**DOCENTE:** Mora Bonilla, Anthony Vladimir

**INTEGRANTES:**

* Aquino Carhuas, Marc Andreessen U23221709
* Canto Alayo, Carla Ariana U23248490
* Mayo Calderon, Jorge Octavio U24233712

**CASO PRÁCTICO**

La Institución Educativa San Martín de Porres requiere gestionar su base de datos académica. El sistema debe considerar:

1.- Registro de estudiantes

2.- Registro de docentes

3.- Registro de cursos

4.- Registro de matrículas

5.- Registro de notas por curso

**ACTIVIDADES A DESARROLLAR**

1. **Analizar qué información es importante**

Para la institución San Martín de Porres, se determinó que es esencial gestionar:

* Alumnos

Datos personales (nombres, apellidos, DNI, fecha de nacimiento, estado).

* Docentes

Información personal, asignación de cursos y horas.

* Cursos

Nombre, descripción, estado.

* Matrículas

Relación entre estudiantes y cursos, montos, descuentos, detalle de asignación.

* Notas

Calificaciones obtenidas por los estudiantes en cada curso, con restricción de máximo 20.

1. **Crear las tablas y poblarlas con datos**

* **Identificación de entidades y atributos**

Se han identificado las siguientes entidades principales:

* PEOPLE (Registro de personas)
* STUDENTS (Registro de estudiantes)
* TEACHERS (Registro de docentes)
* COURSES (Registro de cursos)
* TEACHERS\_BY\_COURSE (Asignación de cursos a docentes)
* ENROLLMENTS (Matrículas de estudiantes)
* ENROLLMENT\_DETAILS (Detalle de cursos matriculados)
* **Definición de llaves primarias y foráneas**
* **Tabla PEOPLE (Registro de personas)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Campo** | **Tipo de Dato** | **Clave** | **Descripción** |
| id | SERIAL | PK | Identificador único de la persona |
| dni | VARCHAR(8) UNIQUE | UK | Documento nacional de identidad |
| names | VARCHAR(500) |  | Nombres de la persona |
| father\_last\_name | VARCHAR(500) |  | Apellido paterno |
| mother\_last\_name | VARCHAR(500) |  | Apellido materno |
| gender | BOOLEAN DEFAULT TRUE |  | Género (true=masculino, false=femenino) |
| status | BOOLEAN DEFAULT TRUE |  | Estado de la persona (activo/inactivo) |
| created\_at | TIMESTAMP WITH TIME ZONE |  | Fecha de creación |
| updated\_at | TIMESTAMP WITH TIME ZONE |  | Última actualización |

* **Tabla STUDENTS (Registro de estudiantes)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Campo** | **Tipo de Dato** | **Clave** | **Descripción** |
| Id | SERIAL | PK | Identificador del estudiante |
| person\_id | INT REFERENCES people(id) | FK | Persona asociada al estudiante |
| Status | VARCHAR(2) |  | Estado del alumno (ej.: A=activo) |
| birth\_date | DATE |  | Fecha de nacimiento |
| created\_at | TIMESTAMP WITH TIME ZONE |  | Fecha de creación |
| updated\_at | TIMESTAMP WITH TIME ZONE |  | Última actualización |

* **Tabla TEACHERS (Registro de docentes)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Campo** | **Tipo de Dato** | **Clave** | **Descripción** |
| Id | SERIAL | PK | Identificador del docente |
| person\_id | INT REFERENCES people(id) | FK | Persona asociada al docente |
| Status | BOOLEAN DEFAULT TRUE |  | Estado del docente |
| created\_at | TIMESTAMP WITH TIME ZONE |  | Fecha de creación |
| updated\_at | TIMESTAMP WITH TIME ZONE |  | Última actualización |

* **Tabla COURSES (Registro de cursos)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Campo** | **Tipo de Dato** | **Clave** | **Descripción** |
| Id | SERIAL | PK | Identificador del curso |
| Name | VARCHAR(500) |  | Nombre del curso |
| Description | VARCHAR(500) |  | Descripción del curso |
| Status | BOOLEAN DEFAULT TRUE |  | Estado del curso (activo/inactivo) |
| created\_at | TIMESTAMP WITH TIME ZONE |  | Fecha de creación |
| updated\_at | TIMESTAMP WITH TIME ZONE |  | Última actualización |

