SCRIP DE DATACAMP

**Hierarchical and Recursive**

Capitulo 1: Recursion and Common Table Expression (CTE)

* Cree el CTE ITjobs.
* Definir los campos del CTE como ID, Name, y Position.
* Encuentre las posiciones que comienzan con ITy el nombre que comienza con A

WITH ITjobs (ID, Name, Position) AS (

SELECT ID,

Name,

Position

FROM employee

-- Find IT jobs and names starting with A

WHERE Position LIKE 'IT%' AND Name LIKE 'A%')

SELECT \* FROM ITjobs;

* Defina el segundo CTE`` ITSalarycon los campos IDy Salary.
* Encuentre los sueldos arriba 3000.
* Combine los dos CTE utilizando una JOINde las coincidencias IDy seleccione el nombre, el salario y el puesto de los empleados seleccionados

WITH ITjobs (ID, Name, Position) AS (

SELECT ID,

Name,

Position

FROM employee

WHERE Position like 'IT%'),

-- Define the second CTE table ITSalary with the fields ID and Salary

ITSalary (ID, Salary) AS (

SELECT

ID,

Salary

FROM Salary

-- Find salaries above 3000

WHERE Salary > 3000)

SELECT

ITjobs.NAME,

ITjobs.POSITION,

ITsalary.Salary

FROM ITjobs

-- Combine the two CTE tables the correct join variant

INNER JOIN ITsalary

-- Execute the join on the ID of the tables

ON ITjobs.ID = ITsalary.ID;

* Establezca el @targetfactorial, que también servirá como condición de terminación, en 5.
* Inicializa el @factorialresultado.
* Calcule el @factorialnúmero tomando el producto del resultado factorial hasta el momento y la iteración actual.
* Reduzca la condición de terminación en 1 al final de la iteración

-- Define the target factorial number

DECLARE @target float = 5

-- Initialization of the factorial result

DECLARE @factorial float = 1

WHILE @target > 0

BEGIN

-- Calculate the factorial number

SET @factorial = @target \* @factorial

-- Reduce the termination condition

SET @target = @target - 1

END

SELECT @factorial;

* Inicialice los campos factorialy stepen 1.
* Calcula la parte recursiva con factorial \* (step + 1).
* Detenga el proceso de recursividad cuando el valor de iteración actual sea menor que el número factorial objetivo.

WITH calculate\_factorial AS (

SELECT

-- Initialize step and the factorial number

1 AS step,

1 AS factorial

UNION ALL

SELECT

step + 1,

-- Calculate the recursive part by n!\*(n+1)

factorial \* (step + 1)

FROM calculate\_factorial

-- Stop the recursion reaching the wanted factorial number

WHERE step < 6)

SELECT factorial

FROM calculate\_factorial;

* Limite el paso de recursividad a 50 en la consulta recursiva.
* Defina el CTE con el nombre counting\_numbers.
* Inicializar numberen la consulta de anclaje.
* Agregue 1a numbercada paso de recursividad.

-- Define the CTE

WITH counting\_numbers AS (

SELECT

-- Initialize number

1 AS number

UNION ALL

SELECT

-- Increment number by 1

number + 1

FROM counting\_numbers

-- Set the termination condition

WHERE number < 50)

SELECT number

FROM counting\_numbers;

* Defina el CTE calculate\_potenciescon los campos stepy result.
* Inicializar stepy result.
* Agregue el siguiente paso al POWER() step + 1a result.

-- Define the CTE calculate\_potencies with the fields step and result

WITH calculate\_potencies (step, result) AS (

SELECT

-- Initialize step and result

1,

1

UNION ALL

SELECT

step + 1,

-- Add the POWER calculation to the result

result + POWER(step + 1, step + 1)

FROM calculate\_potencies

WHERE step < 9)

SELECT

step,

result

FROM calculate\_potencies;

Capitulo 2: **Hierarchical and Recursive Queries**

* Inicialice number\_of\_letteral número que representa la letra A.
* Aumente el valor de number\_of\_letteren 1 en cada paso y establezca el límite en 90el valor de Z.
* Seleccione el miembro recursivo del CTE definido.

