ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

Laboratorio de:

Materia: Fundamentos de Bases de Datos

Práctica No.: LABORATORIO PRÁCTICO - TÓPICO 6

Tema: Consultas SQL en BD relacionales

SGBD: Oracle Database

TABLA DE CONTENIDOS

OBJETIVOS

Objetivo General

Dominar el lenguaje de consultas SQL para extraer, analizar y presentar información de bases de datos relacionales, utilizando consultas simples y complejas con múltiples tablas.

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

Objetivos Específicos

1. **Consultar datos** usando SELECT con todas sus cláusulas
2. **Filtrar información** con WHERE y operadores lógicos
3. **Ordenar resultados** con ORDER BY
4. **Agregar datos** con funciones COUNT, SUM, AVG, MAX, MIN
5. **Agrupar información** con GROUP BY y HAVING
6. **Relacionar tablas** usando INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN
7. **Crear subconsultas** en SELECT, WHERE y FROM
8. **Usar operadores de conjuntos** UNION, INTERSECT, MINUS
9. **Optimizar consultas** para mejor rendimiento
10. **Resolver problemas complejos** con consultas SQL

REQUISITOS PREVIOS

Conocimientos Necesarios

- Laboratorios 4 (DDL) y 5 (DML) completados
- Datos cargados en las tablas del sistema académico
- Comprensión del modelo relacional
- Álgebra relacional básica

Verificación de Datos

```
-- Conectarse al usuario
CONN gestion_academica/EPN2024Secure;


-- Verificar que hay datos en las tablas
SELECT 'CARRERA' AS tabla, COUNT(*) AS registros FROM Carrera
UNION ALL
SELECT 'ESTUDIANTE', COUNT(*) FROM Estudiante
```

```
UNION ALL
SELECT 'ASIGNATURA', COUNT(*) FROM Asignatura
UNION ALL
SELECT 'DOCENTE', COUNT(*) FROM Docente
UNION ALL
SELECT 'MATRICULA', COUNT(*) FROM Matricula
UNION ALL
SELECT 'PRERREQUISITO', COUNT(*) FROM Prerrequisito;
```

Hoja de Trabajo | Generador de Consultas

```
-- Verificar que hay datos en las tablas
SELECT 'CARRERA' AS tabla, COUNT(*) AS registros FROM Carrera UNION ALL
SELECT 'ESTUDIANTE', COUNT(*) FROM Estudiante UNION ALL
SELECT 'ASIGNATURA', COUNT(*) FROM Asignatura UNION ALL
SELECT 'DOCENTE', COUNT(*) FROM Docente UNION ALL
SELECT 'MATRICULA', COUNT(*) FROM Matricula UNION ALL
SELECT 'PRERREQUISITO', COUNT(*) FROM Prerrequisito;
```

Salida de Script | Resultado de la Consulta | Resultado de la Consulta 1 | Resultado de la Consulta 2 | Salida de Script 1 | Resultado de la...

SQL | Todas las Filas Recuperadas: 6 en 0,01 segundos

	TABLA	REGISTROS
1	CARRERA	7
2	ESTUDIANTE	5
3	ASIGNATURA	10
4	DOCENTE	5
5	MATRICULA	5
6	PRERREQUISITO	5

-- Debería mostrar datos en todas las tablas

-- Si alguna tabla está vacía, ejecutar scripts de INSERT del Lab 5

Preparar Datos de Prueba Adicionales

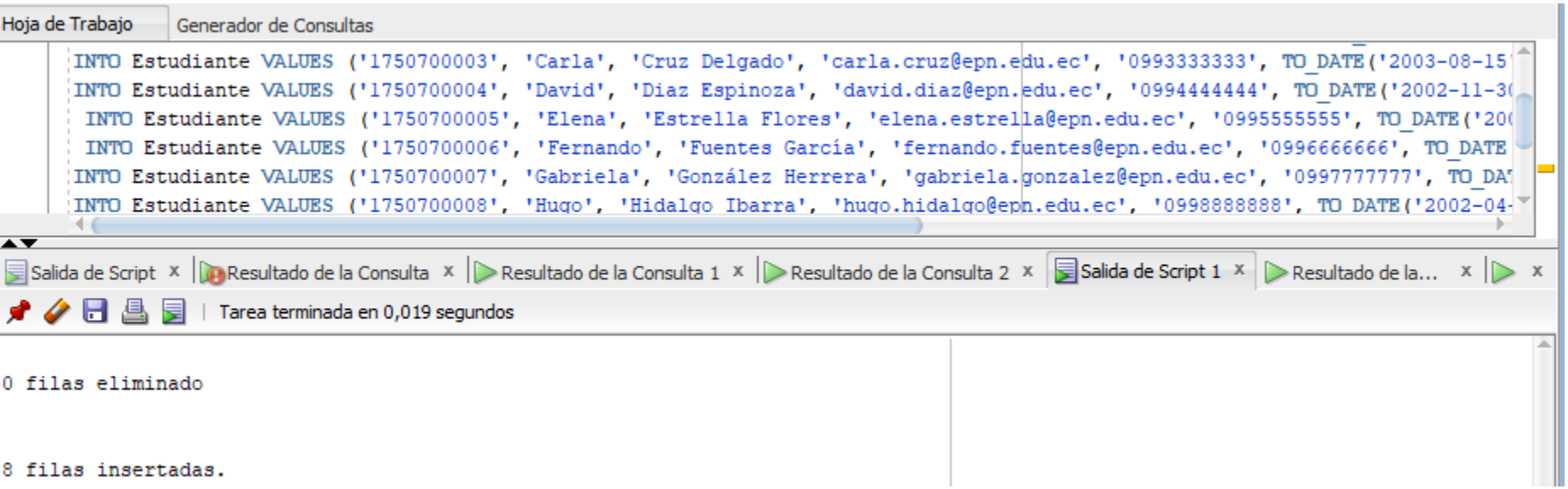
ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

-- Si necesita más datos para práctica, ejecutar:

-- Más estudiantes

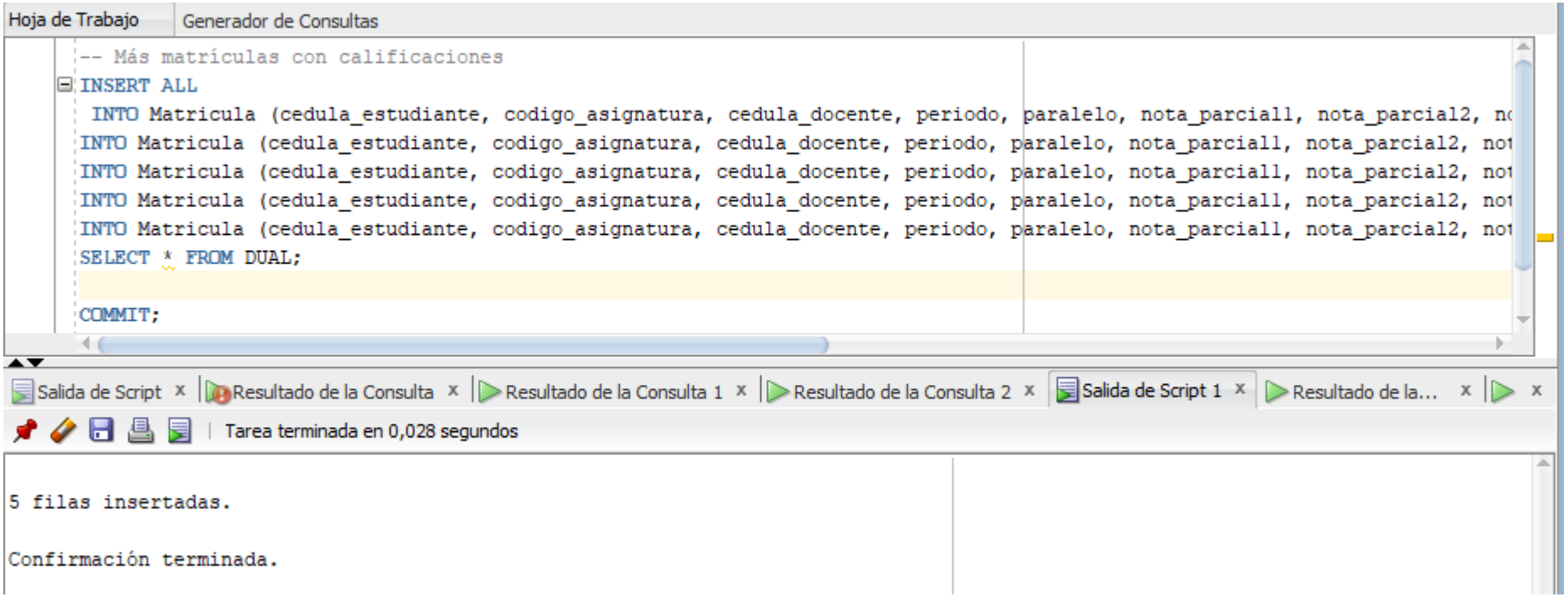
```
INSERT ALL INTO Estudiante VALUES ('1750700001', 'Andrea', 'Alvarez Brito', 'andrea.alvarez@epn.edu.ec', '0991111111', TO_DATE('2003-01-10', 'YYYY-MM-DD'), 'F',
'ING-SIS', 'ACTIVO', 15, SYSDATE)
 INTO Estudiante VALUES ('1750700002', 'Bryan', 'Bravo Cárdenas', 'bryan.bravo@epn.edu.ec', '0992222222', TO_DATE('2002-05-22', 'YYYY-MM-DD'), 'M', 'ING-SIS',
'ACTIVO', 20, SYSDATE)
INTO Estudiante VALUES ('1750700003', 'Carla', 'Cruz Delgado', 'carla.cruz@epn.edu.ec', '0993333333', TO_DATE('2003-08-15', 'YYYY-MM-DD'), 'F', 'ING-CIV', 'ACTIVO',
10, SYSDATE)
INTO Estudiante VALUES ('1750700004', 'David', 'Diaz Espinoza', 'david.diaz@epn.edu.ec', '0994444444', TO_DATE('2002-11-30', 'YYYY-MM-DD'), 'M', 'ING-ELE',
'ACTIVO', 25, SYSDATE)
 INTO Estudiante VALUES ('1750700005', 'Elena', 'Estrella Flores', 'elena.estrella@epn.edu.ec', '0995555555', TO_DATE('2003-03-18', 'YYYY-MM-DD'), 'F', 'ING-SIS',
'ACTIVO', 30, SYSDATE)
 INTO Estudiante VALUES ('1750700006', 'Fernando', 'Fuentes García', 'fernando.fuentes@epn.edu.ec', '0996666666', TO_DATE('2002-07-25', 'YYYY-MM-DD'), 'M', 'ING-
SIS', 'INACTIVO', 5, SYSDATE)
INTO Estudiante VALUES ('1750700007', 'Gabriela', 'González Herrera', 'gabriela.gonzalez@epn.edu.ec', '0997777777', TO_DATE('2003-12-05', 'YYYY-MM-DD'), 'F', 'ING-
CIV', 'ACTIVO', 35, SYSDATE)
INTO Estudiante VALUES ('1750700008', 'Hugo', 'Hidalgo Ibarra', 'hugo.hidalgo@epn.edu.ec', '0998888888', TO_DATE('2002-04-14', 'YYYY-MM-DD'), 'M', 'ING-ELE',
'GRADUADO', 180, SYSDATE) SELECT * FROM DUAL;
```

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS



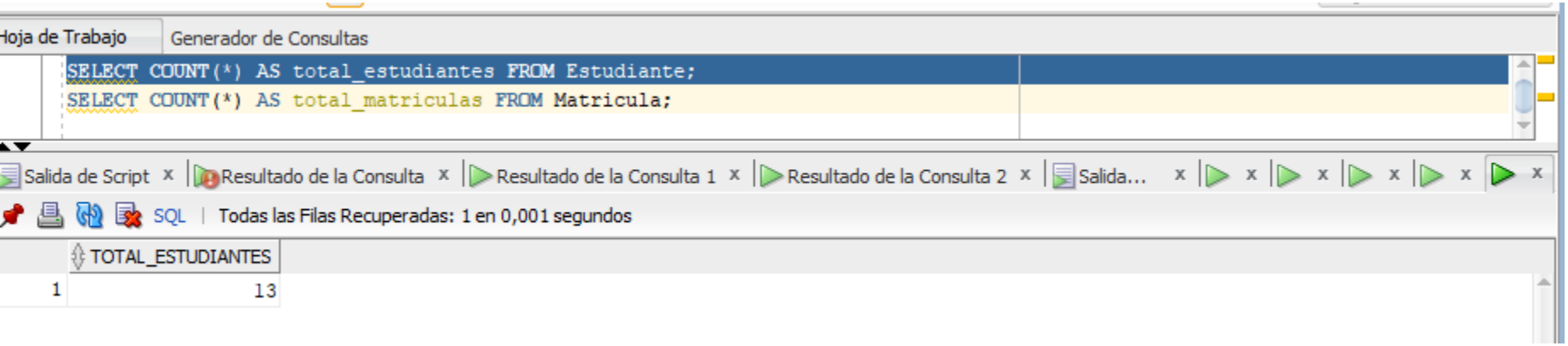

```
-- Más matrículas con calificaciones
INSERT ALL
 INTO Matricula (cedula_estudiante, codigo_asignatura, cedula_docente, periodo, paralelo, nota_parcial1, nota_parcial2, nota_final, estado) VALUES ('1750700001', 'BD-501',
'1712345678', '2024-2S', 'A', 8.5, 9.0, 8.75, 'APROBADO')
INTO Matricula (cedula_estudiante, codigo_asignatura, cedula_docente, periodo, paralelo, nota_parcial1, nota_parcial2, nota_final, estado) VALUES ('1750700002', 'BD-501',
'1712345678', '2024-2S', 'A', 7.0, 7.5, 7.25, 'APROBADO')
INTO Matricula (cedula_estudiante, codigo_asignatura, cedula_docente, periodo, paralelo, nota_parcial1, nota_parcial2, nota_final, estado) VALUES ('1750700005', 'BD-501',
'1712345678', '2024-2S', 'A', 9.5, 9.0, 9.25, 'APROBADO')
INTO Matricula (cedula_estudiante, codigo_asignatura, cedula_docente, periodo, paralelo, nota_parcial1, nota_parcial2, nota_final, estado) VALUES ('1750700001', 'PRG-201',
'1734567890', '2024-2S', 'B', 6.5, 6.0, 6.25, 'REPROBADO')
INTO Matricula (cedula_estudiante, codigo_asignatura, cedula_docente, periodo, paralelo, nota_parcial1, nota_parcial2, nota_final, estado) VALUES ('1750700002', 'PRG-201',
'1734567890', '2024-2S', 'B', 8.0, 8.5, 8.25, 'APROBADO')
SELECT * FROM DUAL;

    COMMIT;
```