* **Tabla TEACHERS\_BY\_COURSE (Asignación de cursos a docentes)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Campo** | **Tipo de Dato** | **Clave** | **Descripción** |
| Id | SERIAL | PK | Identificador de asignación |
| teacher\_id | INT REFERENCES teachers(id) | FK | Docente asignado |
| course\_id | INT REFERENCES courses(id) | FK | Curso asignado |
| quantity\_hours | INT |  | Número de horas asignadas |
| Status | BOOLEAN DEFAULT TRUE |  | Estado de la asignación |
| created\_at | TIMESTAMP WITH TIME ZONE |  | Fecha de creación |
| updated\_at | TIMESTAMP WITH TIME ZONE |  | Última actualización |

* **Tabla ENROLLMENTS (Matrículas de estudiantes)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Campo** | **Tipo de Dato** | **Clave** | **Descripción** |
| Id | SERIAL | PK | Identificador de matrícula |
| student\_id | INT REFERENCES students(id) | FK | Alumno matriculado |
| Amount | NUMERIC |  | Monto total de la matrícula |
| Disacount | NUMERIC |  | Descuento aplicado |
| quantity\_courses | INT |  | Número de cursos matriculados |
| Status | BOOLEAN DEFAULT TRUE |  | Estado de la matrícula |
| created\_at | TIMESTAMP WITH TIME ZONE |  | Fecha de creación |
| updated\_at | TIMESTAMP WITH TIME ZONE |  | Última actualización |

* **Tabla ENROLLMENT\_DETAILS (Detalle de cursos matriculados)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Campo** | **Tipo de Dato** | **Clave** | **Descripción** |
| Id | SERIAL | PK | Identificador del detalle de matrícula |
| enrollment\_id | INT REFERENCES enrollments(id) | FK | Matrícula asociada |
| course\_assigned\_id | INT REFERENCES teachers\_by\_course(id) | FK | Curso asignado al estudiante con su docente |
| Score | NUMERIC |  | Nota obtenida en el curso (0-20) |
| Status | BOOLEAN DEFAULT TRUE |  | Estado del detalle |
| created\_at | TIMESTAMP WITH TIME ZONE |  | Fecha de creación |
| updated\_at | TIMESTAMP WITH TIME ZONE |  | Última actualización |

* **Hacer consultas (ejemplo: listar alumnos por curso).**
* **Listar alumnos por curso:**

SELECT c.name AS curso,  
 p.names || ' ' || p.father\_last\_name || ' ' || p.mother\_last\_name AS alumno  
FROM enrollment\_details ed  
JOIN enrollments e ON e.id = ed.enrollment\_id  
JOIN students s ON s.id = e.student\_id  
JOIN people p ON p.id = s.person\_id  
JOIN teachers\_by\_course tbc ON tbc.id = ed.course\_assigned\_id  
JOIN courses c ON c.id = tbc.course\_id  
ORDER BY c.name, alumno;

* **Cantidad de alumnos por curso:**

SELECT c.name AS curso,  
 COUNT(s.id) AS cantidad\_alumnos  
FROM enrollment\_details ed  
JOIN enrollments e ON e.id = ed.enrollment\_id  
JOIN students s ON s.id = e.student\_id  
JOIN teachers\_by\_course tbc ON tbc.id = ed.course\_assigned\_id  
JOIN courses c ON c.id = tbc.course\_id  
GROUP BY c.name  
ORDER BY cantidad\_alumnos DESC;