WITH alphabet AS (

SELECT

-- Initialize letter to A

65 AS number\_of\_letter

-- Statement to combine the anchor and the recursive query

UNION ALL

SELECT

-- Add 1 each iteration

number\_of\_letter + 1

-- Select from the defined CTE alphabet

FROM alphabet

-- Limit the alphabet to A-Z

WHERE number\_of\_letter < 90)

SELECT char(number\_of\_letter)

FROM alphabet;

* Inicialice la hora actual como time.
* Seleccione el CTE de forma recursiva y combine el ancla y el miembro recursivo con la declaración correcta.
* Limite el número de iteraciones a días en un año menos 1
* Aumente el número máximo de iteraciones al número de días en un año con OPTION (MAXRECURSION n)

WITH time\_series AS (

SELECT

-- Get the current time

getdate() AS time

UNION ALL

SELECT

DATEADD(day, 1, time)

-- Call the CTE recursively

FROM time\_series

-- Limit the time series to 1 year minus 1 (365 days -1)

WHERE time < GETDATE() + 364)

SELECT time

FROM time\_series

-- Increase the number of iterations (365 days)

OPTION(MAXRECURSION 365)

* Create a CTE with the name employee\_hierarchy.
* Select the information of the IT director as the initial step of the CTE by filtering on his Supervisor ID.
* Perform a join with employee to get the name of the manager.

-- Create the CTE employee\_hierarchy

WITH employee\_hierarchy AS (

SELECT

ID,

NAME,

Supervisor

FROM employee

-- Start with the IT Director

WHERE Supervisor = 1

UNION ALL

SELECT

emp.ID,

emp.NAME,

emp.Supervisor

FROM employee emp

JOIN employee\_hierarchy

ON emp.Supervisor = employee\_hierarchy.ID)

SELECT

cte.Name as EmployeeName,

emp.Name as ManagerName

FROM employee\_hierarchy as cte

JOIN employee as emp

-- Perform the JOIN on Supervisor and ID

ON cte.Supervisor = emp.ID;

* Initialize the field LEVEL to 1 at the start of the recursion.
* Select the information of the IT director as the initial step of the CTE by filtering on Supervisor.
* Set LEVEL to the current recursion step.
* Perform a JOIN with the defined CTE on the IDs of the supervisor and the employee.

WITH employee\_hierarchy AS (

SELECT

ID,

NAME,

Supervisor,

-- Initialize the field LEVEL

1 as LEVEL

FROM employee

-- Start with the supervisor ID of the IT Director

WHERE Supervisor = 0

UNION ALL

SELECT

emp.ID,

emp.NAME,

emp.Supervisor,

-- Increment LEVEL by 1 each step

LEVEL + 1

FROM employee emp

JOIN employee\_hierarchy

-- JOIN on supervisor and ID

ON emp.supervisor = employee\_hierarchy.ID)

SELECT

cte.Name, cte.Level,

emp.Name as ManagerID

FROM employee\_hierarchy as cte

JOIN employee as emp

ON cte.Supervisor = emp.ID

ORDER BY Level;

* Initialize Path to the ID of the supervisor (0) and the start condition of the recursion.
* UNION the anchor member and define the recursive member fields (ID, Name, Supervisor).
* Add the the ID of the supervisor Supervisor to the Path in each step.
* Select the IDs of employees Christian Feierabend and Jasmin Mentil in the CTE.

WITH employee\_Hierarchy AS (

SELECT

ID,

NAME,

Supervisor,

-- Initialize the Path with CAST

CAST('0' AS VARCHAR(MAX)) as Path

FROM employee

WHERE Supervisor = 0

-- UNION the anchor query

UNION ALL

-- Select the recursive query fields

SELECT

emp.ID,

emp.Name,

emp.Supervisor,

-- Add the supervisor in each step. CAST the supervisor.

Path + '->' + CAST(emp.supervisor AS VARCHAR(MAX))

FROM employee emp

INNER JOIN employee\_Hierarchy

ON emp.Supervisor = employee\_Hierarchy.ID

)

SELECT Path

FROM employee\_Hierarchy

-- Select the employees Christian Feierabend and Jasmin Mentil

WHERE ID = 16 OR ID = 18;

* Inicialice el inicio de recursividad configurando ParentIDto 101.
* Establecer LEVELen el paso de recursividad actual.
* Únase al miembro ancla al CTE en la identificación del padre y del niño.
* COUNT() el número de generaciones.

