BASE DE DATOS I: SABERES PREVIOS CON RESPECTO AL SCRIPT QUE SE VIENE TRABAJANDO EN LA 3 UNIDAD Y CONCEPTOS GENERALES EN SQL

I. Glosario de términos del script de empleados (Traducción por Google Traductor, se sugiere entender más el contexto que la TABLA/COLUMNA en sí)

- CITY - Ciudad
- COMMISSION PCT - Porcentaje de comisión
- COUNTRY - País
- DEPARTMENT - Departamento
- EMAIL - Correo electrónico
- EMPLOYEE - Empleado
- END DATE - Fecha de finalización
- FIRST NAME - Nombre
- GRADE LEVEL - Nivel de grado
- HIGHEST SAL - Salario más alto
- HIRE DATE - Fecha de contratación
- JOB - Trabajo o Puesto
- JOB GRADE - Nivel de trabajo
- JOB HISTORY - Historial de trabajo
- JOB TITLE - Título del trabajo
- LAST NAME - Apellido
- LOCATION - Ubicación
- LOWEST SAL - Salario más bajo
- MANAGER - Gerente o Jefe
- MAX SALARY - Salario máximo
- MIN SALARY - Salario mínimo
- PHONE NUMBER - Número de teléfono
- POSTAL CODE - Código postal
- REGION - Región
- SALARY - Salario
- START DATE - Fecha de inicio
- STATE PROVINCE - Estado o Provincia
- STREET ADDRESS - Dirección

II. Términos generales de SQL

- AND - Y
- AVG - Promedio
- BETWEEN - Entre
- COUNT - Contar
- DISTINCT - Distinto
- FROM - Desde
- GROUP BY - Agrupar por
- HAVING - Teniendo
- IN - En
- INNER JOIN - Unión interna
- IS NOT NULL - No es nulo
- IS NULL - Es nulo
- JOIN - Unir
- LEFT JOIN - Unión izquierda
- LIKE - Como
- MAX - Máximo
- MIN - Mínimo
- NOT - No
- OR - O
- ORDER BY - Ordenar por
- RIGHT JOIN - Unión derecha
- SELECT - Seleccionar
- SUM - Sumar
- UNION - Unión
- WHERE – Donde

III. Consultas básicas y sentencias utilizadas que se debe tener en cuenta para la base de datos de empleados III Unidad.

1. Consultas básicas:

- SELECT: Para recuperar datos
- FROM: Para especificar las tablas
- WHERE: Para filtrar registros
- ORDER BY: Para ordenar resultados

Ejemplo:

```sql
SELECT column1, column2 FROM table_name WHERE condition ORDER BY column1;
```

2. Joins:

- INNER JOIN: Combina registros de dos tablas cuando hay coincidencia
- LEFT JOIN: Incluye todos los registros de la tabla izquierda
- RIGHT JOIN: Incluye todos los registros de la tabla derecha

Ejemplo:

```sql
SELECT e.name, d.department_name
FROM employees e
INNER JOIN departments d ON e.department_id = d.id;
```

3. Funciones de agregación:

- COUNT(): Cuenta el número de filas
- SUM(): Suma valores
- AVG(): Calcula el promedio
- MAX(): Encuentra el valor máximo
- MIN(): Encuentra el valor mínimo

Ejemplo:

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

4. **Subconsultas:**

- Consultas anidadas dentro de otra consulta

Ejemplo:

```sql
SELECT name FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

5. **Funciones de fecha:**

- DATEADD(): Añade un intervalo a una fecha
- DATEDIFF(): Calcula la diferencia entre fechas
- GETDATE(): Obtiene la fecha y hora actuales

Ejemplo:

```sql
SELECT name, DATEDIFF(year, hire_date, GETDATE()) as years_employed
FROM employees;
```

6. **Manejo de NULL:**

- IS NULL: Comprueba si un valor es nulo
- IS NOT NULL: Comprueba si un valor no es nulo
- COALESCE(): Devuelve el primer valor no nulo

Ejemplo:

```sql
SELECT name, COALESCE(commission, 0) as commission
FROM employees;
```

7. **Funciones de cadena:**

- CONCAT(): Concatena cadenas
- SUBSTRING(): Extrae parte de una cadena
- UPPER() / LOWER(): Convierte a mayúsculas/minúsculas

Ejemplo:

```sql
SELECT CONCAT(first_name, ' ', last_name) as full_name
FROM employees;
```

8. Cláusulas GROUP BY y HAVING:

- GROUP BY: Agrupa filas que tienen los mismos valores
- HAVING: Especifica condiciones para grupos

Ejemplo:

```sql
SELECT department_id, COUNT(*) as employee_count
FROM employees
GROUP BY department_id
HAVING COUNT(*) > 5;
```

9. UNION y UNION ALL:

- Combina los resultados de dos o más consultas SELECT

Ejemplo:

```sql
SELECT name FROM employees
UNION
SELECT name FROM contractors;
```

10. Expresiones CASE:

- Para lógica condicional dentro de una consulta

Ejemplo:

```sql
SELECT name,
       CASE
         WHEN salary < 50000 THEN 'Low'
         WHEN salary BETWEEN 50000 AND 100000 THEN 'Medium'
         ELSE 'High'
       END as salary_category
FROM employees;
```

11. Manejo básico de fechas

```sql
-- Tutorial: Extracción de componentes de fecha
SELECT
    EMPLOYEE_ID,
    FIRST_NAME,
    LAST_NAME,
    HIRE_DATE,
    YEAR(HIRE_DATE) AS hire_year,
    MONTH(HIRE_DATE) AS hire_month,
    DAY(HIRE_DATE) AS hire_day
FROM
    EMPLOYEES
WHERE
    YEAR(HIRE_DATE) = 2000;

-- Esta consulta muestra:
-- 1. Cómo extraer año, mes y día de una fecha
-- 2. Cómo filtrar por un año específico
```

De la misma manera como se manejan fechas, se pueden realizar cálculos con las fechas

```sql
-- Tutorial: Cálculo de antigüedad
SELECT
    EMPLOYEE_ID,
    FIRST_NAME,
    LAST_NAME,
    HIRE_DATE,
    DATEDIFF(year, HIRE_DATE, CURRENT DATE) AS years_employed
FROM
    EMPLOYEES
ORDER BY
    years_employed DESC;

-- Esta consulta muestra:
-- 1. Cómo calcular la diferencia entre dos fechas
-- 2. Uso de CURRENT DATE para obtener la fecha actual
```

12. Manejo de CTE (Expresión de Tabla Común)

Explicación básica del CTE

```sql
-- CTE: Expresión de Tabla Común
-- Es una tabla temporal nombrada que existe solo para una consulta

-- Ejemplo: Listar empleados con salarios altos
WITH altos_salarios AS (
    -- Definición del CTE
    SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME, SALARY
    FROM EMPLOYEES
    WHERE SALARY > 10000
)
-- Uso del CTE
SELECT *
FROM altos_salarios
ORDER BY SALARY DESC;

-- Este ejemplo:
-- 1. Define un CTE llamado 'altos_salarios'
-- 2. El CTE selecciona empleados con salario > 10000
-- 3. Luego usa este CTE en la consulta principal
-- 4. Simplifica la consulta principal y mejora la legibilidad
```

Explicación y cálculos básicos con CTE

sql

```sql
-- Tutorial: CTE para calcular salario promedio
WITH avg_salary AS (
    SELECT AVG(SALARY) AS company_avg
    FROM EMPLOYEES
)
SELECT
    e.EMPLOYEE_ID,
    e.FIRST_NAME,
    e.LAST_NAME,
    e.SALARY,
    a.company_avg,
    e.SALARY - a.company_avg AS difference
FROM
    EMPLOYEES e, avg_salary a
WHERE
    e.SALARY > a.company_avg
ORDER BY
    difference DESC;

-- Esta consulta muestra:
-- 1. Cómo definir un CTE simple
-- 2. Cómo usar el CTE en la consulta principal
```

13. Manejo básico de subconsultas

Subconsulta en SELECT

sql