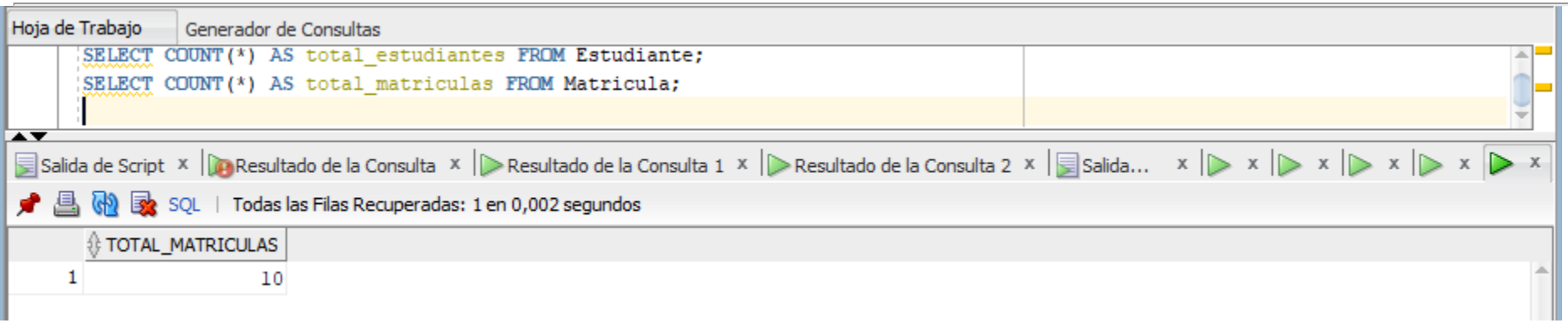
ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS



-- Verificar

```
SELECT COUNT(*) AS total_estudiantes FROM Estudiante;
SELECT COUNT(*) AS total_matriculas FROM Matricula;
```





FUNDAMENTOS DE SELECT

Sintaxis Básica

```
SELECT columna1, columna2, columna3, ...
FROM nombre_tabla;

-- O para todas las columnas
SELECT * FROM nombre_tabla;
```

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

Ejemplo 1: SELECT Simple

```
-- EJEMPLO 1: SELECT básico - todas las columnas
-- ==================================================

-- Seleccionar todas las carreras
SELECT * FROM Carrera;

-- Seleccionar columnas específicas
SELECT codigo, nombre, duracion_semestres
FROM Carrera;

-- Usar alias para columnas
SELECT
    codigo AS "Código",
    nombre AS "Nombre de Carrera",
    duracion_semestres AS "Duración (semestres)",
    creditos_totales AS "Créditos"
FROM Carrera;
```

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

Hoja de Trabajo | Generador de Consultas

```sql
-- EJEMPLO 1: SELECT básico - todas las columnas
-- ================================================

-- Seleccionar todas las carreras
SELECT * FROM Carrera;

-- Seleccionar columnas específicas
SELECT codigo, nombre, duracion_semestres FROM Carrera;

-- Usar alias para columnas
SELECT
codigo AS "Código",
nombre AS "Nombre de Carrera", duracion_semestres AS "Duración (semestres)", creditos_totales AS "Créditos"
FROM Carrera;
```

Salida de Script x | Resultado de la Consulta x | Resultado de la Consulta 1 x | Resultado de la Consulta 2 x

SQL | Todas las Filas Recuperadas: 7 en 0,002 segundos

	CODIGO	NOMBRE	DURACION_SEMESTRES	CREDITOS_TOTALES	FACULTAD
1	ING-SIS	Ingenier¡a en Sistemas Inform ticos	10	180	Facultad de Sistemas - ESFOT
2	ING-CIV	Ingenier¡a Civil	10	200	Facultad de Ingenier¡a Civil y Ambiental -
3	ING-ELE	Ingenier¡a El,ctrica	10	190	Facultad de Ingenier¡a El,ctrica y Electr¢
4	ING-MEC	Ingenier¡a Mec nica	10	195	Facultad de Ingenier¡a Mec nica - FIM
5	ING-QUI	Ingenier¡a Qu¡mica	10	185	Facultad de Ingenier¡a Qu¡mica - FIQ
6	ING-IND	Ingenier¡a Industrial	10	175	Facultad de Ingenier¡a Industrial - FIIS
7	ING-GEO	Ingenier¡a Geol¢gica	10	192	Facultad de Geolog¡a y Petr¢leos - FGP

Hoja de Trabajo | Generador de Consultas

```sql
-- EJEMPLO 1: SELECT básico - todas las columnas
-- ======================================================

-- Seleccionar todas las carreras
SELECT * FROM Carrera;

-- Seleccionar columnas específicas
SELECT codigo, nombre, duracion_semestres FROM Carrera;

-- Usar alias para columnas
SELECT
codigo AS "Código",
nombre AS "Nombre de Carrera", duracion_semestres AS "Duración (semestres)", creditos_totales AS "Créditos"
FROM Carrera;
```

Salida de Script x | Resultado de la Consulta x | Resultado de la Consulta 1 x | Resultado de la Consulta 2 x

SQL | Todas las Filas Recuperadas: 7 en 0,001 segundos

	CODIGO	NOMBRE	DURACION_SEMESTRES
1	ING-SIS	Ingenier¡a en Sistemas Inform ticos	10
2	ING-CIV	Ingenier¡a Civil	10
3	ING-ELE	Ingenier¡a El,ctrica	10
4	ING-MEC	Ingenier¡a Mec nica	10
5	ING-QUI	Ingenier¡a Qu¡mica	10
6	ING-IND	Ingenier¡a Industrial	10
7	ING-GEO	Ingenier¡a Geol◦gica	10

Hoja de Trabajo | Generador de Consultas

```sql
-- EJEMPLO 1: SELECT básico - todas las columnas
-- ================================================

-- Seleccionar todas las carreras
SELECT * FROM Carrera;

-- Seleccionar columnas específicas
SELECT codigo, nombre, duracion_semestres FROM Carrera;

-- Usar alias para columnas
SELECT
codigo AS "Código",
nombre AS "Nombre de Carrera", duracion_semestres AS "Duración (semestres)", creditos_totales AS "Créditos"
FROM Carrera;
```

Salida de Script | Resultado de la Consulta | Resultado de la Consulta 1 | Resultado de la Consulta 2

SQL | Todas las Filas Recuperadas: 7 en 0,001 segundos

	Código	Nombre de Carrera	Duración (semestres)	Créditos
1	ING-SIS	Ingenier¡a en Sistemas Inform ticos	10	180
2	ING-CIV	Ingenier¡a Civil	10	200
3	ING-ELE	Ingenier¡a El,ctrica	10	190
4	ING-MEC	Ingenier¡a Mec nica	10	195
5	ING-QUI	Ingenier¡a Qu¡mica	10	185
6	ING-IND	Ingenier¡a Industrial	10	175
7	ING-GEO	Ingenier¡a Geol◆gica	10	192

Ejemplo 2: DISTINCT

```sql
-- EJEMPLO 2: Eliminar duplicados con DISTINCT
-- ================================================
```

```
-- Ver todos los estados de estudiantes (con duplicados)
SELECT estado FROM Estudiante;

-- Ver estados únicos
SELECT DISTINCT estado FROM Estudiante;

-- Combinaciones únicas de carrera y estado

SELECT DISTINCT codigo_carrera, estado
FROM Estudiante
ORDER BY codigo_carrera, estado;

-- Contar valores únicos

SELECT COUNT(DISTINCT codigo_carrera) AS total_carreras_con_estudiantes
FROM Estudiante;
```

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

Hoja de Trabajo | Generador de Consultas

```sql
-- Ver todos los estados de estudiantes (con duplicados)
SELECT estado FROM Estudiante;

-- Ver estados únicos
SELECT DISTINCT estado FROM Estudiante;

-- Combinaciones únicas de carrera y estado

SELECT DISTINCT codigo_carrera, estado FROM Estudiante
ORDER BY codigo_carrera, estado;

-- Contar valores únicos

SELECT COUNT(DISTINCT codigo_carrera) AS total_carreras_con_estudiantes FROM Estudiante;
```

Salida de Script x | Resultado de la Consulta x | Resultado de la Consulta 1 x | Resultado de la Consulta 2 x | Resultado de la Consulta 3 x

SQL | Todas las Filas Recuperadas: 13 en 0,003 segundos

	ESTADO
1	ACTIVO
2	ACTIVO
3	ACTIVO
4	ACTIVO
5	ACTIVO
6	ACTIVO
7	ACTIVO
8	ACTIVO
9	ACTIVO
10	ACTIVO
11	ACTIVO
12	GRADUADO
13	INACTIVO

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

Hoja de Trabajo | Generador de Consultas

```sql
-- Ver todos los estados de estudiantes (con duplicados)
SELECT estado FROM Estudiante;

-- Ver estados únicos
SELECT DISTINCT estado FROM Estudiante;

-- Combinaciones únicas de carrera y estado

SELECT DISTINCT codigo_carrera, estado FROM Estudiante
ORDER BY codigo_carrera, estado;

-- Contar valores únicos

SELECT COUNT(DISTINCT codigo_carrera) AS total_carreras_con_estudiantes FROM Estudiante;
```

Salida de Script x | Resultado de la Consulta x | Resultado de la Consulta 1 x | Resultado de la Consulta 2 x | Resultado de la Consulta 3 x

SQL | Todas las Filas Recuperadas: 3 en 0,01 segundos

	ESTADO
1	ACTIVO
2	GRADUADO
3	INACTIVO

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

gestionacademica

Hoja de Trabajo | Generador de Consultas

```sql
-- Ver todos los estados de estudiantes (con duplicados)
SELECT estado FROM Estudiante;

-- Ver estados únicos
SELECT DISTINCT estado FROM Estudiante;

-- Combinaciones únicas de carrera y estado

SELECT DISTINCT codigo_carrera, estado FROM Estudiante
ORDER BY codigo_carrera, estado;

-- Contar valores únicos

SELECT COUNT(DISTINCT codigo_carrera) AS total_carreras_con_estudiantes FROM Estudiante;
```

Salida de Script | Resultado de la Consulta | Resultado de la Consulta 1 | Resultado de la Consulta 2 | Resultado de la Consulta 3

SQL | Todas las Filas Recuperadas: 5 en 0,004 segundos

	CODIGO_CARRERA	ESTADO
1	ING-CIV	ACTIVO
2	ING-ELE	ACTIVO
3	ING-ELE	GRADUADO
4	ING-SIS	ACTIVO
5	ING-SIS	INACTIVO

Hoja de Trabajo | Generador de Consultas

```sql
-- Ver todos los estados de estudiantes (con duplicados)
SELECT estado FROM Estudiante;

-- Ver estados únicos
SELECT DISTINCT estado FROM Estudiante;

-- Combinaciones únicas de carrera y estado

SELECT DISTINCT codigo_carrera, estado FROM Estudiante
ORDER BY codigo_carrera, estado;

-- Contar valores únicos

SELECT COUNT(DISTINCT codigo_carrera) AS total_carreras_con_estudiantes FROM Estudiante;
```

Salida de Script | Resultado de la Consulta | Resultado de la Consulta 1 | Resultado de la Consulta 2 | Resultado de la Consulta 3

SQL | Todas las Filas Recuperadas: 1 en 0,002 segundos

	TOTAL_CARRERAS_CON_ESTUDIANTES
1	3

Ejemplo 3: Expresiones y Cálculos

```sql
-- EJEMPLO 3: Cálculos en SELECT
-- ================================================

-- Calcular horas totales de cada asignatura
SELECT
    codigo,
    nombre,
    horas_teoria,
    horas_practica,
```

```
    (horas_teoria + horas_practica) AS horas_totales
FROM Asignatura;

-- Concatenar cadenas
SELECT
    cedula,
    nombres || ' ' || apellidos AS nombre_completo,
    email
FROM Estudiante;

-- Cálculos con NULL
SELECT
    codigo,
    nombre,
    creditos,
    creditos * 16 AS horas_estimadas_estudio  -- 16 horas por crédito
FROM Asignatura;

-- Formato de números
SELECT
    nombres,
    apellidos,
    creditos_aprobados,
    ROUND(creditos_aprobados / 180 * 100, 2) AS porcentaje_avance
FROM Estudiante
WHERE creditos_aprobados > 0;
```

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

Hoja de Trabajo | Generador de Consultas

```
FROM Estudiante;

-- Cálculos con NULL
SELECT
```

Salida de Script x | Resultado de la Consulta x | Resultado de la Consulta 1 x | Resultado de la Consulta 2 x | Resultado de la Consulta 3 x

SQL | Todas las Filas Recuperadas: 10 en 0,007 segundos

	CODIGO	NOMBRE	HORAS_TEORIA	HORAS_PRACTICA	HORAS_TOTALES
1	MAT-101	C lculo Diferencial	4	2	6
2	FIS-101	F¡sica I	3	2	5
3	PRG-101	Fundamentos de Programaci¢n	3	4	7
4	MAT-201	C lculo Integral	4	2	6
5	PRG-201	Programaci¢n Orientada a Objetos	3	4	7
6	EST-301	Estructura de Datos	3	4	7
7	BD-501	Fundamentos de Bases de Datos	3	2	5
8	BD-601	Bases de Datos Avanzadas	3	4	7
9	CIV-101	Dibujo T,cnico	2	4	6
10	ELE-101	Circuitos El,ctricos I	4	2	6

