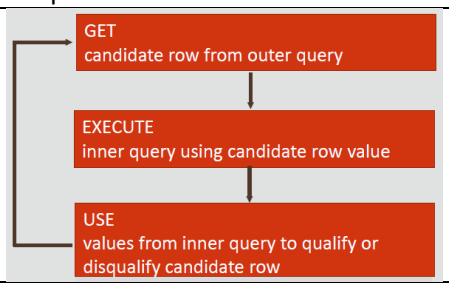
10-4 Correlated Subqueries

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

	`WITH subquery-name AS (subquery),`
	`subquery-name AS (subquery)`
	`SELECT column-list`
	`FROM {table | subquery-name | view}`
	`WHERE condition is true;`

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

`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 > promedio`
`order by e.department_id;`
	Not Correlated Subqueries:

`SELECT e.department_id ,e.last_name, e.salary`
`FROM employees e`
`WHERE e.salary >`
`(SELECT AVG(sc.salary) promedio`
`FROM employees sc`
`WHERE sc.department_id = e.department_id)`
`order by e.department_id;`
	Correlated Subqueries:

`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 > promedio`
`order by e.department_id;`
	WITH

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

Correlated Subqueries:	Not Correlated Subqueries:
SELECT last_name "Not a Manager" FROM employees emp WHERE **NOT EXISTS** (SELECT * FROM employees mgr WHERE mgr.manager_id = emp.employee_id);	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**);

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.