```sql
-- Tutorial: Subconsulta para obtener el nombre del manager
SELECT
    e.EMPLOYEE_ID,
    e.FIRST_NAME,
    e.LAST_NAME,
    (SELECT m.FIRST_NAME || ' ' || m.LAST_NAME
     FROM EMPLOYEES m
     WHERE m.EMPLOYEE_ID = e.MANAGER_ID) AS manager_name
FROM
    EMPLOYEES e
WHERE
    e.MANAGER_ID IS NOT NULL;

-- Esta consulta muestra:
-- 1. Cómo usar una subconsulta en la cláusula SELECT
-- 2. Cómo relacionar la subconsulta con la consulta principal
```

Subconsulta en WHERE

sql

```sql
-- Tutorial: Subconsulta para encontrar empleados con salario superior al promedio
SELECT
    EMPLOYEE_ID,
    FIRST_NAME,
    LAST_NAME,
    SALARY
FROM
    EMPLOYEES
WHERE
    SALARY > (SELECT AVG(SALARY) FROM EMPLOYEES)
ORDER BY
    SALARY DESC;

-- Esta consulta muestra:
-- 1. Cómo usar una subconsulta en la cláusula WHERE
-- 2. Cómo comparar valores con el resultado de una subconsulta
```

14. Consultas básicas y filtrado avanzado:

Estas consultas demuestran el uso de subconsultas en la cláusula WHERE. La primera compara cada salario con el promedio, mientras que la segunda utiliza IN para encontrar empleados en departamentos específicos.

```sql
-- Selecciona empleados con salario superior al promedio
SELECT employee_id, last_name, salary
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees)
ORDER BY salary DESC;

-- Uso de IN con una subconsulta
SELECT last_name, department_id
FROM employees
WHERE department_id IN (
    SELECT department_id
    FROM departments
    WHERE location_id = 1700
);
```

15. Joins y subconsultas correlacionadas:

Esta consulta combina un JOIN con una subconsulta correlacionada. Muestra empleados en departamentos con más de 5 empleados.

```sql
-- Join con una subconsulta correlacionada
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
JOIN (
    SELECT department_id, department_name
    FROM departments
    WHERE EXISTS (
        SELECT 1
        FROM employees
        WHERE department_id = departments.department_id
        HAVING COUNT(*) > 5
    )
) d ON e.department_id = d.department_id;
```

16. Funciones de agregación y subconsultas:

Esta consulta utiliza una subconsulta correlacionada para encontrar el salario máximo en cada departamento.

```sql
-- Encuentra el empleado con el salario más alto en cada departamento
SELECT e.department_id, e.last_name, e.salary
FROM employees e
WHERE e.salary = (
    SELECT MAX(salary)
    FROM employees
    WHERE department_id = e.department_id
);
```

17. Subconsultas en la cláusula FROM:

Aquí, la subconsulta en la cláusula FROM calcula el total de salarios una sola vez, mejorando la eficiencia.

```sql
-- Calcula el porcentaje del salario de cada empleado respecto al total
SELECT e.last_name,
       e.salary,
       (e.salary / s.total_salary * 100) AS percentage
FROM employees e,
     (SELECT SUM(salary) AS total_salary FROM employees) s;
```

18. Uso avanzado de CASE y subconsultas:

Esta consulta utiliza CASE junto con subconsultas para clasificar los salarios de los empleados en relación con el promedio de su departamento.

```sql
-- Clasifica empleados basado en su salario relativo al promedio del departamento
SELECT e.last_name,
       e.salary,
       e.department_id,
       CASE
           WHEN e.salary > (
               SELECT AVG(salary)
               FROM employees
               WHERE department_id = e.department_id
           ) THEN 'Above Average'
           WHEN e.salary < (
               SELECT AVG(salary)
               FROM employees
               WHERE department_id = e.department_id
           ) THEN 'Below Average'
           ELSE 'Average'
       END AS salary_status
FROM employees e;
```

19. Subconsultas en la cláusula HAVING:

Esta consulta utiliza una subconsulta en la cláusula HAVING para comparar promedios.

```
-- Encuentra departamentos donde el salario promedio es mayor que el promedio general
SELECT department_id, AVG(salary) AS avg_salary
FROM employees
GROUP BY department_id
HAVING AVG(salary) > (SELECT AVG(salary) FROM employees);
```