Hoja de Trabajo | Generador de Consultas

```
FROM Estudiante;

-- Cálculos con NULL
SELECT
```

Salida de Script | Resultado de la Consulta | Resultado de la Consulta 1 | Resultado de la Consulta 2 | Resultado de la Consulta 3

SQL | Todas las Filas Recuperadas: 13 en 0,002 segundos

	CEDULA	NOMBRE_COMPLETO	EMAIL
1	1750700001	Andrea Alvarez Brito	andrea.alvarez@epn.edu.ec
2	1750700002	Bryan Bravo Cárdenas	bryan.bravo@epn.edu.ec
3	1750700003	Carla Cruz Delgado	carla.cruz@epn.edu.ec
4	1750700004	David Diaz Espinoza	david.diaz@epn.edu.ec
5	1750700005	Elena Estrella Flores	elena.estrella@epn.edu.ec
6	1750700006	Fernando Fuentes García	fernando.fuentes@epn.edu.ec
7	1750700007	Gabriela González Herrera	gabriela.gonzalez@epn.edu.ec
8	1750700008	Hugo Hidalgo Ibarra	hugo.hidalgo@epn.edu.ec
9	1750123456	Juan Pablo Andrade Morales	juan.andrade.nuevo@epn.edu.ec
10	1750234567	Mar¡a Jos, Ben¡tez Castro	maria.benitez@epn.edu.ec
11	1750345678	Carlos Andr,s Castillo D¡az	carlos.castillo@epn.edu.ec
12	1750456789	Ana Sof¡a Dom¡nguez Escobar	ana.dominguez@epn.edu.ec
13	1750567890	Pedro Luis Espinoza Flores	pedro.espinoza@epn.edu.ec

Hoja de Trabajo | Generador de Consultas

```
FROM Estudiante;

-- Cálculos con NULL
SELECT
```

Salida de Script x | Resultado de la Consulta x | Resultado de la Consulta 1 x | Resultado de la Consulta 2 x | Resultado de la Consulta 3 x

SQL | Todas las Filas Recuperadas: 10 en 0,002 segundos

	CODIGO	NOMBRE	CREDITOS	HORAS_ESTIMADAS_ESTUDIO
1	MAT-101	C lculo Diferencial	5	80
2	FIS-101	F¡sica I	4	64
3	PRG-101	Fundamentos de Programaci¢n	5	80
4	MAT-201	C lculo Integral	5	80
5	PRG-201	Programaci¢n Orientada a Objetos	5	80
6	EST-301	Estructura de Datos	5	80
7	BD-501	Fundamentos de Bases de Datos	4	64
8	BD-601	Bases de Datos Avanzadas	5	80
9	CIV-101	Dibujo T,cnico	4	64
10	ELE-101	Circuitos El,ctricos I	5	80

}

Hoja de Trabajo | Generador de Consultas

```
FROM Estudiante;

-- Cálculos con NULL
SELECT
```

Salida de Script | Resultado de la Consulta | Resultado de la Consulta 1 | Resultado de la Consulta 2 | Resultado de la Consulta 3

SQL | Todas las Filas Recuperadas: 11 en 0,002 segundos

	NOMBRES	APELLIDOS	CREDITOS_APROBADOS	PORCENTAJE_AVANCE
1	Andrea	Alvarez Brito	15	8,33
2	Bryan	Bravo Cárdenas	20	11,11
3	Carla	Cruz Delgado	10	5,56
4	David	Diaz Espinoza	25	13,89
5	Elena	Estrella Flores	30	16,67
6	Fernando	Fuentes García	5	2,78
7	Gabriela	González Herrera	35	19,44
8	Hugo	Hidalgo Ibarra	180	100
9	Juan Pablo	Andrade Morales	28	15,56
10	Mar¡a Jos,	Ben¡tez Castro	4	2,22
11	Pedro Luis	Espinoza Flores	5	2,78

Ejemplo 4: Funciones de Cadena

```sql
-- EJEMPLO 4: Funciones de texto
-- ================================================

-- Convertir a mayúsculas/minúsculas
SELECT
    UPPER(nombres) AS nombres_mayusculas,
    LOWER(apellidos) AS apellidos_minusculas,
    INITCAP(email) AS email_capitalizado
```

```
FROM Estudiante
WHERE ROWNUM <= 5;

-- Extraer subcadenas
SELECT
    codigo,
    SUBSTR(codigo, 1, 3) AS prefijo_carrera,
    SUBSTR(codigo, 5, 3) AS numero_asignatura
FROM Asignatura;

-- Longitud de cadenas
SELECT
    nombres,
    LENGTH(nombres) AS longitud_nombre
FROM Estudiante
ORDER BY LENGTH(nombres) DESC;

-- Buscar y reemplazar
SELECT
    email,
    REPLACE(email, '@epn.edu.ec', '@gmail.com') AS email_alternativo
FROM Estudiante
WHERE ROWNUM <= 3;

-- Eliminar espacios
SELECT
    ' ' || nombres || ' ' AS con_espacios,
```

```
    TRIM('  ' || nombres || '  ') AS sin_espacios
FROM Estudiante
WHERE ROWNUM <= 3;
```

Ejemplo 5: Funciones de Fecha

```
-- EJEMPLO 5: Trabajar con fechas
-- ================================================

-- Fecha actual
SELECT SYSDATE AS fecha_actual FROM DUAL;
SELECT CURRENT_DATE AS fecha_actual FROM DUAL;
SELECT SYSTIMESTAMP AS timestamp_actual FROM DUAL;

-- Calcular edad de estudiantes
SELECT
    nombres,
    apellidos,
    fecha_nacimiento,
    TRUNC((SYSDATE - fecha_nacimiento) / 365.25) AS edad_anios,
    TRUNC(MONTHS_BETWEEN(SYSDATE, fecha_nacimiento) / 12) AS edad_meses
FROM Estudiante;

-- Formatear fechas
SELECT
    nombres,
```

```
    fecha_nacimiento,
    TO_CHAR(fecha_nacimiento, 'DD/MM/YYYY') AS fecha_formato_1,
    TO_CHAR(fecha_nacimiento, 'Day, DD "de" Month "de" YYYY') AS fecha_formato_2,
    TO_CHAR(fecha_nacimiento, 'DD-Mon-YYYY') AS fecha_formato_3
FROM Estudiante
WHERE ROWNUM <= 5;

-- Extraer partes de fecha
SELECT
    nombres,
    fecha_nacimiento,
    EXTRACT(YEAR FROM fecha_nacimiento) AS anio_nacimiento,
    EXTRACT(MONTH FROM fecha_nacimiento) AS mes_nacimiento,
    EXTRACT(DAY FROM fecha_nacimiento) AS dia_nacimiento
FROM Estudiante
WHERE ROWNUM <= 5;

-- Operaciones con fechas
SELECT
    nombres,
    fecha_ingreso,
    fecha_ingreso + 30 AS fecha_30_dias_despues,
    ADD_MONTHS(fecha_ingreso, 6) AS fecha_6_meses_despues,
    TRUNC(fecha_ingreso, 'MONTH') AS primer_dia_mes
FROM Estudiante
WHERE ROWNUM <= 5;
```

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

Ejemplo 6: Funciones de Conversión

```
-- EJEMPLO 6: Conversión de tipos de datos
-- ================================================

-- TO_CHAR: Convertir a cadena
SELECT
    creditos_aprobados,
    TO_CHAR(creditos_aprobados) AS creditos_texto,
    TO_CHAR(creditos_aprobados, '999') AS creditos_formato
FROM Estudiante
WHERE ROWNUM <= 5;

-- TO_NUMBER: Convertir a número
SELECT
    codigo,
    TO_NUMBER(SUBSTR(codigo, 5, 3)) AS numero_asignatura
FROM Asignatura
WHERE REGEXP_LIKE(SUBSTR(codigo, 5, 3), '^\d+$')
AND ROWNUM <= 5;

-- TO_DATE: Convertir a fecha
SELECT
    TO_DATE('2024-09-01', 'YYYY-MM-DD') AS fecha_inicio_semestre,
    TO_DATE('15/12/2024', 'DD/MM/YYYY') AS fecha_fin_semestre
FROM DUAL;
```


```
-- NVL: Manejar valores NULL
SELECT
    nombres,
    telefono,
    NVL(telefono, 'Sin teléfono') AS telefono_con_default
FROM Estudiante
WHERE ROWNUM <= 10;

-- NVL2: Valor si es NULL o no NULL
SELECT
    nombres,
    telefono,
    NVL2(telefono, 'Tiene teléfono', 'No tiene teléfono') AS tiene_telefono
FROM Estudiante
WHERE ROWNUM <= 10;

-- COALESCE: Primera expresión no nula
SELECT
    nombres,
    telefono,
    email,
    COALESCE(telefono, email, 'Sin contacto') AS contacto_disponible
FROM Estudiante
WHERE ROWNUM <= 5;
```

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

CLÁUSULA WHERE Y OPERADORES

Operadores de Comparación

```sql
-- OPERADORES DE COMPARACIÓN
-- ========================================================

-- Igual (=)
SELECT * FROM Estudiante WHERE estado = 'ACTIVO';

-- Diferente (!= o <>)
SELECT * FROM Estudiante WHERE genero != 'M';
SELECT * FROM Estudiante WHERE genero <> 'M';

-- Mayor que (>)
SELECT * FROM Estudiante WHERE creditos_aprobados > 20;

-- Mayor o igual (>=)
SELECT * FROM Asignatura WHERE creditos >= 5;

-- Menor que (<)
SELECT * FROM Estudiante WHERE creditos_aprobados < 10;

-- Menor o igual (<=)
SELECT * FROM Asignatura WHERE nivel <= 3;
```

Operadores Lógicos

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

```
-- OPERADORES LÓGICOS: AND, OR, NOT
-- ================================================

-- AND: Ambas condiciones deben cumplirse
SELECT nombres, apellidos, codigo_carrera, creditos_aprobados
FROM Estudiante
WHERE codigo_carrera = 'ING-SIS'
AND creditos_aprobados >= 20;

-- OR: Al menos una condición debe cumplirse
SELECT nombres, apellidos, estado
FROM Estudiante
WHERE estado = 'GRADUADO'
OR creditos_aprobados >= 180;

-- NOT: Negación
SELECT nombres, apellidos, genero
FROM Estudiante
WHERE NOT genero = 'M';  -- Equivalente a genero != 'M'

-- Combinación de operadores (usar paréntesis)
SELECT nombres, apellidos, codigo_carrera, creditos_aprobados
FROM Estudiante
WHERE (codigo_carrera = 'ING-SIS' OR codigo_carrera = 'ING-CIV')
AND creditos_aprobados > 15
AND estado = 'ACTIVO';
```

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

Operador BETWEEN

```
-- BETWEEN: Rango de valores
-- ========================================================

-- Estudiantes con créditos entre 10 y 30
SELECT nombres, apellidos, creditos_aprobados
FROM Estudiante
WHERE creditos_aprobados BETWEEN 10 AND 30
ORDER BY creditos_aprobados;

-- Asignaturas de nivel 2 a 5
SELECT codigo, nombre, nivel
FROM Asignatura
WHERE nivel BETWEEN 2 AND 5
ORDER BY nivel, codigo;

-- Estudiantes nacidos en 2002
SELECT nombres, apellidos, fecha_nacimiento
FROM Estudiante
WHERE fecha_nacimiento BETWEEN TO_DATE('2002-01-01', 'YYYY-MM-DD')
                           AND TO_DATE('2002-12-31', 'YYYY-MM-DD')
ORDER BY fecha_nacimiento;

-- NOT BETWEEN
SELECT nombres, creditos_aprobados
```

```
FROM Estudiante
WHERE creditos_aprobados NOT BETWEEN 10 AND 30;
```

Operador IN

```
-- IN: Lista de valores
-- ==================================================

-- Estudiantes de carreras específicas
SELECT nombres, apellidos, codigo_carrera
FROM Estudiante
WHERE codigo_carrera IN ('ING-SIS', 'ING-CIV', 'ING-ELE')
ORDER BY codigo_carrera, apellidos;

-- Estados válidos
SELECT nombres, apellidos, estado
FROM Estudiante
WHERE estado IN ('ACTIVO', 'GRADUADO')
ORDER BY estado, apellidos;

-- NOT IN



SELECT nombres, apellidos, estado
FROM Estudiante
WHERE estado NOT IN ('ACTIVO', 'GRADUADO')
ORDER BY apellidos;
```


```sql
-- IN con subconsulta (ver más adelante)
SELECT codigo, nombre
FROM Asignatura
WHERE codigo_carrera IN (
    SELECT codigo FROM Carrera WHERE duracion_semestres = 10
);

-- LIKE: Búsqueda de patrones
-- ==================================================
-- % = Cualquier secuencia de caracteres (0 o más)
-- _ = Un solo carácter

-- Estudiantes cuyo apellido empieza con 'G'
SELECT nombres, apellidos
FROM Estudiante
WHERE apellidos LIKE 'G%'
ORDER BY apellidos;

-- Estudiantes con 'ana' en el nombre
SELECT nombres, apellidos
FROM Estudiante
WHERE LOWER(nombres) LIKE '%ana%'
ORDER BY nombres;

-- Emails que contienen 'garcia'
SELECT nombres, email
FROM Estudiante
WHERE email LIKE '%garcia%';
```


```sql
-- Cédulas que terminan en '01'
SELECT cedula, nombres, apellidos
FROM Estudiante
WHERE cedula LIKE '%01';

-- Códigos de asignatura con patrón específico (XXX-###)
SELECT codigo, nombre
FROM Asignatura
WHERE codigo LIKE '___-___';  -- 3 caracteres, guion, 3 caracteres

-- NOT LIKE
SELECT nombres, email
FROM Estudiante
WHERE email NOT LIKE '%@epn.edu.ec';

-- LIKE con ESCAPE (para buscar % o _)
-- Si necesita buscar el carácter % literal
SELECT * FROM Tabla WHERE columna LIKE '%\%%' ESCAPE '\';
```