WITH children AS (

SELECT

ID,

Name,

ParentID,

0 as LEVEL

FROM family

-- Set the targeted parent as recursion start

WHERE ParentID = 101

UNION ALL

SELECT

child.ID,

child.NAME,

child.ParentID,

-- Increment LEVEL by 1 each step

LEVEL + 1

FROM family child

INNER JOIN children

-- Join the anchor query with the CTE

ON child.ParentID = children.ID)

SELECT

-- Count the number of generations

COUNT(LEVEL) as Generations

FROM children

OPTION(MAXRECURSION 300);

* Establezca el ParentIDde 290 como punto de partida.
* Si está Parent.ID = ''en la CASEoperación, el Parentcampo debe establecerse en el actual ParentID.
* Si está Parent.ID <> ''en la CASEoperación, el ID de padre debe agregarse al padre actual para cada iteración.
* Seleccione Namey Parentdel CTE definido.

WITH tree AS (

SELECT

ID,

Name,

ParentID,

CAST('0' AS VARCHAR(MAX)) as Parent

FROM family

-- Initialize the ParentID to 290

WHERE ParentID = 290

UNION ALL

SELECT

Next.ID,

Next.Name,

Parent.ID,

CAST(CASE WHEN Parent.ID = ''

-- Set the Parent field to the current ParentID

THEN(CAST(Next.ParentID AS VARCHAR(MAX)))

-- Add the ParentID to the current Parent in each iteration

ELSE(Parent.Parent + ' -> ' + CAST(Next.ParentID AS VARCHAR(MAX)))

END AS VARCHAR(MAX))

FROM family AS Next

INNER JOIN tree AS Parent

ON Next.ParentID = Parent.ID)

-- Select the Name, Parent from tree

SELECT Name, Parent

FROM tree;

Capitulo 3: ***Creating Data Models on Your Own***

* Define the table Person.
* Define a field IndividualID.
* Set Firstname and Lastname not to be NULL and of type VARCHAR(255).
* Define Birthday as DATE.

-- Define the table Person

CREATE TABLE Person (

-- Define the Individual ID

IndividualID INT NOT NULL,

-- Set Firstname and Lastname not to be NULL of type VARCHAR(255)

Firstname VARCHAR(255) NOT NULL,

Lastname VARCHAR(255) NOT NULL,

Address VARCHAR(255) NOT NULL,

City CHAR(32) NOT NULL,

-- Define Birthday as DATE

Birthday DATE

);

SELECT \*

FROM Person;

* Insert the new values for ID = 1 into the Person table.
* Insert the values Peter, Jackson, 342 Flushing st, New York, and 1986-12-30 in the corresponding fields of the table.

-- Insert the records for the person with ID=1

Insert INTO person

VALUES ('1','Andrew','Anderson','Union Ave 10','New York','1986-12-30');

-- Insert the records for the person with ID=2

INSERT INTO Person

VALUES ('2','Peter','Jackson','342 Flushing st','New York','1986-12-30');

SELECT \* FROM Person;

* Delete the person with an ID equal to 1 in the table.
* Delete the person whose last name is Jackson.

INSERT INTO Person

VALUES ('1','Andrew','Anderson','Adress 1','City 1','1986-12-30');

INSERT INTO Person

VALUES ('2','Peter','Jackson','Adress 2','City 2','1986-12-30');

INSERT INTO Person

VALUES ('5','Michaela','James','Adress 3','City 3','1976-03-07');

-- Delete the record for the person with the ID of 1

Delete FROM person

WHERE ID = 1;

-- Delete the record with the name Jackson

Delete FROM person

WHERE lastname = 'Jackson';

SELECT \* FROM Person;

* Add the column Email to Person.
* Delete the column Birthday from Person.

-- Add the column EMail to Person

ALTER TABLE Person

ADD EMail VARCHAR(255);

-- Delete the column Birthday of Person

ALTER TABLE Person

DROP column Birthday ;

-- Check the table definition

SELECT \* FROM Person;

* Define the primary key PersonID for Person of type INT.
* Define the primary key OrderID for History.
* Define the foreign key PersonID referencing the primary key of Person.