* **Alumnos por curso y docente:**

SELECT c.name AS curso,  
 p\_student.names || ' ' || p\_student.father\_last\_name AS alumno,  
 p\_teacher.names || ' ' || p\_teacher.father\_last\_name AS docente  
FROM enrollment\_details ed  
JOIN enrollments e ON e.id = ed.enrollment\_id  
JOIN students s ON s.id = e.student\_id  
JOIN people p\_student ON p\_student.id = s.person\_id  
JOIN teachers\_by\_course tbc ON tbc.id = ed.course\_assigned\_id  
JOIN teachers t ON t.id = tbc.teacher\_id  
JOIN people p\_teacher ON p\_teacher.id = t.person\_id  
JOIN courses c ON c.id = tbc.course\_id  
ORDER BY c.name, alumno;

* **Alumnos de Matemáticas:**

SELECT c.name AS curso,  
 p.names || ' ' || p.father\_last\_name AS alumno  
FROM enrollment\_details ed  
JOIN enrollments e ON e.id = ed.enrollment\_id  
JOIN students s ON s.id = e.student\_id  
JOIN people p ON p.id = s.person\_id  
JOIN teachers\_by\_course tbc ON tbc.id = ed.course\_assigned\_id  
JOIN courses c ON c.id = tbc.course\_id  
WHERE c.name = 'Matemáticas'  
ORDER BY alumno;

* **Crear un procedimiento almacenado (ejemplo: registrar matrícula).**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* PEOPLE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DROP FUNCTION IF EXISTS public.fx\_sel\_people(JSONB);

CREATE FUNCTION public.fx\_sel\_people(JSONB)

RETURNS TABLE (

id INT,

dni VARCHAR,

names VARCHAR,

father\_last\_name VARCHAR,

mother\_last\_name VARCHAR,

gender BOOLEAN,

status BOOLEAN,

created\_at TIMESTAMP WITH TIME ZONE,

updated\_at TIMESTAMP WITH TIME ZONE

)

AS $$

DECLARE

p\_json\_data ALIAS FOR $1;

BEGIN

RETURN QUERY

WITH filtros AS (

SELECT

x.id,

x.dni,

x.names,

x.father\_last\_name,

x.mother\_last\_name,

x.gender,

x.status,

x.created\_at

FROM JSONB\_TO\_RECORDSET(COALESCE(p\_json\_data,'[]'::JSONB)) AS x(

id INT,

dni VARCHAR(8),

names VARCHAR(500),

father\_last\_name VARCHAR(500),

mother\_last\_name VARCHAR(500),

gender BOOLEAN,

status BOOLEAN,

created\_at TIMESTAMP WITH TIME ZONE

)

)

SELECT

p.id,

p.dni,

p.names,

p.father\_last\_name,

p.mother\_last\_name,

p.gender,

p.status,

p.created\_at,

p.updated\_at

FROM people p

LEFT JOIN filtros f ON TRUE

WHERE (f.id IS NULL OR p.id = f.id)

AND (f.dni IS NULL OR p.dni ILIKE '%' || f.dni || '%')

AND (f.names IS NULL OR p.names ILIKE '%' || f.names || '%')

AND (f.father\_last\_name IS NULL OR p.father\_last\_name ILIKE '%' || f.father\_last\_name || '%')

AND (f.mother\_last\_name IS NULL OR p.mother\_last\_name ILIKE '%' || f.mother\_last\_name || '%')

AND (f.gender IS NULL OR p.gender = f.gender)

AND (f.status IS NULL OR p.status = f.status)

AND (f.created\_at IS NULL OR DATE(p.created\_at) = DATE(f.created\_at));

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER COST 1000;

COMMENT ON FUNCTION public.fx\_sel\_people(JSONB) IS

$$

OBJETIVO : Consultar registros en people con filtros opcionales enviados en JSONB

SINTAXIS DE EJEMPLO:

-- Todos los registros

SELECT \* FROM public.fx\_sel\_people(NULL);

-- Filtrar por DNI

SELECT \* FROM public.fx\_sel\_people('[{"dni":"12345678"}]'::jsonb);

$$;

DROP FUNCTION IF EXISTS public.fx\_ins\_people(JSONB);

CREATE FUNCTION public.fx\_ins\_people(JSONB)

RETURNS TABLE (

id INT,

dni VARCHAR

)