Operador IS NULL

```sql
-- IS NULL / IS NOT NULL
--

-- Estudiantes sin teléfono registrado
```

```sql
SELECT nombres, apellidos, telefono
FROM Estudiante
WHERE telefono IS NULL;

-- Estudiantes con teléfono registrado
SELECT nombres, apellidos, telefono
FROM Estudiante
WHERE telefono IS NOT NULL;

-- Matrículas sin calificaciones
SELECT
    m.id_matricula,
    e.nombres,
    a.nombre AS asignatura
FROM Matricula m
JOIN Estudiante e ON m.cedula_estudiante = e.cedula
JOIN Asignatura a ON m.codigo_asignatura = a.codigo
WHERE m.nota_final IS NULL;

-- IMPORTANTE: No usar = NULL o != NULL
-- ❌ INCORRECTO
SELECT * FROM Estudiante WHERE telefono = NULL;  -- No funciona
-- ✅ CORRECTO
SELECT * FROM Estudiante WHERE telefono IS NULL;
```

Operador exists

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

```
-- EXISTS: Verificar existencia en subconsulta
-- ============================================================

-- Estudiantes que tienen al menos una matrícula
SELECT e.cedula, e.nombres, e.apellidos
FROM Estudiante e
WHERE EXISTS (
    SELECT 1 FROM Matricula m
    WHERE m.cedula_estudiante = e.cedula
);

-- Carreras que tienen estudiantes matriculados
SELECT c.codigo, c.nombre
FROM Carrera c
WHERE EXISTS (
    SELECT 1 FROM Estudiante e
    WHERE e.codigo_carrera = c.codigo
);

-- NOT EXISTS: Estudiantes sin matrículas
SELECT e.cedula, e.nombres, e.apellidos
FROM Estudiante e
WHERE NOT EXISTS (
    SELECT 1 FROM Matricula m
    WHERE m.cedula_estudiante = e.cedula
);
```

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

Ejemplo Completo: Múltiples Condiciones

```
-- EJEMPLO COMPLETO: Combinación de operadores
-- ================================================

-- Consulta compleja con múltiples condiciones
SELECT
    e.cedula,
    e.nombres || ' ' || e.apellidos AS nombre_completo,
    e.codigo_carrera,
    e.creditos_aprobados,
    e.estado,
    TRUNC(MONTHS_BETWEEN(SYSDATE, e.fecha_nacimiento) / 12) AS edad
FROM Estudiante e
WHERE
    -- Condiciones con AND/OR
    (e.codigo_carrera IN ('ING-SIS', 'ING-CIV')
    OR e.creditos_aprobados >= 30)
    -- Rango de créditos
    AND e.creditos_aprobados BETWEEN 10 AND 100
    -- Estado activo
    AND e.estado = 'ACTIVO'
    -- Apellido empieza con letra específica
    AND e.apellidos LIKE 'A%' OR e.apellidos LIKE 'G%'
    -- Tiene teléfono
    AND e.telefono IS NOT NULL
```

```
    -- Mayor de edad
    AND MONTHS_BETWEEN(SYSDATE, e.fecha_nacimiento) / 12 >= 18
ORDER BY
    e.codigo_carrera,
    e.creditos_aprobados DESC,
    e.apellidos;
```

ORDER BY - ORDENAMIENTO

Sintaxis y Ejemplos

```
-- ORDER BY: Ordenar resultados
-- ====================================================

-- Orden ascendente (ASC) - por defecto
SELECT nombres, apellidos, creditos_aprobados
FROM Estudiante
ORDER BY apellidos ASC;

-- Orden descendente (DESC)
SELECT nombres, apellidos, creditos_aprobados
FROM Estudiante
ORDER BY creditos_aprobados DESC;

-- Ordenar por múltiples columnas
SELECT codigo_carrera, apellidos, nombres, creditos_aprobados
```

```
FROM Estudiante
ORDER BY codigo_carrera ASC, creditos_aprobados DESC, apellidos ASC;

-- Ordenar por posición de columna (no recomendado)
SELECT nombres, apellidos, creditos_aprobados
FROM Estudiante
ORDER BY 3 DESC, 2 ASC;  -- 3=creditos_aprobados, 2=apellidos

-- Ordenar por expresión calculada
SELECT
    nombres,
    apellidos,
    creditos_aprobados,
    (creditos_aprobados / 180 * 100) AS porcentaje_avance
FROM Estudiante
WHERE creditos_aprobados > 0
ORDER BY (creditos_aprobados / 180 * 100) DESC;

-- Ordenar por alias
SELECT
    nombres || ' ' || apellidos AS nombre_completo,
    creditos_aprobados
FROM Estudiante
ORDER BY nombre_completo;

-- Ordenar con NULL
-- NULLS FIRST: NULL al inicio
```

```sql
SELECT nombres, telefono
FROM Estudiante
ORDER BY telefono NULLS FIRST;

-- NULLS LAST: NULL al final (por defecto en ASC)
SELECT nombres, telefono
FROM Estudiante
ORDER BY telefono NULLS LAST;

-- Ordenar por fecha
SELECT nombres, apellidos, fecha_nacimiento
FROM Estudiante
ORDER BY fecha_nacimiento DESC;  -- Más jóvenes primero

-- Ordenar resultados de JOIN
SELECT
    e.apellidos,
    e.nombres,
    c.nombre AS carrera
FROM Estudiante e
JOIN Carrera c ON e.codigo_carrera = c.codigo
ORDER BY c.nombre, e.apellidos;
```

FUNCIONES DE AGREGACIÓN

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

Funciones Básicas

```
-- FUNCIONES DE AGREGACIÓN
-- ======================================================

-- COUNT: Contar registros
SELECT COUNT(*) AS total_estudiantes FROM Estudiante;

-- COUNT con columna específica (no cuenta NULL)
SELECT COUNT(telefono) AS estudiantes_con_telefono FROM Estudiante;

-- COUNT DISTINCT: Contar valores únicos
SELECT COUNT(DISTINCT codigo_carrera) AS total_carreras FROM Estudiante;

-- SUM: Suma total
SELECT SUM(creditos) AS total_creditos FROM Asignatura;
SELECT SUM(creditos_aprobados) AS total_creditos_todos_estudiantes FROM Estudiante;

-- AVG: Promedio
SELECT AVG(creditos_aprobados) AS promedio_creditos FROM Estudiante;
SELECT AVG(nota_final) AS promedio_nota_final FROM Matricula WHERE nota_final IS NOT NULL;

-- MAX: Valor máximo
SELECT MAX(creditos_aprobados) AS maximo_creditos FROM Estudiante;
SELECT MAX(nota_final) AS nota_maxima FROM Matricula WHERE nota_final IS NOT NULL;

-- MIN: Valor mínimo
```

```
SELECT MIN(creditos_aprobados) AS minimo_creditos FROM Estudiante WHERE creditos_aprobados > 0;
SELECT MIN(nota_final) AS nota_minima FROM Matricula WHERE nota_final IS NOT NULL;

-- Múltiples agregaciones en una consulta
SELECT

    COUNT(*) AS total_estudiantes,
    COUNT(DISTINCT codigo_carrera) AS total_carreras,
    AVG(creditos_aprobados) AS promedio_creditos,
    MAX(creditos_aprobados) AS maximo_creditos,
    MIN(creditos_aprobados) AS minimo_creditos,
    SUM(creditos_aprobados) AS suma_total_creditos
FROM Estudiante;
```

Ejemplos Prácticos

```
-- EJEMPLOS PRÁCTICOS DE AGREGACIÓN
-- ================================================

-- Estadísticas de calificaciones
SELECT
    COUNT(*) AS total_matriculas,
    COUNT(nota_final) AS matriculas_con_nota,
    ROUND(AVG(nota_final), 2) AS promedio_nota,
    MAX(nota_final) AS nota_maxima,
    MIN(nota_final) AS nota_minima,
    COUNT(CASE WHEN estado = 'APROBADO' THEN 1 END) AS total_aprobados,
```

```
    COUNT(CASE WHEN estado = 'REPROBADO' THEN 1 END) AS total_reprobados
FROM Matricula
WHERE nota_final IS NOT NULL;

-- Estadísticas de asignaturas
SELECT
    COUNT(*) AS total_asignaturas,
    AVG(creditos) AS promedio_creditos,
    SUM(creditos) AS total_creditos,
    AVG(horas_teoria + horas_practica) AS promedio_horas_totales
FROM Asignatura;

-- Edad promedio de estudiantes
SELECT
    ROUND(AVG(TRUNC(MONTHS_BETWEEN(SYSDATE, fecha_nacimiento) / 12)), 1) AS edad_promedio,
    MIN(TRUNC(MONTHS_BETWEEN(SYSDATE, fecha_nacimiento) / 12)) AS edad_minima,
    MAX(TRUNC(MONTHS_BETWEEN(SYSDATE, fecha_nacimiento) / 12)) AS edad_maxima
FROM Estudiante;
-- Distribución de género (usando COUNT con CASE)
SELECT
    COUNT(CASE WHEN genero = 'M' THEN 1 END) AS total_masculino,
    COUNT(CASE WHEN genero = 'F' THEN 1 END) AS total_femenino,
    COUNT(CASE WHEN genero = 'O' THEN 1 END) AS total_otro,
    COUNT(*) AS total_general
FROM Estudiante;
```

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

GROUP BY Y HAVING

GROUP BY Básico

```sql
-- GROUP BY: Agrupar datos
-- ========================================================

-- Contar estudiantes por carrera
SELECT
    codigo_carrera,
    COUNT(*) AS total_estudiantes
FROM Estudiante
GROUP BY codigo_carrera
ORDER BY total_estudiantes DESC;

-- Estudiantes por estado
SELECT
    estado,
    COUNT(*) AS total_estudiantes
FROM Estudiante
GROUP BY estado
ORDER BY total_estudiantes DESC;

-- Promedio de créditos por carrera
SELECT
    codigo_carrera,
    AVG(creditos_aprobados) AS promedio_creditos,
```

```
    MAX(creditos_aprobados) AS maximo_creditos,
    MIN(creditos_aprobados) AS minimo_creditos
FROM Estudiante
GROUP BY codigo_carrera
ORDER BY promedio_creditos DESC;

-- Asignaturas por nivel

SELECT
    nivel,
    COUNT(*) AS total_asignaturas,
    AVG(creditos) AS promedio_creditos
FROM Asignatura
GROUP BY nivel
ORDER BY nivel;
```

GROUP BY con Múltiples Columnas

```
-- GROUP BY con múltiples columnas
-- ==================================================

-- Estudiantes por carrera y estado
SELECT
    codigo_carrera,
    estado,
    COUNT(*) AS total_estudiantes
FROM Estudiante
```

```
GROUP BY codigo_carrera, estado
ORDER BY codigo_carrera, estado;

-- Matrículas por periodo y paralelo
SELECT
    periodo,
    paralelo,
    COUNT(*) AS total_matriculas,
    AVG(nota_final) AS promedio_notas
FROM Matricula
WHERE nota_final IS NOT NULL
GROUP BY periodo, paralelo
ORDER BY periodo, paralelo;

-- Estudiantes por carrera y género
SELECT
    codigo_carrera,
    genero,
    COUNT(*) AS total,
    ROUND(AVG(creditos_aprobados), 2) AS promedio_creditos
FROM Estudiante
GROUP BY codigo_carrera, genero
ORDER BY codigo_carrera, genero;
```

HAVING - Filtrar Grupos

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

```sql
-- HAVING: Filtrar después de agrupar
-- ============================================================

-- Carreras con más de 5 estudiantes
SELECT
    codigo_carrera,
    COUNT(*) AS total_estudiantes
FROM Estudiante
GROUP BY codigo_carrera
HAVING COUNT(*) > 5
ORDER BY total_estudiantes DESC;

-- Asignaturas por nivel con promedio de créditos >= 4
SELECT
    nivel,
    COUNT(*) AS total_asignaturas,
    AVG(creditos) AS promedio_creditos
FROM Asignatura
GROUP BY nivel
HAVING AVG(creditos) >= 4
ORDER BY nivel;

-- Estudiantes con promedio de notas >= 8.0
SELECT
    m.cedula_estudiante,
    e.nombres || ' ' || e.apellidos AS nombre_completo,
    COUNT(*) AS total_materias,
```

```
    ROUND(AVG(m.nota_final), 2) AS promedio_notas
FROM Matricula m
JOIN Estudiante e ON m.cedula_estudiante = e.cedula
WHERE m.nota_final IS NOT NULL
GROUP BY m.cedula_estudiante, e.nombres, e.apellidos
HAVING AVG(m.nota_final) >= 8.0
ORDER BY promedio_notas DESC;

-- Diferencia WHERE vs HAVING
-- WHERE filtra ANTES de agrupar, HAVING filtra DESPUÉS de agrupar

-- Estudiantes activos, agrupados por carrera, con más de 3 estudiantes
SELECT
    codigo_carrera,
    COUNT(*) AS total_estudiantes
FROM Estudiante
WHERE estado = 'ACTIVO'  -- WHERE filtra antes de GROUP BY
GROUP BY codigo_carrera
HAVING COUNT(*) > 3      -- HAVING filtra después de GROUP BY
ORDER BY total_estudiantes DESC;
```