CREATE TABLE Person (

-- Define the primary key for Person of type INT

PersonID INT NOT NULL primary key,

Firstname VARCHAR(255) NOT NULL,

Lastname VARCHAR(255) NOT NULL,

);

CREATE TABLE History (

-- Define the primary key for History

OrderID INT NOT NULL primary key,

Item VARCHAR(255) NOT NULL,

Price INT NOT NULL,

-- Define the foreign key for History

PersonID INT foreign key REFERENCES Person(PersonID)

);

SELECT \* FROM History;

* Insert new data for the person with ID=1, Andrew Anderson.
* Insert new data for the second person with ID=2, Sam Smith.
* Insert a new order for Andrew Anderson: iPhone XS for 1000.
* Insert a new order for Sam Smith: MacBook Pro for 1800.

-- Insert new data into the table Person

Insert INTO Person

VALUES ('1','Andrew ','Anderson','Union Ave 10','New York','1986-12-30');

Insert INTO Person

VALUES ('2','Sam ','Smith','Flushing Ave 342','New York','1986-12-30');

-- Insert new data into the table History

Insert INTO History

VALUES ('1','IPhone XS','1000','1');

INSERT INTO History

VALUES ('2','MacBook Pro','1800','2');

SELECT \* FROM History;

* COUNT() the number of orders and alias it as Orders.
* SUM() the total price of all orders and alias it as Costs.
* Join the Person and History tables.
* Aggregate the information on ID using GROUP BY.

INSERT INTO Person

VALUES ('1', 'Andrew', 'Anderson','Union Ave 10','New York','1986-12-30');

INSERT INTO Person

VALUES ('2', 'Sam', 'Smith','Flushing Ave 342','New York','1986-12-30');

INSERT INTO History VALUES ( '1', 'IPhone XS', '1000', '1');

INSERT INTO History VALUES ( '2', 'MacBook Pro', '1800', '1');

INSERT INTO History VALUES ( '5', 'IPhone XR', '600', '2');

INSERT INTO History VALUES ( '6', 'IWatch 4', '400', '1');

SELECT

Person.ID,

-- Count the number of orders

Count(Item) as Orders,

-- Add the total price of all orders

SUM(Price) as Costs

FROM Person

-- Join the tables Person and History on their IDs

Join History

ON Person.ID = History.PersonID

-- Aggregate the information on the ID

GROUP BY Person.ID;

* Define the fields ID and ParentID of type INT. ID should not be NULL, ParentID can be NULL.
* Insert the equipment Software into the table with the correct IDs. The software is part of Asset.
* Insert the equipment Monitor into the table with the correct IDs. The monitor is part of Hardware.
* Insert the equipment Microsoft Office into the table with the correct IDs. This software is part of Application.

CREATE TABLE Equipment (

-- Define ID and ParentID

ID int primary key,

Equipment VARCHAR(255) NOT NULL,

ParentID int null

);

INSERT INTO Equipment VALUES ('1','Asset',NULL);

INSERT INTO Equipment VALUES ('2','Hardware','1');

-- Insert the type Software

INSERT INTO Equipment VALUES ('3','Software','1');

INSERT INTO Equipment VALUES ('4','Application','3');

INSERT INTO Equipment VALUES ('5','Tool','3');

INSERT INTO Equipment VALUES ('6','PC','2');

-- Insert the type Monitor

INSERT INTO Equipment VALUES ('7','Monitor','2');

INSERT INTO Equipment VALUES ('8','Phone','2');

INSERT INTO Equipment VALUES ('9','IPhone','8');

-- Insert the type Microsoft Office

INSERT INTO Equipment VALUES ('10','Microsoft Office','4');

SELECT \* FROM Equipment;

* Define the fields Departure and Destination, neither of which can be NULL.
* Insert the route from San Francisco to New York for Bus 1.
* Insert the route from Florida to San Francisco for Bus 9.
* Select all possible departure locations.