AS $$

DECLARE

p\_json\_data ALIAS FOR $1;

BEGIN

DROP TABLE IF EXISTS tmp\_people;

CREATE TEMPORARY TABLE tmp\_people AS

SELECT

x.dni,

x.names,

x.father\_last\_name,

x.mother\_last\_name,

x.gender,

x.status,

x.created\_at,

x.updated\_at

FROM JSONB\_TO\_RECORDSET(COALESCE(p\_json\_data,'[]'::JSONB)) AS x(

dni VARCHAR(8),

names VARCHAR(500),

father\_last\_name VARCHAR(500),

mother\_last\_name VARCHAR(500),

gender BOOLEAN,

status BOOLEAN,

created\_at TIMESTAMP WITH TIME ZONE,

updated\_at TIMESTAMP WITH TIME ZONE

);

RETURN QUERY

INSERT INTO people(

dni,

names,

father\_last\_name,

mother\_last\_name,

gender,

status,

created\_at,

updated\_at

)

SELECT

TRIM(B.dni),

INITCAP(TRIM(B.names)),

INITCAP(TRIM(B.father\_last\_name)),

INITCAP(TRIM(B.mother\_last\_name)),

COALESCE(B.gender, TRUE),

COALESCE(B.status, TRUE),

COALESCE(B.created\_at, CURRENT\_TIMESTAMP),

B.updated\_at

FROM tmp\_people B

RETURNING id, dni;

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER COST 1000;

COMMENT ON FUNCTION public.fx\_ins\_people(JSONB) IS

$$

OBJETIVO : Insertar registros en people desde JSONB

SINTAXIS:

SELECT \* FROM public.fx\_ins\_people(

'[{"dni":"12345678","names":"Juan Perez","father\_last\_name":"Perez","mother\_last\_name":"Lopez","gender":true,"created\_at":"2025-01-01T10:00:00Z","status":true}]'::jsonb

);

$$;

DROP FUNCTION IF EXISTS public.fx\_upd\_people(JSONB);

CREATE FUNCTION public.fx\_upd\_people(JSONB)

RETURNS BOOLEAN

AS $$

DECLARE

p\_json\_data ALIAS FOR $1;

BEGIN

DROP TABLE IF EXISTS tmp\_people\_upd;

CREATE TEMPORARY TABLE tmp\_people\_upd AS

SELECT

x.id,

x.dni,

x.names,

x.father\_last\_name,

x.mother\_last\_name,

x.gender,

x.status,

x.updated\_at

FROM JSONB\_TO\_RECORDSET(COALESCE(p\_json\_data,'[]'::JSONB)) AS x(

id INT,

dni VARCHAR(8),

names VARCHAR(500),

father\_last\_name VARCHAR(500),

mother\_last\_name VARCHAR(500),

gender BOOLEAN,

status BOOLEAN,

updated\_at TIMESTAMP WITH TIME ZONE

);

UPDATE people p

SET

dni = COALESCE(TRIM(u.dni), p.dni),

names = COALESCE(INITCAP(TRIM(u.names)), p.names),

father\_last\_name = COALESCE(INITCAP(TRIM(u.father\_last\_name)), p.father\_last\_name),

mother\_last\_name = COALESCE(INITCAP(TRIM(u.mother\_last\_name)), p.mother\_last\_name),

gender = COALESCE(u.gender, p.gender),

status = COALESCE(u.status, p.status),

updated\_at = COALESCE(u.updated\_at, CURRENT\_TIMESTAMP)

FROM tmp\_people\_upd u

WHERE p.id = u.id;

RETURN TRUE;

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER COST 1000;

COMMENT ON FUNCTION public.fx\_upd\_people(JSONB) IS

$$

OBJETIVO : Actualizar registros en people (actualización parcial mediante JSONB)

SINTAXIS:

SELECT public.fx\_upd\_people('[{"id":1,"names":"Juanito Perez","updated\_at":"2025-08-01T12:00:00Z"}]'::jsonb);

$$;

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- TEACHERS

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DROP FUNCTION IF EXISTS public.fx\_sel\_teachers(JSONB);

