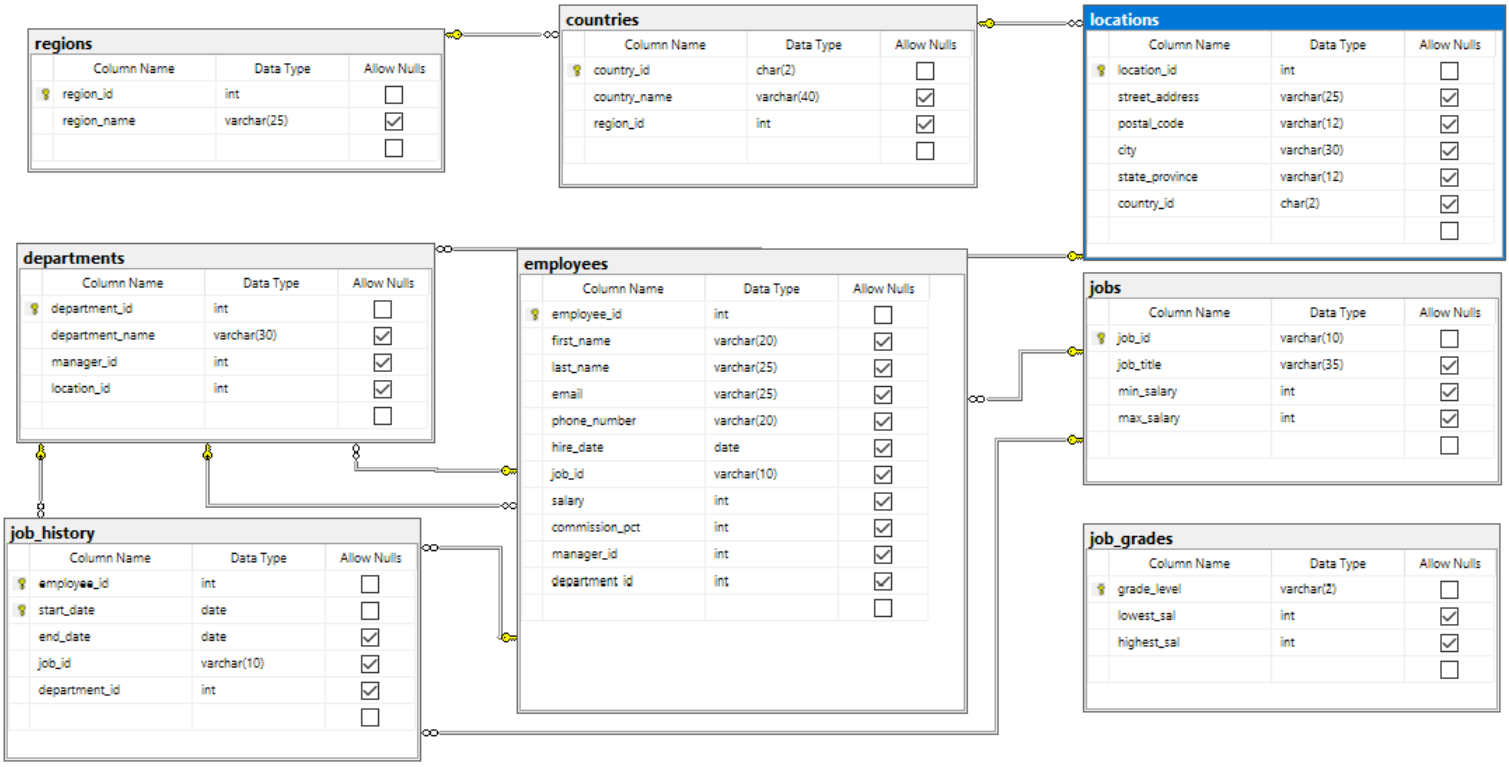
1. Database sample: Human Resource (HR) Database



2. User-defined functions and Stored procedures in SQLServer

A user-defined function is a Transact-SQL or common language runtime (CLR) routine that accepts parameters, performs an action, such as a complex calculation, and returns the result of that action as a value. The return value can either be a scalar (single) value or a table.