Ejemplos Avanzados GROUP BY

```
-- EJEMPLOS AVANZADOS CON GROUP BY
-- ================================================
```

```sql
-- Top 5 asignaturas con más matrículas
SELECT
    a.codigo,
    a.nombre,
    COUNT(m.id_matricula) AS total_matriculas
FROM Asignatura a
LEFT JOIN Matricula m ON a.codigo = m.codigo_asignatura
GROUP BY a.codigo, a.nombre
HAVING COUNT(m.id_matricula) > 0
ORDER BY total_matriculas DESC
FETCH FIRST 5 ROWS ONLY;

-- Docentes con su carga académica
SELECT
    d.cedula,
    d.nombres || ' ' || d.apellidos AS nombre_docente,
    d.tipo_contrato,
    COUNT(DISTINCT m.codigo_asignatura) AS total_asignaturas_distintas,
    COUNT(m.id_matricula) AS total_estudiantes_atendidos,
    ROUND(AVG(m.nota_final), 2) AS promedio_notas_estudiantes
FROM Docente d
LEFT JOIN Matricula m ON d.cedula = m.cedula_docente
GROUP BY d.cedula, d.nombres, d.apellidos, d.tipo_contrato
ORDER BY total_estudiantes_atendidos DESC;

-- Análisis por periodo académico
SELECT
```

```
    periodo,
    COUNT(DISTINCT cedula_estudiante) AS estudiantes_unicos,
    COUNT(DISTINCT codigo_asignatura) AS asignaturas_ofertadas,
    COUNT(*) AS total_matriculas,
    COUNT(CASE WHEN estado = 'APROBADO' THEN 1 END) AS aprobados,
    COUNT(CASE WHEN estado = 'REPROBADO' THEN 1 END) AS reprobados,
    ROUND(COUNT(CASE WHEN estado = 'APROBADO' THEN 1 END) * 100.0 /
        NULLIF(COUNT(CASE WHEN estado IN ('APROBADO','REPROBADO') THEN 1 END), 0), 2) AS porcentaje_aprobacion
FROM Matricula
WHERE estado IN ('APROBADO', 'REPROBADO')
GROUP BY periodo
ORDER BY periodo DESC;
```

JOINS - UNIONES DE TABLAS

INNER JOIN

```
-- INNER JOIN: Registros que coinciden en ambas tablas
-- ====================================================

-- Estudiantes con su carrera
SELECT
    e.cedula,
    e.nombres,
    e.apellidos,
    c.codigo AS codigo_carrera,
```

```
    c.nombre AS nombre_carrera
FROM Estudiante e
INNER JOIN Carrera c ON e.codigo_carrera = c.codigo
ORDER BY c.nombre, e.apellidos;

-- Matrículas con información completa
SELECT
    m.id_matricula,
    e.nombres || ' ' || e.apellidos AS estudiante,
    a.nombre AS asignatura,
    d.nombres || ' ' || d.apellidos AS docente,
    m.periodo,
    m.nota_final,
    m.estado
FROM Matricula m
INNER JOIN Estudiante e ON m.cedula_estudiante = e.cedula
INNER JOIN Asignatura a ON m.codigo_asignatura = a.codigo
INNER JOIN Docente d ON m.cedula_docente = d.cedula
WHERE m.periodo = '2024-2S'
ORDER BY m.id_matricula;
-- Asignaturas con sus prerrequisitos
SELECT
    a1.codigo AS asignatura_codigo,
    a1.nombre AS asignatura_nombre,
    a2.codigo AS prerrequisito_codigo,
    a2.nombre AS prerrequisito_nombre
FROM Prerrequisito p
```

```
INNER JOIN Asignatura a1 ON p.codigo_asignatura = a1.codigo
INNER JOIN Asignatura a2 ON p.codigo_prerrequisito = a2.codigo
ORDER BY a1.codigo;

-- JOIN con condiciones adicionales
SELECT
    e.nombres,
    e.apellidos,
    c.nombre AS carrera,
    e.creditos_aprobados
FROM Estudiante e
INNER JOIN Carrera c ON e.codigo_carrera = c.codigo
WHERE e.creditos_aprobados >= 30
AND e.estado = 'ACTIVO'
ORDER BY e.creditos_aprobados DESC;
```

LEFT JOIN (LEFT OUTER JOIN)

```
-- LEFT JOIN: Todos los registros de la tabla izquierda
-- ================================================

-- Todos los estudiantes con o sin matrículas
SELECT
    e.cedula,
    e.nombres,
    e.apellidos,
```

```
    COUNT(m.id_matricula) AS total_matriculas
FROM Estudiante e
LEFT JOIN Matricula m ON e.cedula = m.cedula_estudiante
GROUP BY e.cedula, e.nombres, e.apellidos
ORDER BY total_matriculas DESC, e.apellidos;

-- Estudiantes sin matrículas (usando LEFT JOIN y WHERE NULL)
SELECT
    e.cedula,
    e.nombres,
    e.apellidos,
    e.estado
FROM Estudiante e
LEFT JOIN Matricula m ON e.cedula = m.cedula_estudiante
WHERE m.id_matricula IS NULL
ORDER BY e.apellidos;

-- Todas las asignaturas con conteo de matrículas
SELECT
    a.codigo,
    a.nombre,
    a.nivel,
    a.creditos,
    COUNT(m.id_matricula) AS total_matriculas
FROM Asignatura a
LEFT JOIN Matricula m ON a.codigo = m.codigo_asignatura
GROUP BY a.codigo, a.nombre, a.nivel, a.creditos
```

```
ORDER BY total_matriculas DESC, a.codigo;

-- Asignaturas sin matrículas
SELECT
    a.codigo,
    a.nombre,
    a.nivel
FROM Asignatura a
LEFT JOIN Matricula m ON a.codigo = m.codigo_asignatura
WHERE m.id_matricula IS NULL
ORDER BY a.nivel, a.codigo;

-- Docentes con su carga (incluyendo docentes sin asignaciones)
SELECT
    d.cedula,
    d.nombres || ' ' || d.apellidos AS docente,
    d.tipo_contrato,
    COUNT(m.id_matricula) AS estudiantes_asignados
FROM Docente d
LEFT JOIN Matricula m ON d.cedula = m.cedula_docente
GROUP BY d.cedula, d.nombres, d.apellidos, d.tipo_contrato
ORDER BY estudiantes_asignados DESC;
```

RIGHT JOIN (RIGHT OUTER JOIN)

```
-- RIGHT JOIN: Todos los registros de la tabla derecha
-- ==================================================
```

```sql
-- Todas las carreras con conteo de estudiantes
SELECT
    c.codigo,
    c.nombre AS carrera,
    COUNT(e.cedula) AS total_estudiantes
FROM Estudiante e
RIGHT JOIN Carrera c ON e.codigo_carrera = c.codigo
GROUP BY c.codigo, c.nombre
ORDER BY total_estudiantes DESC;

-- Carreras sin estudiantes
SELECT
    c.codigo,
    c.nombre,
    c.facultad
FROM Estudiante e
RIGHT JOIN Carrera c ON e.codigo_carrera = c.codigo
WHERE e.cedula IS NULL;

-- NOTA: RIGHT JOIN es menos común que LEFT JOIN
-- Normalmente se usa LEFT JOIN reorganizando las tablas
-- Estos dos son equivalentes:

-- Con RIGHT JOIN
SELECT c.nombre, COUNT(e.cedula)
FROM Estudiante e
RIGHT JOIN Carrera c ON e.codigo_carrera = c.codigo
```

```
-- Con LEFT JOIN (más común)
SELECT c.nombre, COUNT(e.cedula)
FROM Carrera c
LEFT JOIN Estudiante e ON c.codigo = e.codigo_carrera
GROUP BY c.nombre;
```

FULL OUTER JOIN

```
-- FULL OUTER JOIN: Todos los registros de ambas tablas
-- ====================================================

-- Ejemplo conceptual (menos común en práctica)
-- Todos los estudiantes y todas las carreras
SELECT
    e.cedula,
    e.nombres,
    e.apellidos,
    c.codigo AS codigo_carrera,
    c.nombre AS nombre_carrera
FROM Estudiante e
FULL OUTER JOIN Carrera c ON e.codigo_carrera = c.codigo
ORDER BY c.nombre, e.apellidos;

-- Encontrar estudiantes sin carrera Y carreras sin estudiantes
SELECT
    e.cedula,
```

```
    e.nombres,
    c.codigo AS codigo_carrera,
    c.nombre AS nombre_carrera,
    CASE
        WHEN e.cedula IS NULL THEN 'Carrera sin estudiantes'
        WHEN c.codigo IS NULL THEN 'Estudiante sin carrera'
        ELSE 'Ambos tienen match'
    END AS estado_relacion
FROM Estudiante e
FULL OUTER JOIN Carrera c ON e.codigo_carrera = c.codigo
WHERE e.cedula IS NULL OR c.codigo IS NULL;
```

CROSS JOIN (Producto Cartesiano)

```
-- CROSS JOIN: Todas las combinaciones posibles
-- ============================================================

-- ADVERTENCIA: Genera muchas filas (n × m)

-- Ejemplo: Combinar todas las asignaturas con todos los docentes
-- (Útil para generar horarios posibles)
SELECT
    a.codigo AS codigo_asignatura,
    a.nombre AS asignatura,
    d.cedula,
    d.nombres || ' ' || d.apellidos AS docente
```

```
FROM Asignatura a
CROSS JOIN Docente d
WHERE a.codigo_carrera = 'ING-SIS'
AND ROWNUM <= 20;  -- Limitar resultados

-- Combinaciones de periodos y paralelos
SELECT
    periodo,
    paralelo
FROM (SELECT '2024-1S' AS periodo FROM DUAL UNION ALL SELECT '2024-2S' FROM DUAL)
CROSS JOIN (SELECT 'A' AS paralelo FROM DUAL UNION ALL SELECT 'B' FROM DUAL);
```

SELF JOIN

```
-- ============================================================
-- SELF JOIN: Unir tabla consigo misma
-- ============================================================

-- Encontrar pares de estudiantes de la misma carrera
SELECT
    e1.nombres || ' ' || e1.apellidos AS estudiante1,
    e2.nombres || ' ' || e2.apellidos AS estudiante2,
    e1.codigo_carrera
FROM Estudiante e1
JOIN Estudiante e2 ON e1.codigo_carrera = e2.codigo_carrera
WHERE e1.cedula < e2.cedula  -- Evitar duplicados y comparar consigo mismo
AND ROWNUM <= 10;
```

```sql
-- Asignaturas que comparten el mismo prerrequisito
SELECT
    p1.codigo_asignatura AS asignatura1,
    p2.codigo_asignatura AS asignatura2,
    p1.codigo_prerrequisito AS prerrequisito_comun
FROM Prerrequisito p1
JOIN Prerrequisito p2 ON p1.codigo_prerrequisito = p2.codigo_prerrequisito
WHERE p1.codigo_asignatura < p2.codigo_asignatura;
```

Ejemplos Complejos con Múltiples JOINs

```sql
-- CONSULTAS COMPLEJAS CON MÚLTIPLES JOINS
-- ================================================

-- Reporte completo de matrículas
SELECT
    m.id_matricula,
    e.cedula,
    e.nombres || ' ' || e.apellidos AS estudiante,
    c.nombre AS carrera,
    a.codigo AS codigo_asignatura,
    a.nombre AS asignatura,
    a.creditos,
    d.nombres || ' ' || d.apellidos AS docente,
    m.periodo,
    m.paralelo,
```

```
    m.nota_parcial1,
    m.nota_parcial2,
    m.nota_final,
    m.estado,
    CASE
        WHEN m.nota_final >= 9.0 THEN 'Excelente'
        WHEN m.nota_final >= 8.0 THEN 'Muy Bueno'
        WHEN m.nota_final >= 7.0 THEN 'Bueno'
        ELSE 'Insuficiente'
    END AS calificacion_cualitativa
FROM Matricula m
INNER JOIN Estudiante e ON m.cedula_estudiante = e.cedula
INNER JOIN Carrera c ON e.codigo_carrera = c.codigo
INNER JOIN Asignatura a ON m.codigo_asignatura = a.codigo
INNER JOIN Docente d ON m.cedula_docente = d.cedula
WHERE m.periodo = '2024-2S'
ORDER BY m.id_matricula;

-- Estudiantes con asignaturas pendientes por prerrequisitos
SELECT DISTINCT
    e.cedula,
    e.nombres || ' ' || e.apellidos AS estudiante,
    a.codigo AS asignatura_bloqueada,
    a.nombre AS nombre_asignatura,
    apre.codigo AS prerrequisito_codigo,
    apre.nombre AS prerrequisito_nombre
FROM Estudiante e
```

```
CROSS JOIN Asignatura a
INNER JOIN Prerrequisito p ON a.codigo = p.codigo_asignatura
INNER JOIN Asignatura apre ON p.codigo_prerrequisito = apre.codigo
LEFT JOIN Matricula m ON e.cedula = m.cedula_estudiante
   AND apre.codigo = m.codigo_asignatura
   AND m.estado = 'APROBADO'
WHERE e.codigo_carrera = a.codigo_carrera
AND m.id_matricula IS NULL
AND ROWNUM <= 20
ORDER BY estudiante;

-- Top estudiantes por carrera
SELECT
   c.nombre AS carrera,
   e.nombres || ' ' || e.apellidos AS estudiante,
   e.creditos_aprobados,
   ROUND(AVG(m.nota_final), 2) AS promedio_general,
   COUNT(CASE WHEN m.estado = 'APROBADO' THEN 1 END) AS materias_aprobadas
FROM Carrera c
INNER JOIN Estudiante e ON c.codigo = e.codigo_carrera
LEFT JOIN Matricula m ON e.cedula = m.cedula_estudiante AND m.nota_final IS NOT NULL
WHERE e.estado = 'ACTIVO'
GROUP BY c.nombre, e.cedula, e.nombres, e.apellidos, e.creditos_aprobados
HAVING COUNT(CASE WHEN m.estado = 'APROBADO' THEN 1 END) > 0
ORDER BY c.nombre, promedio_general DESC, e.creditos_aprobados DESC;
```

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

SUBCONSULTAS (SUBQUERIES)