CREATE FUNCTION public.fx\_sel\_teachers(JSONB)

RETURNS TABLE (

id INT,

person\_id INT,

dni VARCHAR,

names VARCHAR,

father\_last\_name VARCHAR,

mother\_last\_name VARCHAR,

status BOOLEAN,

created\_at TIMESTAMP WITH TIME ZONE,

updated\_at TIMESTAMP WITH TIME ZONE

)

AS $$

DECLARE

p\_json\_data ALIAS FOR $1;

BEGIN

RETURN QUERY

WITH filtros AS (

SELECT

x.id,

x.person\_id,

x.status,

x.created\_at

FROM JSONB\_TO\_RECORDSET(COALESCE(p\_json\_data,'[]'::JSONB)) AS x(

id INT,

person\_id INT,

status BOOLEAN,

created\_at TIMESTAMP WITH TIME ZONE

)

)

SELECT

t.id,

t.person\_id,

p.dni,

p.names,

p.father\_last\_name,

p.mother\_last\_name,

t.status,

t.created\_at,

t.updated\_at

FROM teachers t

LEFT JOIN people p ON p.id = t.person\_id

LEFT JOIN filtros f ON TRUE

WHERE (f.id IS NULL OR t.id = f.id)

AND (f.person\_id IS NULL OR t.person\_id = f.person\_id)

AND (f.status IS NULL OR t.status = f.status)

AND (f.created\_at IS NULL OR DATE(t.created\_at) = DATE(f.created\_at));

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER COST 1000;

COMMENT ON FUNCTION public.fx\_sel\_teachers(JSONB) IS

$$

OBJETIVO : Consultar teachers (posible join con people)

SINTAXIS:

SELECT \* FROM public.fx\_sel\_teachers(NULL);

SELECT \* FROM public.fx\_sel\_teachers('[{"person\_id":2}]'::jsonb);

$$;

DROP FUNCTION IF EXISTS public.fx\_ins\_teachers(JSONB);

CREATE FUNCTION public.fx\_ins\_teachers(JSONB)

RETURNS BOOLEAN

AS $$

DECLARE

p\_json\_data ALIAS FOR $1;

BEGIN

DROP TABLE IF EXISTS tmp\_teachers;

CREATE TEMPORARY TABLE tmp\_teachers AS

SELECT

x.person\_id,

x.status,

x.created\_at,

x.updated\_at

FROM JSONB\_TO\_RECORDSET(COALESCE(p\_json\_data,'[]'::JSONB)) AS x(

person\_id INT,

status BOOLEAN,

created\_at TIMESTAMP WITH TIME ZONE,

updated\_at TIMESTAMP WITH TIME ZONE

);

INSERT INTO teachers(

person\_id,

status,

created\_at,

updated\_at

)

SELECT

person\_id,

COALESCE(status, TRUE),

COALESCE(created\_at, CURRENT\_TIMESTAMP),

updated\_at

FROM tmp\_teachers;

RETURN TRUE;

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER COST 1000;

COMMENT ON FUNCTION public.fx\_ins\_teachers(JSONB) IS

$$

OBJETIVO : Insertar registros en teachers

SINTAXIS:

SELECT public.fx\_ins\_teachers('[{"person\_id":1,"status":true}]'::jsonb);

$$;

DROP FUNCTION IF EXISTS public.fx\_upd\_teachers(JSONB);

CREATE FUNCTION public.fx\_upd\_teachers(JSONB)

RETURNS BOOLEAN

AS $$

DECLARE

p\_json\_data ALIAS FOR $1;

BEGIN

DROP TABLE IF EXISTS tmp\_teachers\_upd;

CREATE TEMPORARY TABLE tmp\_teachers\_upd AS

SELECT

x.id,

x.person\_id,

x.status,

x.updated\_at

FROM JSONB\_TO\_RECORDSET(COALESCE(p\_json\_data,'[]'::JSONB)) AS x(

id INT,

person\_id INT,

status BOOLEAN,

updated\_at TIMESTAMP WITH TIME ZONE

);