CREATE FUNCTION (Transact-SQL) - SQL Server | Microsoft Docs

```
CREATE [ OR ALTER ] FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ][ type_schema_name. ] parameter_data_type
 [ = default ] [ READONLY ] }
    [ ,...n ]
  ]
)
RETURNS return_data_type
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    BEGIN
        function_body
        RETURN scalar_expression
    END

[ ; ]
```

Example:

```
CREATE FUNCTION deptInfos(@deptid int)
RETURNS TABLE
```

```
AS
RETURN
(
    SELECT d.department_id,d.department_name, count(e.employee_id) 'EmpNumber'
    FROM departments d left join employees e on d.department_id = e.employee_id
       WHERE d.department_id = @deptid
       GROUP BY d.department_id, d.department_name
       );
GO


select * from deptInfos(1);



CREATE OR ALTER FUNCTION getTotalSalary(@deptID int)
RETURNS int
AS
-- Returns the total salary of a department
BEGIN
    DECLARE @ret int;
    SELECT @ret = sum(salary)
    FROM employees
    WHERE department_id = @deptID ;
       IF (@ret IS NULL)
        SET @ret = 0;
    RETURN @ret;
END;

GO
select *, dbo.getTotalSalary(department_id) as  totalsalary

from departments;
```

A stored procedure in SQL Server is a group of one or more Transact-SQL statements or a reference to a Microsoft .NET Framework common runtime language (CLR) method. Procedures can:

- **Accept input parameters** and return multiple values in the form of **output parameters** to the calling program.
- Contain **programming statements** that perform operations in the database. These include calling other procedures.
- **Return a status value** to a calling program to indicate success or failure (and the reason for failure).

Create: CREATE PROCEDURE (Transact-SQL) - SQL Server | Microsoft Docs

```
CREATE [ OR ALTER ] { PROC | PROCEDURE }
    [schema_name.] procedure_name [ ; number ]
    [ { @parameter [ type_schema_name. ] data_type }
        [ VARYING ] [ = default ] [ OUT | OUTPUT | [READONLY]
    ] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
[ FOR REPLICATION ]
AS { [ BEGIN ] sql_statement [;] [ ...n ] [ END ] }
[;]
```

```
<procedure_option> ::=
    [ ENCRYPTION ]
    [ RECOMPILE ]
    [ EXECUTE AS Clause ]
```

Exemple:

```
USE HR_test;

CREATE OR ALTER PROCEDURE getEmpInfos
    @LastName nvarchar(30),
    @FirstName nvarchar(30)
AS
    SET NOCOUNT ON;  -- Stops the message that shows the count of the number of rows affected by
a Transact-SQL statement
    SELECT first_name, last_name, salary
    FROM employees
    WHERE first_name = @FirstName AND last_name = @LastName ;
GO



CREATE OR ALTER PROC What_DB_is_this
AS
SELECT DB_NAME() AS ThisDB;
```

Execute a stored procedure:

```
EXECUTE getEmpInfos 'Steven', 'King';
-- Or
EXEC getEmpInfos @LastName = 'Stenven', @FirstName = 'King';

-- Or
EXECUTE getEmpInfos @FirstName = 'King', @LastName = 'Steven';
```

Delete a procedure:

```
DROP PROCEDURE <stored procedure name>;
```

3. **SQL Server triggers**

   SQL Server triggers are special stored procedures that are *executed automatically* in response to the database object, database, and server events. SQL Server provides three type of triggers:

   - Data manipulation language (DML) triggers which are invoked automatically in response to INSERT, UPDATE, and DELETE events against tables.
   - Data definition language (DDL) triggers which fire in response to CREATE, ALTER, and DROP statements. DDL triggers also fire in response to some system stored procedures that perform DDL-like operations.
   - Logon triggers which fire in response to LOGON events

CREATE TRIGGER (Transact-SQL) - SQL Server | Microsoft Docs

Data manipulation language (DML) triggers:

Create DML Triggers - SQL Server | Microsoft Docs

```
-- SQL Server Syntax
-- Trigger on an INSERT, UPDATE, or DELETE statement to a table or view (DML Trigger)

CREATE [ OR ALTER ] TRIGGER [ schema_name . ]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS { sql_statement  [ ; ] [ ,...n ] | EXTERNAL NAME <method specifier [ ; ] > }


DISABLE TRIGGER [schema_name.][trigger_name]
ON [object_name | DATABASE | ALL SERVER];


ENABLE TRIGGER [schema_name.][trigger_name]
ON [object_name | DATABASE | ALL SERVER]


DROP TRIGGER [ IF EXISTS ] [schema_name.]trigger_name [ ,...n ];


-- Trigger on an INSERT, UPDATE, or DELETE statement to a table or view (DML Trigger)

DROP TRIGGER [ IF EXISTS ] [schema_name.]trigger_name [ ,...n ] [ ; ]

-- Trigger on a CREATE, ALTER, DROP, GRANT, DENY, REVOKE or UPDATE statement (DDL Trigger)

DROP TRIGGER [ IF EXISTS ] trigger_name [ ,...n ]
ON { DATABASE | ALL SERVER }
[ ; ]

-- Trigger on a LOGON event (Logon Trigger)

DROP TRIGGER [ IF EXISTS ] trigger_name [ ,...n ]
ON ALL SERVER
```

FOR or AFTER specifies that the DML trigger fires only when all operations specified in the triggering SQL statement have launched successfully. All referential cascade actions and constraint checks must also succeed before this trigger fires.

You can't define AFTER triggers on views.

INSTEAD OF

An INSTEAD OF trigger is a trigger that allows you to skip an INSERT, DELETE, or UPDATE statement to a table or a view and execute other statements defined in the trigger instead. The actual insert, delete, or update operation does not occur at all.

"Virtual" tables for triggers: INSERTED and DELETED

SQL Server provides two virtual tables that are available specifically for triggers called INSERTED and DELETED tables. SQL Server uses these tables to capture the data of the modified row before and after the event occurs.

The following table shows the content of the INSERTED and DELETED tables before and after each event:

DML event	INSERTED table holds	DELETED table holds
INSERT	rows to be inserted	empty
UPDATE	new rows modified by the update	existing rows modified by the update
DELETE	empty	rows to be deleted

```
-- TRIGGERS
USE HR_test;
GO
CREATE OR ALTER TRIGGER tg_test_insert
ON employees
AFTER INSERT AS
BEGIN
	DECLARE @nbrinsertedrow int;

	select @nbrinsertedrow = count(*)
	from inserted;

	--SET NOCOUNT ON;
	PRINT 'Trigger is fired '
	PRINT '@nbrinsertedrow = ' + CAST(@nbrinsertedrow as varchar(3));

	IF @@ROWCOUNT = 1
		BEGIN
		   PRINT 'There is one inserted row';
		END
	ELSE
		BEGIN
		   PRINT 'There are ' + CAST(@nbrinsertedrow as varchar(3)) + ' inserted rows.' ;
		   select * from inserted;
		END;
END;

GO


-- test
delete from employees where employee_id >= 500;
```

```sql
--insert into employees(employee_id,salary) values(550, 5000);
insert into employees(employee_id,salary) values(550, 5000), (558, 2000);
select * from employees where employee_id >= 500;


DROP TRIGGER IF EXISTS tg_test_insert ;


-- instead of trigger
DROP TRIGGER IF EXISTS tg_insteadofinsert ;
CREATE OR ALTER TRIGGER tg_insteadofinsert
ON employees
INSTEAD OF INSERT AS
       PRINT 'Trigger is fired. Row can not be inserted into employees table!'
GO

-- test
delete from employees where employee_id >= 500;
--insert into employees(employee_id,salary) values(550, 5000);
insert into employees(employee_id,salary) values(550, 5000), (558, 2000);
select * from employees where employee_id >= 500;
```

4. **Practices:**
   - **Department table**
     - **Create a new column to store the average salary in each department**
     - **Create a new column for store the number of employees in each department**
   - **Define triggers to update automatically for these 2 columns?**