Subconsultas en WHERE

```sql
-- SUBCONSULTAS EN WHERE
-- ====================================================

-- Estudiantes de la carrera con más estudiantes matriculados
SELECT nombres, apellidos, codigo_carrera
FROM Estudiante
WHERE codigo_carrera = (
    SELECT codigo_carrera
    FROM Estudiante
    GROUP BY codigo_carrera
    ORDER BY COUNT(*) DESC
    FETCH FIRST 1 ROW ONLY
);

-- Estudiantes con más créditos que el promedio
SELECT nombres, apellidos, creditos_aprobados
FROM Estudiante
WHERE creditos_aprobados > (
    SELECT AVG(creditos_aprobados)
    FROM Estudiante
)
ORDER BY creditos_aprobados DESC;
```

```sql
-- Asignaturas que son prerrequisito de otras
SELECT codigo, nombre
FROM Asignatura
WHERE codigo IN (
    SELECT codigo_prerrequisito
    FROM Prerrequisito
)
ORDER BY codigo;

-- Estudiantes que NO tienen matrículas
SELECT cedula, nombres, apellidos
FROM Estudiante
WHERE cedula NOT IN (
    SELECT DISTINCT cedula_estudiante
    FROM Matricula
);

-- Docentes que imparten asignaturas de nivel >= 5
SELECT DISTINCT d.cedula, d.nombres, d.apellidos
FROM Docente d
WHERE d.cedula IN (
    SELECT DISTINCT m.cedula_docente
    FROM Matricula m
    INNER JOIN Asignatura a ON m.codigo_asignatura = a.codigo
    WHERE a.nivel >= 5
);
```

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

Subconsultas en SELECT

```
-- SUBCONSULTAS EN SELECT (columnas calculadas)
-- ============================================================

-- Estudiantes con total de matrículas
SELECT
    e.cedula,
    e.nombres,
    e.apellidos,
    (SELECT COUNT(*)
     FROM Matricula m
     WHERE m.cedula_estudiante = e.cedula) AS total_matriculas,
    (SELECT COUNT(*)
     FROM Matricula m
     WHERE m.cedula_estudiante = e.cedula
     AND m.estado = 'APROBADO') AS materias_aprobadas
FROM Estudiante e
ORDER BY total_matriculas DESC;

-- Carreras con estadísticas
SELECT
    c.codigo,
    c.nombre,
    (SELECT COUNT(*)
     FROM Estudiante e
```

```
    WHERE e.codigo_carrera = c.codigo) AS total_estudiantes,
   (SELECT COUNT(*)
    FROM Asignatura a
    WHERE a.codigo_carrera = c.codigo) AS total_asignaturas,
   (SELECT AVG(creditos_aprobados)
    FROM Estudiante e
    WHERE e.codigo_carrera = c.codigo) AS promedio_creditos
FROM Carrera c
ORDER BY total_estudiantes DESC;

-- Asignaturas con información de matrículas
SELECT
   a.codigo,
   a.nombre,
   a.creditos,
   (SELECT COUNT(*)
    FROM Matricula m
    WHERE m.codigo_asignatura = a.codigo) AS veces_matriculada,
   (SELECT AVG(nota_final)
    FROM Matricula m
    WHERE m.codigo_asignatura = a.codigo
    AND m.nota_final IS NOT NULL) AS promedio_notas
FROM Asignatura a
WHERE a.codigo_carrera = 'ING-SIS'
ORDER BY veces_matriculada DESC;
```

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

Subconsultas en FROM (Derived Tables)

```sql
-- SUBCONSULTAS EN FROM (tablas derivadas)
-- ================================================

-- Estadísticas por carrera usando tabla derivada
SELECT
    stats.codigo_carrera,
    c.nombre AS nombre_carrera,
    stats.total_estudiantes,
    stats.promedio_creditos
FROM (
    SELECT
        codigo_carrera,
        COUNT(*) AS total_estudiantes,
        ROUND(AVG(creditos_aprobados), 2) AS promedio_creditos
    FROM Estudiante
    WHERE estado = 'ACTIVO'
    GROUP BY codigo_carrera
) stats
INNER JOIN Carrera c ON stats.codigo_carrera = c.codigo
ORDER BY stats.total_estudiantes DESC;

-- Top 5 estudiantes por promedio
SELECT
    notas.cedula,
    e.nombres || ' ' || e.apellidos AS estudiante,
```

```
    notas.promedio,
    notas.total_materias
FROM (
    SELECT
        cedula_estudiante AS cedula,
        ROUND(AVG(nota_final), 2) AS promedio,
        COUNT(*) AS total_materias
    FROM Matricula
    WHERE nota_final IS NOT NULL
    GROUP BY cedula_estudiante
    HAVING COUNT(*) >= 3
) notas
INNER JOIN Estudiante e ON notas.cedula = e.cedula
ORDER BY notas.promedio DESC
FETCH FIRST 5 ROWS ONLY;

-- Asignaturas populares (con más de N matrículas)
SELECT
    popular.codigo,
    a.nombre,
    popular.total_matriculas,
    popular.promedio_nota
FROM (
    SELECT
        codigo_asignatura AS codigo,
        COUNT(*) AS total_matriculas,
        ROUND(AVG(nota_final), 2) AS promedio_nota
```

```
    FROM Matricula
    WHERE nota_final IS NOT NULL
    GROUP BY codigo_asignatura
    HAVING COUNT(*) >= 3
) popular
INNER JOIN Asignatura a ON popular.codigo = a.codigo
ORDER BY popular.total_matriculas DESC;
```

Subconsultas Correlacionadas

```
-- SUBCONSULTAS CORRELACIONADAS
-- ============================================================

-- Estudiantes con nota superior al promedio de su carrera
SELECT
    e.cedula,
    e.nombres,
    e.apellidos,
    e.codigo_carrera,
    (SELECT ROUND(AVG(m2.nota_final), 2)
     FROM Matricula m2
     INNER JOIN Estudiante e2 ON m2.cedula_estudiante = e2.cedula
     WHERE e2.codigo_carrera = e.codigo_carrera
     AND m2.nota_final IS NOT NULL) AS promedio_carrera,
    (SELECT ROUND(AVG(m1.nota_final), 2)
     FROM Matricula m1
     WHERE m1.cedula_estudiante = e.cedula
     AND m1.nota_final IS NOT NULL) AS promedio_estudiante
```

```
FROM Estudiante e
WHERE EXISTS (
   SELECT 1 FROM Matricula m WHERE m.cedula_estudiante = e.cedula
)
AND (SELECT AVG(m1.nota_final)
    FROM Matricula m1
    WHERE m1.cedula_estudiante = e.cedula
   AND m1.nota_final IS NOT NULL) >
   (SELECT AVG(m2.nota_final)
    FROM Matricula m2
    INNER JOIN Estudiante e2 ON m2.cedula_estudiante = e2.cedula
    WHERE e2.codigo_carrera = e.codigo_carrera

   AND m2.nota_final IS NOT NULL)
ORDER BY promedio_estudiante DESC;

-- Asignaturas con nota promedio superior a la general
SELECT
   a.codigo,
   a.nombre,
   (SELECT ROUND(AVG(nota_final), 2)
    FROM Matricula
    WHERE codigo_asignatura = a.codigo
    AND nota_final IS NOT NULL) AS promedio_asignatura
FROM Asignatura a
WHERE (SELECT AVG(nota_final)
      FROM Matricula
```

```
    WHERE codigo_asignatura = a.codigo
    AND nota_final IS NOT NULL) >
    (SELECT AVG(nota_final)
    FROM Matricula
    WHERE nota_final IS NOT NULL)
ORDER BY promedio_asignatura DESC;
```

Subconsultas con ANY, ALL

```
-- ANY, SOME, ALL con subconsultas
-- ================================================

-- Estudiantes con más créditos que CUALQUIER estudiante de ING-CIV
SELECT nombres, apellidos, codigo_carrera, creditos_aprobados
FROM Estudiante
WHERE creditos_aprobados > ANY (
    SELECT creditos_aprobados
    FROM Estudiante
    WHERE codigo_carrera = 'ING-CIV'
)
AND codigo_carrera != 'ING-CIV'
ORDER BY creditos_aprobados DESC;

-- Estudiantes con más créditos que TODOS los estudiantes de ING-CIV
SELECT nombres, apellidos, codigo_carrera, creditos_aprobados
FROM Estudiante
```

```sql
WHERE creditos_aprobados > ALL (
    SELECT creditos_aprobados
    FROM Estudiante
    WHERE codigo_carrera = 'ING-CIV'
)
ORDER BY creditos_aprobados DESC;

-- Asignaturas con créditos >= a cualquier asignatura de nivel 1
SELECT codigo, nombre, nivel, creditos
FROM Asignatura
WHERE creditos >= ANY (
    SELECT creditos
    FROM Asignatura
    WHERE nivel = 1
)
AND nivel > 1
ORDER BY nivel, creditos DESC;
```

OPERADORES DE CONJUNTOS

UNION y UNION ALL

```sql
-- UNION: Combinar resultados (sin duplicados)
-- ============================================

-- Todos los emails de estudiantes y docentes
```

```sql
SELECT email, 'Estudiante' AS tipo
FROM Estudiante
UNION
SELECT email, 'Docente' AS tipo
FROM Docente
ORDER BY email;

-- UNION ALL: Incluir duplicados (más rápido)
SELECT email
FROM Estudiante
UNION ALL
SELECT email
FROM Docente;

-- Combinar diferentes consultas con la misma estructura
SELECT cedula AS identificacion, nombres, apellidos, 'Estudiante' AS rol
FROM Estudiante
WHERE estado = 'ACTIVO'
UNION
SELECT cedula, nombres, apellidos, 'Docente' AS rol
FROM Docente
WHERE tipo_contrato = 'TIEMPO_COMPLETO'
ORDER BY apellidos;

-- Lista de todos los códigos (carreras y asignaturas)
SELECT codigo AS codigo, nombre AS descripcion, 'Carrera' AS tipo
FROM Carrera
```

```
UNION
SELECT codigo, nombre, 'Asignatura' AS tipo
FROM Asignatura
ORDER BY tipo, codigo;
```

INTERSECT

```
-- INTERSECT: Elementos comunes entre conjuntos
-- ============================================================

-- Estudiantes que también son docentes (poco común, pero para ejemplo)
SELECT cedula, nombres, apellidos
FROM Estudiante
INTERSECT
SELECT cedula, nombres, apellidos
FROM Docente;

-- Códigos que aparecen tanto en Asignatura como en Prerrequisito
SELECT codigo
FROM Asignatura
INTERSECT
SELECT codigo_asignatura
FROM Prerrequisito;

-- Estudiantes matriculados en ambos periodos
SELECT cedula_estudiante AS cedula
```

```
FROM Matricula
WHERE periodo = '2024-1S'
INTERSECT
SELECT cedula_estudiante
FROM Matricula
WHERE periodo = '2024-2S';
```

MINUS (EXCEPT en otros SGBD)

```
-- MINUS: Elementos en el primer conjunto pero no en el segundo
-- ============================================================

-- Estudiantes que NO tienen matrículas
SELECT cedula, nombres, apellidos
FROM Estudiante
MINUS
SELECT DISTINCT e.cedula, e.nombres, e.apellidos
FROM Estudiante e
INNER JOIN Matricula m ON e.cedula = m.cedula_estudiante;

-- Asignaturas que NO son prerrequisito de ninguna otra
SELECT codigo, nombre
FROM Asignatura
MINUS
SELECT a.codigo, a.nombre
FROM Asignatura a
INNER JOIN Prerrequisito p ON a.codigo = p.codigo_prerrequisito;
```

```
-- Docentes que NO han impartido clases
SELECT cedula, nombres, apellidos
FROM Docente
MINUS
SELECT DISTINCT d.cedula, d.nombres, d.apellidos
FROM Docente d
INNER JOIN Matricula m ON d.cedula = m.cedula_docente;

-- Estudiantes de ING-SIS que NO están matriculados en BD-501
SELECT cedula, nombres, apellidos
FROM Estudiante
WHERE codigo_carrera = 'ING-SIS'
MINUS
SELECT e.cedula, e.nombres, e.apellidos
FROM Estudiante e
INNER JOIN Matricula m ON e.cedula = m.cedula_estudiante
WHERE m.codigo_asignatura = 'BD-501';
```

Combinación de Operadores

```
-- COMBINACIÓN DE OPERADORES DE CONJUNTOS
-- ==========================================================

-- Estudiantes activos o graduados, pero no retirados
(
```

```
    SELECT cedula, nombres, apellidos, estado
    FROM Estudiante
    WHERE estado = 'ACTIVO'
    UNION
    SELECT cedula, nombres, apellidos, estado
    FROM Estudiante
    WHERE estado = 'GRADUADO'
)
MINUS
SELECT cedula, nombres, apellidos, estado
FROM Estudiante
WHERE estado = 'RETIRADO'
ORDER BY apellidos;

-- Todos los involucrados en el sistema (estudiantes, docentes)
-- que tienen email @epn.edu.ec
SELECT 'E-' || cedula AS id, nombres, apellidos, email, 'Estudiante' AS tipo
FROM Estudiante
WHERE email LIKE '%@epn.edu.ec'
UNION
SELECT 'D-' || cedula AS id, nombres, apellidos, email, 'Docente' AS tipo
FROM Docente
WHERE email LIKE '%@epn.edu.ec'
ORDER BY apellidos;
```