UPDATE teachers t

SET

person\_id = COALESCE(u.person\_id, t.person\_id),

status = COALESCE(u.status, t.status),

updated\_at = COALESCE(u.updated\_at, CURRENT\_TIMESTAMP)

FROM tmp\_teachers\_upd u

WHERE t.id = u.id;

RETURN TRUE;

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER COST 1000;

COMMENT ON FUNCTION public.fx\_upd\_teachers(JSONB) IS

$$

OBJETIVO : Actualizar registros en teachers (parcial)

SINTAXIS:

SELECT public.fx\_upd\_teachers('[{"id":1,"status":false}]'::jsonb);

$$;

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- STUDENTS

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DROP FUNCTION IF EXISTS public.fx\_sel\_students(JSONB);

CREATE FUNCTION public.fx\_sel\_students(JSONB)

RETURNS TABLE (

id INT,

person\_id INT,

dni VARCHAR,

names VARCHAR,

father\_last\_name VARCHAR,

mother\_last\_name VARCHAR,

status VARCHAR,

birth\_date DATE,

created\_at TIMESTAMP WITH TIME ZONE,

updated\_at TIMESTAMP WITH TIME ZONE

)

AS $$

DECLARE

p\_json\_data ALIAS FOR $1;

v\_person\_id INT;

v\_status TEXT;

BEGIN

-- Permitir filtro por person\_id y status (texto) si se quiere

SELECT x.person\_id, x.status INTO v\_person\_id, v\_status

FROM JSONB\_TO\_RECORD(COALESCE(p\_json\_data,'{}'::JSONB)) AS x(person\_id INT, status VARCHAR);

RETURN QUERY

SELECT

s.id,

s.person\_id,

p.dni,

p.names,

p.father\_last\_name,

p.mother\_last\_name,

s.status,

s.birth\_date,

s.created\_at,

s.updated\_at

FROM students s

INNER JOIN people p ON p.id = s.person\_id

WHERE (v\_person\_id IS NULL OR s.person\_id = v\_person\_id)

AND (v\_status IS NULL OR s.status = v\_status);

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER COST 1000;

COMMENT ON FUNCTION public.fx\_sel\_students(JSONB) IS

$$

OBJETIVO : Consultar students con join a people

SINTAXIS:

SELECT \* FROM public.fx\_sel\_students(NULL);

SELECT \* FROM public.fx\_sel\_students('{"person\_id":1}'::jsonb);

$$;

DROP FUNCTION IF EXISTS public.fx\_ins\_students(JSONB);

CREATE FUNCTION public.fx\_ins\_students(JSONB)

RETURNS BOOLEAN

AS $$

DECLARE

p\_json\_data ALIAS FOR $1;

BEGIN

DROP TABLE IF EXISTS tmp\_students;

CREATE TEMPORARY TABLE tmp\_students AS

SELECT

x.person\_id,

x.status,

x.birth\_date,

x.created\_at,

x.updated\_at

FROM JSONB\_TO\_RECORDSET(COALESCE(p\_json\_data,'[]'::JSONB)) AS x(

person\_id INT,

status VARCHAR(2),

birth\_date DATE,

created\_at TIMESTAMP WITH TIME ZONE,

updated\_at TIMESTAMP WITH TIME ZONE

);

INSERT INTO students(

person\_id,

status,

birth\_date,

created\_at,

updated\_at

)

SELECT

person\_id,

status,

birth\_date,

COALESCE(created\_at, CURRENT\_TIMESTAMP),

updated\_at

FROM tmp\_students;

RETURN TRUE;

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER COST 1000;

COMMENT ON FUNCTION public.fx\_ins\_students(JSONB) IS

$$

OBJETIVO : Insertar registros en students

SINTAXIS:

SELECT public.fx\_ins\_students('[{"person\_id":2,"status":"AC","birth\_date":"2010-05-20","created\_at":"2025-01-01T10:00:00Z"}]'::jsonb);

$$;

DROP FUNCTION IF EXISTS public.fx\_upd\_students(JSONB);