CREATE TABLE Trip (

-- Define the Departure

Departure VARCHAR(255) NOT NULL,

BusName VARCHAR(255) NOT NULL,

-- Define the Destination

Destination VARCHAR(255) NOT NULl,

);

-- Insert a route from San Francisco to New York

Insert INTO Trip VALUES ('San Francisco', 'Bus 1','New York');

-- Insert a route from Florida to San Francisco

INSERT INTO Trip VALUES ( 'Florida', 'Bus 9','San Francisco');

INSERT INTO Trip VALUES ( 'San Francisco', 'Bus 2','Texas');

INSERT INTO Trip VALUES ( 'San Francisco', 'Bus 3','Florida');

INSERT INTO Trip VALUES ( 'San Francisco', 'Bus 4','Washington');

INSERT INTO Trip VALUES ( 'New York', 'Bus 5','Texas');

INSERT INTO Trip VALUES ( 'New York', 'Bus 6','Washington');

INSERT INTO Trip VALUES ( 'Florida', 'Bus 7','New York');

INSERT INTO Trip VALUES ( 'Florida', 'Bus 8','Toronto');

-- Get all possible departure locations

SELECT distinct Departure

FROM trip;

Capitulo 4: **Hierarchical Queries of Real-World Examples**

* Define the CTE table possible\_Airports with the field Airports.
* Select the airports by combining Departure and Arrival airports.
* Combine the departure with the destination airports using the correct statement.
* Select all possible Airports from possible\_Airports.

-- Definition of the CTE table

WITH possible\_Airports (Airports) AS(

-- Select the departure airports

SELECT Departure

FROM flightPlan

-- Combine the two queries

UNION

-- Select the destination airports

SELECT Arrival

FROM flightPlan)

-- Get the airports from the CTE table

SELECT Airports

FROM possible\_Airports;

* Initialize the number of stops, increment it in the recursive query, and limit it to less than 5.

WITH flight\_route (Departure, Arrival, stops) AS(

SELECT

f.Departure, f.Arrival,

-- Initialize the number of stops

0

FROM flightPlan f

WHERE Departure = 'Vienna'

UNION ALL

SELECT

p.Departure, f.Arrival,

-- Increment the number of stops

p.stops + 1

FROM flightPlan f, flight\_route p

-- Limit the number of stops

WHERE p.Arrival = f.Departure AND

p.stops < 5)

SELECT

DISTINCT Arrival,

Departure,

stops

FROM flight\_route;

* Define PartID as PRIMARY KEY of type INT.
* Define Cost of type INT and not to be NULL.
* Insert the root element SUV as described in the context section.
* Insert the entry Wheels as described in the context section.

CREATE TABLE Bill\_Of\_Material (

-- Define PartID as primary key of type INT

PartID INT NOT NULL primary KEY,

SubPartID INT,

Component VARCHAR(255) NOT NULL,

Title VARCHAR(255) NOT NULL,

Vendor VARCHAR(255) NOT NULL,

ProductKey CHAR(32) NOT NULL,

-- Define Cost of type INT and NOT NULL

Cost INT NOT NULL,

Quantity INT NOT NULL);

-- Insert the root element SUV as described

INSERT INTO Bill\_Of\_Material

VALUES ('1',NULL,'SUV','BMW X1','BMW','F48',50000,1);

INSERT INTO Bill\_Of\_Material

VALUES ('2','1','Engine','V6BiTurbro','BMW','EV3891ASF',3000,1);

INSERT INTO Bill\_Of\_Material

VALUES ('3','1','Body','AL\_Race\_Body','BMW','BD39281PUO',5000,1);

INSERT INTO Bill\_Of\_Material

VALUES ('4','1','Interior Decoration','All\_Leather\_Brown','BMW','ZEU198292',2500,1);

-- Insert the entry Wheels as described

INSERT INTO Bill\_Of\_Material

VALUES ('5','1','Wheels','M-Performance 19/255','BMW','MKQ134098URZ','400','4');

SELECT \*

FROM Bill\_Of\_Material;

* Define construction\_Plan with the fields: PartID, SubPartID, Title, Component and Level.
* Initialize the field Level to 1.
* Increase Level by 1 in every recursion step.
* Limit the number of steps to Level = 2.