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

CONSULTAS AVANZADAS

WITH (Common Table Expressions - CTE)

```
-- WITH: Consultas temporales nombradas (CTE)
-- ==================================================

-- Ejemplo básico de CTE
WITH PromediosCarrera AS (
    SELECT
        codigo_carrera,
        ROUND(AVG(creditos_aprobados), 2) AS promedio_creditos,
        COUNT(*) AS total_estudiantes
    FROM Estudiante
    GROUP BY codigo_carrera
)
SELECT
    c.nombre AS carrera,
    pc.total_estudiantes,
    pc.promedio_creditos
FROM PromediosCarrera pc
INNER JOIN Carrera c ON pc.codigo_carrera = c.codigo
WHERE pc.total_estudiantes >= 3
ORDER BY pc.promedio_creditos DESC;

-- Múltiples CTEs
WITH
```

```
EstudiantesActivos AS (
    SELECT cedula, nombres, apellidos, codigo_carrera, creditos_aprobados
    FROM Estudiante
    WHERE estado = 'ACTIVO'
),
MatriculasPorEstudiante AS (
    SELECT
        cedula_estudiante,
        COUNT(*) AS total_matriculas,
        ROUND(AVG(nota_final), 2) AS promedio_notas
    FROM Matricula
    WHERE nota_final IS NOT NULL
    GROUP BY cedula_estudiante
)
SELECT
    ea.cedula,
    ea.nombres || ' ' || ea.apellidos AS estudiante,
    ea.codigo_carrera,
    ea.creditos_aprobados,
    NVL(mpe.total_matriculas, 0) AS total_matriculas,
    NVL(mpe.promedio_notas, 0) AS promedio_notas
FROM EstudiantesActivos ea
LEFT JOIN MatriculasPorEstudiante mpe ON ea.cedula = mpe.cedula_estudiante
ORDER BY promedio_notas DESC, creditos_aprobados DESC;

-- CTE recursiva (ejemplo: jerarquía de prerrequisitos)
WITH RECURSIVE PrerrequisitosCadena (asignatura, prerrequisito, nivel) AS (
```

```
    -- Caso base: asignaturas sin prerrequisitos
    SELECT codigo, CAST(NULL AS VARCHAR2(10)), 0
    FROM Asignatura
    WHERE codigo NOT IN (SELECT codigo_asignatura FROM Prerrequisito)

    UNION ALL

    -- Caso recursivo: agregar prerrequisitos
    SELECT p.codigo_asignatura, p.codigo_prerrequisito, pc.nivel + 1
    FROM Prerrequisito p
    INNER JOIN PrerrequisitosCadena pc ON p.codigo_prerrequisito = pc.asignatura
    WHERE pc.nivel < 5  -- Limitar profundidad
)
SELECT DISTINCT asignatura, prerrequisito, nivel
FROM PrerrequisitosCadena
ORDER BY nivel, asignatura;
```

Consultas Analíticas (Window Functions)

```
-- ============================================================
-- FUNCIONES DE VENTANA (WINDOW FUNCTIONS)
-- ============================================================

-- ROW_NUMBER: Numerar filas
SELECT
    ROW_NUMBER() OVER (ORDER BY creditos_aprobados DESC) AS ranking,
```

```
    nombres,
    apellidos,
    codigo_carrera,
    creditos_aprobados
FROM Estudiante
WHERE estado = 'ACTIVO'
FETCH FIRST 10 ROWS ONLY;

-- RANK y DENSE_RANK
SELECT
    nombres,
    apellidos,
    creditos_aprobados,
    RANK() OVER (ORDER BY creditos_aprobados DESC) AS rank_con_gaps,
    DENSE_RANK() OVER (ORDER BY creditos_aprobados DESC) AS rank_sin_gaps
FROM Estudiante
WHERE estado = 'ACTIVO'
FETCH FIRST 15 ROWS ONLY;

-- PARTITION BY: Ranking por carrera
SELECT
    codigo_carrera,
    nombres,
    apellidos,
    creditos_aprobados,
    RANK() OVER (PARTITION BY codigo_carrera ORDER BY creditos_aprobados DESC) AS rank_en_carrera
FROM Estudiante
```

```
WHERE estado = 'ACTIVO'
ORDER BY codigo_carrera, rank_en_carrera;

-- Mejor estudiante por carrera (usando ROW_NUMBER)
SELECT *
FROM (
    SELECT
        codigo_carrera,
        nombres || ' ' || apellidos AS estudiante,
        creditos_aprobados,
        ROW_NUMBER() OVER (PARTITION BY codigo_carrera ORDER BY creditos_aprobados DESC) AS rn
    FROM Estudiante
    WHERE estado = 'ACTIVO'
)
WHERE rn = 1
ORDER BY codigo_carrera;

-- LAG y LEAD: Acceder a filas anteriores/siguientes
SELECT
    nombres,
    apellidos,
    creditos_aprobados,
    LAG(creditos_aprobados, 1) OVER (ORDER BY creditos_aprobados) AS creditos_anterior,
    LEAD(creditos_aprobados, 1) OVER (ORDER BY creditos_aprobados) AS creditos_siguiente,
    creditos_aprobados - LAG(creditos_aprobados, 1) OVER (ORDER BY creditos_aprobados) AS diferencia
FROM Estudiante
WHERE estado = 'ACTIVO'
```

```
AND creditos_aprobados > 0
ORDER BY creditos_aprobados;


-- SUM con OVER: Totales acumulados
SELECT
    codigo_carrera,
    nombres,
    apellidos,
    creditos_aprobados,
    SUM(creditos_aprobados) OVER (PARTITION BY codigo_carrera ORDER BY apellidos) AS creditos_acumulados,
    SUM(creditos_aprobados) OVER (PARTITION BY codigo_carrera) AS total_carrera
FROM Estudiante
WHERE estado = 'ACTIVO'
ORDER BY codigo_carrera, apellidos;

-- AVG con ventanas móviles
SELECT
    periodo,
    AVG(nota_final) AS promedio_periodo,
    AVG(AVG(nota_final)) OVER (ORDER BY periodo ROWS BETWEEN 1 PRECEDING AND CURRENT ROW) AS promedio_movil_2periodos
FROM Matricula
WHERE nota_final IS NOT NULL
GROUP BY periodo
ORDER BY periodo;
```

PIVOT y UNPIVOT

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

```
-- PIVOT: Convertir filas en columnas
-- ==================================================

-- Estudiantes por carrera y género (matriz)
SELECT *
FROM (
    SELECT codigo_carrera, genero
    FROM Estudiante
)
PIVOT (
    COUNT(*)
    FOR genero IN ('M' AS Masculino, 'F' AS Femenino, 'O' AS Otro)
)
ORDER BY codigo_carrera;

-- Matrículas por periodo y estado
SELECT *
FROM (
    SELECT periodo, estado
    FROM Matricula
)
PIVOT (
    COUNT(*)
    FOR estado IN (
        'CURSANDO' AS Cursando,
        'APROBADO' AS Aprobado,
```

```
        'REPROBADO' AS Reprobado,
        'RETIRADO' AS Retirado
    )
)
ORDER BY periodo DESC;


-- UNPIVOT: Convertir columnas en filas (inverso de PIVOT)
-- Primero crear una vista con datos pivoteados
CREATE OR REPLACE VIEW EstudiantesPorCarreraGenero AS
SELECT *
FROM (
    SELECT codigo_carrera, genero
    FROM Estudiante
)
PIVOT (
    COUNT(*) AS cantidad
    FOR genero IN ('M' AS Masculino, 'F' AS Femenino)
);

-- Luego hacer UNPIVOT
SELECT *
FROM EstudiantesPorCarreraGenero
UNPIVOT (
    cantidad
    FOR genero IN (Masculino, Femenino)
);
```

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

CASE Avanzado

```
-- CASE: Lógica condicional compleja
-- ==================================================

-- Clasificación de estudiantes por avance
SELECT
    cedula,
    nombres || ' ' || apellidos AS estudiante,
    codigo_carrera,
    creditos_aprobados,
    CASE
        WHEN creditos_aprobados >= 180 THEN 'Listo para graduación'
        WHEN creditos_aprobados >= 135 THEN 'Avance alto (75%+)'
        WHEN creditos_aprobados >= 90 THEN 'Avance medio (50%+)'
        WHEN creditos_aprobados >= 45 THEN 'Avance inicial (25%+)'
        ELSE 'Inicio de carrera'
    END AS nivel_avance,
    CASE
        WHEN creditos_aprobados >= 180 THEN '🎓'
        WHEN creditos_aprobados >= 90 THEN '📚'
        ELSE '📖'
    END AS icono
FROM Estudiante
WHERE estado = 'ACTIVO'
ORDER BY creditos_aprobados DESC;
```

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

```sql
-- Análisis de rendimiento académico
SELECT
    m.cedula_estudiante,
    e.nombres || ' ' || e.apellidos AS estudiante,
    COUNT(*) AS total_materias,
    ROUND(AVG(m.nota_final), 2) AS promedio,
    CASE
        WHEN AVG(m.nota_final) >= 9.5 THEN 'Sobresaliente'
        WHEN AVG(m.nota_final) >= 9.0 THEN 'Excelente'
        WHEN AVG(m.nota_final) >= 8.0 THEN 'Muy Bueno'
        WHEN AVG(m.nota_final) >= 7.0 THEN 'Bueno'
        ELSE 'Regular'
    END AS calificacion,
    SUM(CASE WHEN m.estado = 'APROBADO' THEN 1 ELSE 0 END) AS aprobadas,
    SUM(CASE WHEN m.estado = 'REPROBADO' THEN 1 ELSE 0 END) AS reprobadas,
    ROUND(
        SUM(CASE WHEN m.estado = 'APROBADO' THEN 1 ELSE 0 END) * 100.0 /
        COUNT(*),
        2
    ) AS tasa_aprobacion
FROM Matricula m
JOIN Estudiante e ON m.cedula_estudiante = e.cedula
WHERE m.nota_final IS NOT NULL
GROUP BY m.cedula_estudiante, e.nombres, e.apellidos
HAVING COUNT(*) >= 1
ORDER BY promedio DESC;
```

```sql
-- Búsqueda con múltiples condiciones
SELECT
    codigo,
    nombre,
    nivel,
    creditos,
    CASE
        WHEN nivel <= 2 THEN 'Nivel Básico'
        WHEN nivel <= 5 THEN 'Nivel Intermedio'
        WHEN nivel <= 7 THEN 'Nivel Avanzado'
        ELSE 'Nivel Especialización'
    END AS categoria_nivel,
    CASE
        WHEN creditos >= 5 THEN 'Alto'
        WHEN creditos >= 4 THEN 'Medio'
        ELSE 'Bajo'
    END AS peso_creditos
FROM Asignatura
ORDER BY nivel, creditos DESC;
```

OPTIMIZACIÓN DE CONSULTAS

Buenas Prácticas

```sql
-- ================================================
```

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

```
-- BUENAS PRÁCTICAS PARA OPTIMIZACIÓN
-- ==================================================

-- ✓ 1. Usar WHERE en lugar de HAVING cuando sea posible
-- MALO (filtra después de agrupar)
SELECT codigo_carrera, COUNT(*) AS total
FROM Estudiante
GROUP BY codigo_carrera
HAVING codigo_carrera = 'ING-SIS';

-- BUENO (filtra antes de agrupar)
SELECT codigo_carrera, COUNT(*) AS total
FROM Estudiante
WHERE codigo_carrera = 'ING-SIS'
GROUP BY codigo_carrera;

-- ✓ 2. Limitar columnas con SELECT específico
-- MALO
SELECT * FROM Estudiante;

-- BUENO
SELECT cedula, nombres, apellidos, email
FROM Estudiante;

-- ✓ 3. Usar EXISTS en lugar de IN con subconsultas grandes
-- MENOS EFICIENTE
SELECT * FROM Estudiante
```

```
WHERE cedula IN (SELECT cedula_estudiante FROM Matricula);

-- MÁS EFICIENTE
SELECT * FROM Estudiante e
WHERE EXISTS (
   SELECT 1 FROM Matricula m
   WHERE m.cedula_estudiante = e.cedula
);

-- ✅ 4. Evitar funciones en WHERE sobre columnas indexadas
-- MALO (no usa índice)
SELECT * FROM Estudiante
WHERE UPPER(email) = 'JUAN@EPN.EDU.EC';

-- BUENO (usa índice)
SELECT * FROM Estudiante
WHERE email = 'juan@epn.edu.ec';

-- ✅ 5. Usar INNER JOIN en lugar de WHERE para relacionar tablas
-- ANTIGUO ESTILO (menos eficiente)
SELECT e.nombres, c.nombre
FROM Estudiante e, Carrera c
WHERE e.codigo_carrera = c.codigo;

-- ESTILO MODERNO (más eficiente)
SELECT e.nombres, c.nombre
FROM Estudiante e
```

```
INNER JOIN Carrera c ON e.codigo_carrera = c.codigo;

-- ✓ 6. Limitar resultados con FETCH FIRST
SELECT nombres, apellidos, creditos_aprobados
FROM Estudiante
ORDER BY creditos_aprobados DESC
FETCH FIRST 10 ROWS ONLY;  -- Más eficiente que ROWNUM en algunos casos
```

Uso de EXPLAIN PLAN

```
-- EXPLAIN PLAN: Analizar plan de ejecución
-- ================================================

-- Generar plan de ejecución
EXPLAIN PLAN FOR
SELECT
    e.nombres,
    e.apellidos,
    c.nombre AS carrera,
    COUNT(m.id_matricula) AS total_matriculas
FROM Estudiante e
INNER JOIN Carrera c ON e.codigo_carrera = c.codigo
LEFT JOIN Matricula m ON e.cedula = m.cedula_estudiante
WHERE e.estado = 'ACTIVO'
GROUP BY e.nombres, e.apellidos, c.nombre
ORDER BY total_matriculas DESC;
```

```
-- Ver el plan
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

-- Borrar plan anterior
DELETE FROM PLAN_TABLE;

-- Estadísticas de consulta
SELECT
    sql_id,
    child_number,
    plan_hash_value,
    executions,
    elapsed_time/1000000 AS elapsed_seconds,
    cpu_time/1000000 AS cpu_seconds,
    buffer_gets,
    disk_reads
FROM v$sql
WHERE sql_text LIKE '%Estudiante%'
AND sql_text NOT LIKE '%v$sql%'
ORDER BY elapsed_time DESC;
```