-- Define CTE with the fields: PartID, SubPartID, Title, Component, Level

with construction\_Plan (PartID, SubPartID,Title, Component, Level) AS (

SELECT

PartID,

SubPartID,

Title,

Component,

-- Initialize the field Level

1

FROM partList

WHERE PartID = '1'

UNION ALL

SELECT

CHILD.PartID,

CHILD.SubPartID,

CHILD.Title,

CHILD.Component,

-- Increment the field Level each recursion step

PARENT.Level + 1

FROM construction\_Plan PARENT, partList CHILD

WHERE CHILD.SubPartID = PARENT.PartID

-- Limit the number of iterations to Level < 2

AND PARENT.Level < 2)

SELECT DISTINCT PartID, SubPartID, Title, Component, Level

FROM construction\_Plan

ORDER BY PartID, SubPartID, Level;

* Define construction\_Plan with the necessary fields.
* Initialize Total with the Quantity in the anchor element of CTE.
* Increase Total with the Quantity of the child element in the recursion element.
* Use SUM() to create the sum of Total on the aggregated information on IDs of the hierarchy.

-- Define CTE with the fields: PartID, SubPartID, Level, Component, Total

WITH construction\_Plan (PartID, SubPartID, Level, Component, Total) AS (

SELECT

PartID,SubPartID,

0,

Component,

-- Initialize Total

Quantity

FROM partList

WHERE PartID = '1'

UNION ALL

SELECT

CHILD.PartID, CHILD.SubPartID,

PARENT.Level + 1,

CHILD.Component,

-- Increase Total by the quantity of the child element

PARENT.Total + CHILD.Quantity

FROM construction\_Plan PARENT, partList CHILD

WHERE CHILD.SubPartID = PARENT.PartID

AND PARENT.Level < 3)

SELECT

PartID, SubPartID,Component,

-- Calculate the sum of total on the aggregated information

SUM(Total)

FROM construction\_Plan

GROUP BY PartID, SubPartID, Component

ORDER BY PartID, SubPartID;

* CREATE the structure table.
* Define the EquipmentID field as a PRIMARY KEY of type INT.
* Insert the record for line 1: 1, 2, <no from line>, 'HV', 'Cable', 2000, 2016, 'good'
* Insert the record for line 14: - 14, 15, 3, 'MV', 'Cable', 1976, 2002, 'bad'

-- Create the table

CREATE TABLE structure (

-- Define the field EquipmentID

EquipmentID int NOT null PRIMARY KEY,

EquipmentID\_To INT ,

EquipmentID\_From INT,

VoltageLevel TEXT NOT NULL,

Description TEXT NOT NULL,

ConstructionYear INT NOT NULL,

InspectionYear INT NOT NULL,

ConditionAssessment TEXT NOT NULL

);

-- Insert the record for line 1 as described

INSERT INTO structure

VALUES ( 1, 2, NULL, 'HV', 'Cable', 2000, 2016, 'good');

INSERT INTO Structure

VALUES ( 2, 3 , 1, 'HV', 'Overhead Line', 1968, 2016, 'bad');

INSERT INTO Structure

VALUES ( 3, 14, 2, 'HV', 'TRANSFORMER', 1972, 2001, 'good');

-- Insert the record for line 14 as described

INSERT INTO Structure

VALUES ( 14, 15, 3 , 'MV', 'Cable', 1976, '2002', 'bad');

SELECT \*

FROM structure

* Define the CTE maintenance\_List.
* Start the evaluation for line 3.
* Join GridStructure with maintenance\_List on the corresponding endpoints.
* Use LIKE to filter the power lines with ConditionAssessment of either exchange or repair, and a VoltageLevel of HV

-- Define the table CTE

WITH maintenance\_List (Line, Destination, Source, Description, ConditionAssessment, VoltageLevel) AS (

SELECT

EquipmentID,

EquipmentID\_To,

EquipmentID\_From,

Description,

ConditionAssessment,

VoltageLevel

FROM GridStructure

-- Start the evaluation for line 3

WHERE EquipmentID = 3

UNION ALL

SELECT

Child.EquipmentID,

Child.EquipmentID\_To,

Child.EquipmentID\_FROM,

Child.Description,

Child.ConditionAssessment,

Child.VoltageLevel

FROM GridStructure Child

-- Join GridStructure with CTE on the corresponding endpoints

Join maintenance\_List

ON maintenance\_List.Line = Child.EquipmentID\_FROM)

SELECT Line, Description, ConditionAssessment

FROM maintenance\_List

-- Filter the lines based on ConditionAssessment and VoltageLevel

WHERE

(ConditionAssessment LIKE '%exchange%' OR ConditionAssessment LIKE '%repair%') AND

VoltageLevel LIKE '%HV%'