Índices y Rendimiento

```
-- CREACIÓN DE ÍNDICES PARA OPTIMIZACIÓN
-- ================================================
```

```sql
-- Ver índices existentes
SELECT
    index_name,
    table_name,
    column_name,
    column_position
FROM user_ind_columns
WHERE table_name IN ('ESTUDIANTE', 'MATRICULA', 'ASIGNATURA')
ORDER BY table_name, index_name, column_position;

-- Crear índices en columnas frecuentemente consultadas
CREATE INDEX idx_estudiante_carrera ON Estudiante(codigo_carrera);
CREATE INDEX idx_estudiante_estado ON Estudiante(estado);
CREATE INDEX idx_matricula_estudiante ON Matricula(cedula_estudiante);
CREATE INDEX idx_matricula_asignatura ON Matricula(codigo_asignatura);
CREATE INDEX idx_matricula_periodo ON Matricula(periodo);

-- Índice compuesto
CREATE INDEX idx_matricula_periodo_estado ON Matricula(periodo, estado);

-- Índice en expresión
CREATE INDEX idx_estudiante_apellidos_upper ON Estudiante(UPPER(apellidos));

-- Eliminar índice
DROP INDEX idx_estudiante_apellidos_upper;

-- Reconstruir índice
```

```
ALTER INDEX idx_estudiante_carrera REBUILD;

-- Ver estadísticas de uso de índices
SELECT
    i.index_name,
    i.table_name,
    s.num_rows AS table_rows,
    i.distinct_keys,
    i.clustering_factor
FROM user_indexes i
LEFT JOIN user_tables s ON i.table_name = s.table_name
WHERE i.table_name IN ('ESTUDIANTE', 'MATRICULA')
ORDER BY i.table_name, i.index_name;
```

Hints de Oracle

```
-- HINTS: Sugerencias al optimizador
-- ========================================================

-- Forzar uso de índice
SELECT /*+ INDEX(e idx_estudiante_carrera) */
    e.nombres, e.apellidos, e.codigo_carrera
FROM Estudiante e
WHERE codigo_carrera = 'ING-SIS';
```

```
-- Forzar FULL TABLE SCAN
SELECT /*+ FULL(e) */
    * FROM Estudiante e;

-- Sugerir orden de JOIN
SELECT /*+ LEADING(c e m) */
    c.nombre,
    e.nombres,
    COUNT(m.id_matricula)
FROM Carrera c
JOIN Estudiante e ON c.codigo = e.codigo_carrera
LEFT JOIN Matricula m ON e.cedula = m.cedula_estudiante
GROUP BY c.nombre, e.nombres;

-- Paralelizar consulta
SELECT /*+ PARALLEL(e, 4) */
    codigo_carrera,
    COUNT(*) AS total
FROM Estudiante e
GROUP BY codigo_carrera;
-- Usar hash join
SELECT /*+ USE_HASH(e c) */
    e.nombres,
    c.nombre
FROM Estudiante e
JOIN Carrera c ON e.codigo_carrera = c.codigo;
```

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

EJERCICIOS PRÁCTICOS

Ejercicio 1: Consultas Básicas

Instrucciones: Resolver las siguientes consultas

1. Listar todos los estudiantes ordenados por apellido
2. Mostrar asignaturas con más de 4 créditos
3. Encontrar estudiantes con créditos entre 20 y 50
4. Buscar docentes cuyo apellido empiece con 'G' o 'M'
5. Listar estudiantes sin teléfono registrado
6. Mostrar las 10 asignaturas con más horas totales (teoría + práctica)
7. Calcular la edad de todos los estudiantes activos
8. Encontrar emails que no terminen en @epn.edu.ec
9. Listar carreras con duración mayor o igual a 10 semestres
10. Mostrar estudiantes nacidos en 2003

Entregable: Script SQL con las 10 consultas

Ejercicio 2: Funciones de Agregación y GROUP BY

Tareas:

1. Contar estudiantes por carrera y estado
2. Calcular promedio, máximo y mínimo de créditos por carrera
3. Encontrar carreras con más de 5 estudiantes activos
4. Calcular promedio de notas por asignatura
5. Mostrar docentes con más de 3 matrículas asignadas
6. Contar asignaturas por nivel y carrera
7. Calcular tasa de aprobación por periodo
8. Encontrar estudiantes con promedio >= 8.5

9. Mostrar distribución de estudiantes por género y carrera
10. Calcular estadísticas completas del sistema

Entregable: Consultas con interpretación de resultados

Ejercicio 3: JOINs y Relaciones

Desarrollar:

1. Reporte de estudiantes con sus carreras y número de matrículas
2. Asignaturas con sus prerrequisitos en formato legible
3. Matrículas completas (estudiante, asignatura, docente, notas)
4. Estudiantes sin ninguna matrícula (usar LEFT JOIN)
5. Docentes sin asignaciones actuales
6. Carreras sin estudiantes matriculados
7. Top 5 asignaturas más populares (más matrículas)
8. Estudiantes con todas sus calificaciones
9. Asignaturas que nadie ha aprobado aún
10. Reporte completo de un estudiante específico

Entregable: 10 consultas con JOINs variados

Ejercicio 4: Subconsultas

Resolver:

1. Estudiantes con más créditos que el promedio
2. Asignaturas más difíciles (menor promedio de notas)
3. Docente con más estudiantes asignados
4. Estudiantes que aprobaron todas las materias que cursaron
5. Carrera con mejor rendimiento académico promedio
6. Asignaturas sin prerrequisitos
7. Estudiantes que NO han cursado BD-501
8. Asignaturas que son prerrequisito de más de 2 materias

9. Docentes que solo imparten asignaturas de nivel avanzado (>=5)
10. Estudiantes con todas las notas sobre 7.0

Entregable: Subconsultas en WHERE, SELECT y FROM

Ejercicio 5: Consultas Avanzadas

Implementar:

1. **Ranking de estudiantes** por carrera usando ROW_NUMBER
2. **CTE complejo** con estadísticas por carrera y comparación con promedio general
3. **Consulta PIVOT** mostrando matrículas por periodo y estado
4. **Window function** para mostrar tendencia de notas por estudiante
5. **Consulta con UNION** combinando información de diferentes periodos
6. **Subconsulta correlacionada** para comparar estudiante vs promedio de su carrera
7. **CASE complejo** clasificando estudiantes en categorías de rendimiento
8. **Consulta recursiva** para cadena de prerrequisitos
9. **Análisis temporal** con LAG/LEAD mostrando evolución de matrículas
10. **Reporte ejecutivo** combinando múltiples técnicas

Entregable: Consultas avanzadas documentadas

CASOS DE PRUEBA

Caso de Prueba 1: Validación de Consultas Básicas

```
-- CASO DE PRUEBA 1: Consultas básicas
-- ==================================================

SET SERVEROUTPUT ON;

DECLARE
```

```
    v_count NUMBER;
    v_avg NUMBER;
BEGIN
    DBMS_OUTPUT.PUT_LINE('=== PRUEBAS DE CONSULTAS BÁSICAS ===');

    -- TEST 1: COUNT funciona correctamente
    SELECT COUNT(*) INTO v_count FROM Estudiante;
    IF v_count > 0 THEN
        DBMS_OUTPUT.PUT_LINE('✓ TEST 1 PASSED: ' || v_count || ' estudiantes encontrados');
    ELSE
        DBMS_OUTPUT.PUT_LINE('✗ TEST 1 FAILED: No hay estudiantes');
    END IF;

    -- TEST 2: AVG calcula correctamente
    SELECT AVG(creditos_aprobados) INTO v_avg FROM Estudiante WHERE creditos_aprobados > 0;
    IF v_avg > 0 THEN
        DBMS_OUTPUT.PUT_LINE('✓ TEST 2 PASSED: Promedio de créditos = ' || ROUND(v_avg, 2));
    ELSE
        DBMS_OUTPUT.PUT_LINE('✗ TEST 2 FAILED: Promedio incorrecto');
    END IF;

    -- TEST 3: JOINs funcionan
    SELECT COUNT(*) INTO v_count
    FROM Estudiante e
    INNER JOIN Carrera c ON e.codigo_carrera = c.codigo;

    IF v_count > 0 THEN
```

```
    DBMS_OUTPUT.PUT_LINE('✓ TEST 3 PASSED: JOIN exitoso, ' || v_count || ' registros');
  ELSE
    DBMS_OUTPUT.PUT_LINE('✗ TEST 3 FAILED: JOIN falló');
  END IF;

  -- TEST 4: WHERE filtra correctamente
  SELECT COUNT(*) INTO v_count
  FROM Estudiante
  WHERE estado = 'ACTIVO';

  DBMS_OUTPUT.PUT_LINE('✓ TEST 4 PASSED: ' || v_count || ' estudiantes activos');

  -- TEST 5: GROUP BY agrupa correctamente
  SELECT COUNT(DISTINCT codigo_carrera) INTO v_count
  FROM Estudiante;

  DBMS_OUTPUT.PUT_LINE('✓ TEST 5 PASSED: Estudiantes en ' || v_count || ' carreras diferentes');

  DBMS_OUTPUT.PUT_LINE('=== FIN PRUEBAS BÁSICAS ===');
END;
/
```

Caso de Prueba 2: Verificación de JOINs

```
-- CASO DE PRUEBA 2: Tipos de JOINs
-- ================================================
```


```sql
-- Comparar resultados de diferentes tipos de JOIN
SELECT 'INNER JOIN' AS tipo_join, COUNT(*) AS total
FROM Estudiante e
INNER JOIN Matricula m ON e.cedula = m.cedula_estudiante

UNION ALL

SELECT 'LEFT JOIN', COUNT(*)
FROM Estudiante e
LEFT JOIN Matricula m ON e.cedula = m.cedula_estudiante

UNION ALL

SELECT 'Estudiantes sin matrícula (LEFT JOIN + NULL)', COUNT(*)
FROM Estudiante e
LEFT JOIN Matricula m ON e.cedula = m.cedula_estudiante
WHERE m.id_matricula IS NULL

UNION ALL

SELECT 'RIGHT JOIN', COUNT(*)
FROM Matricula m
RIGHT JOIN Estudiante e ON m.cedula_estudiante = e.cedula;

-- Verificar consistencia
SELECT
   (SELECT COUNT(*) FROM Estudiante) AS total_estudiantes,
```

```
    (SELECT COUNT(DISTINCT cedula_estudiante) FROM Matricula) AS estudiantes_con_matricula,
    (SELECT COUNT(*) FROM Estudiante) -
    (SELECT COUNT(DISTINCT cedula_estudiante) FROM Matricula) AS estudiantes_sin_matricula
FROM DUAL;
```

Caso de Prueba 3: Funciones de Agregación

```
-- CASO DE PRUEBA 3: Validar agregaciones
-- ======================================================

-- Comparar COUNT(*) vs COUNT(columna)
SELECT
    'COUNT(*)' AS tipo,
    COUNT(*) AS resultado
FROM Estudiante

UNION ALL

SELECT
    'COUNT(telefono)',
    COUNT(telefono)
FROM Estudiante

UNION ALL

SELECT
```

```
    'COUNT(DISTINCT codigo_carrera)',
    COUNT(DISTINCT codigo_carrera)
FROM Estudiante;

-- Verificar cálculos manuales vs automáticos
SELECT
    AVG(nota_final) AS promedio_automatico,
    SUM(nota_final) / COUNT(nota_final) AS promedio_manual,
    CASE
        WHEN ABS(AVG(nota_final) - (SUM(nota_final) / COUNT(nota_final))) < 0.001
        THEN 'CORRECTO'
        ELSE 'ERROR'
    END AS verificacion
FROM Matricula
WHERE nota_final IS NOT NULL;
```

Análisis de resultados:

Conclusiones y recomendaciones:

Bibliografía:

(La bibliografía en formato IEEE indicada al principio del curso debe ser utilizada como principal)

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

PREGUNTAS DE EVALUACIÓN

Sección A: Preguntas Teóricas (40 puntos)

1. Explique la diferencia entre INNER JOIN, LEFT JOIN, RIGHT JOIN y FULL OUTER JOIN. Proporcione un ejemplo de cuándo usar cada uno.
2. ¿Cuál es la diferencia entre WHERE y HAVING? ¿Por qué no se puede usar WHERE después de GROUP BY?
3. Explique qué son las funciones de ventana (Window Functions) y en qué se diferencian de las funciones de agregación regulares.
4. ¿Qué es una subconsulta correlacionada? ¿Cuándo es preferible usar EXISTS en lugar de IN?
5. Describa tres técnicas para optimizar consultas SQL lentas.
6. Explique la diferencia entre UNION y UNION ALL. ¿Cuándo usaría cada uno?
7. ¿Qué son los CTEs (Common Table Expressions)? ¿Qué ventajas ofrecen sobre las subconsultas en FROM?
8. Explique cómo afectan los índices al rendimiento de las consultas SELECT. ¿Cuándo podría un índice empeorar el rendimiento?

Sección B: Preguntas Prácticas

Pregunta 9 : Escriba una consulta que muestre: • Nombre

del estudiante
- Carrera
- Total de matrículas
- Promedio de notas
- Número de asignaturas aprobadas y reprobadas
- Porcentaje de aprobación
- Ranking dentro de su carrera por promedio

Solo estudiantes con al menos 3 matrículas. Ordenar por carrera y promedio descendente.

Pregunta 10 : Crear una consulta que identifique:

- Asignaturas que tienen baja tasa de aprobación (<70%)
- Que han sido cursadas por al menos 5 estudiantes

- Mostrar: código, nombre, nivel, total matriculados, aprobados, reprobados, tasa de aprobación
- Incluir el promedio de notas de los que aprobaron
- Ordenar por tasa de aprobación ascendente

Pregunta 11 : Desarrolle un reporte ejecutivo que muestre por cada carrera: • Nombre

de la carrera
- Total de estudiantes activos
- Estudiantes por género (M/F/O) •
Promedio de créditos aprobados •
Total de asignaturas ofertadas
- Asignaturas más populares (top 3)
- Docentes asignados
- Tasa de aprobación del último periodo

Use CTEs, JOINs y subconsultas según sea necesario.

Pregunta 12 (: Escriba una consulta que encuentre:

- Estudiantes que han cursado asignaturas sin cumplir los prerrequisitos
- Mostrar: estudiante, asignatura cursada, prerrequisito faltante
- Incluir información de periodo y estado de la matrícula
- Ordenar por estudiante y periodo

Use JOINs y subconsultas correlacionadas.