CREATE FUNCTION public.fx\_upd\_students(JSONB)

RETURNS BOOLEAN

AS $$

DECLARE

p\_json\_data ALIAS FOR $1;

BEGIN

DROP TABLE IF EXISTS tmp\_students\_upd;

CREATE TEMPORARY TABLE tmp\_students\_upd AS

SELECT

x.id,

x.person\_id,

x.status,

x.birth\_date,

x.updated\_at

FROM JSONB\_TO\_RECORDSET(COALESCE(p\_json\_data,'[]'::JSONB)) AS x(

id INT,

person\_id INT,

status VARCHAR(2),

birth\_date DATE,

updated\_at TIMESTAMP WITH TIME ZONE

);

UPDATE students s

SET

person\_id = COALESCE(u.person\_id, s.person\_id),

status = COALESCE(u.status, s.status),

birth\_date = COALESCE(u.birth\_date, s.birth\_date),

updated\_at = COALESCE(u.updated\_at, CURRENT\_TIMESTAMP)

FROM tmp\_students\_upd u

WHERE s.id = u.id;

RETURN TRUE;

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER COST 1000;

COMMENT ON FUNCTION public.fx\_upd\_students(JSONB) IS

$$

OBJETIVO : Actualizar students (parcial)

SINTAXIS:

SELECT public.fx\_upd\_students('[{"id":1,"status":"IN","updated\_at":"2025-08-01T12:00:00Z"}]'::jsonb);

$$;

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- COURSES

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DROP FUNCTION IF EXISTS public.fx\_sel\_courses(JSONB);

CREATE FUNCTION public.fx\_sel\_courses(JSONB)

RETURNS TABLE (

id INT,

name VARCHAR,

description VARCHAR,

status BOOLEAN,

created\_at TIMESTAMP WITH TIME ZONE,

updated\_at TIMESTAMP WITH TIME ZONE

)

AS $$

DECLARE

p\_json\_data ALIAS FOR $1;

BEGIN

RETURN QUERY

WITH filtros AS (

SELECT

x.id,

x.name,

x.status,

x.created\_at

FROM JSONB\_TO\_RECORDSET(COALESCE(p\_json\_data,'[]'::JSONB)) AS x(

id INT,

name VARCHAR(500),

status BOOLEAN,

created\_at TIMESTAMP WITH TIME ZONE

)

)

SELECT

c.id,

c.name,

c.description,

c.status,

c.created\_at,

c.updated\_at

FROM courses c

LEFT JOIN filtros f ON TRUE

WHERE (f.id IS NULL OR c.id = f.id)

AND (f.name IS NULL OR c.name ILIKE '%' || f.name || '%')

AND (f.status IS NULL OR c.status = f.status)

AND (f.created\_at IS NULL OR DATE(c.created\_at) = DATE(f.created\_at));

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER COST 1000;

COMMENT ON FUNCTION public.fx\_sel\_courses(JSONB) IS

$$

OBJETIVO : Consultar courses con filtros JSONB

SINTAXIS:

SELECT \* FROM public.fx\_sel\_courses(NULL);

SELECT \* FROM public.fx\_sel\_courses('[{"name":"Matematica"}]'::jsonb);

$$;

DROP FUNCTION IF EXISTS public.fx\_ins\_courses(JSONB);

CREATE FUNCTION public.fx\_ins\_courses(JSONB)

RETURNS BOOLEAN

AS $$

DECLARE

p\_json\_data ALIAS FOR $1;

BEGIN

DROP TABLE IF EXISTS tmp\_courses;

CREATE TEMPORARY TABLE tmp\_courses AS

SELECT

x.name,

x.description,

x.status,

x.created\_at,

x.updated\_at

FROM JSONB\_TO\_RECORDSET(COALESCE(p\_json\_data,'[]'::JSONB)) AS x(

name VARCHAR(500),

description VARCHAR(500),

status BOOLEAN,

created\_at TIMESTAMP WITH TIME ZONE,

updated\_at TIMESTAMP WITH TIME ZONE

);

INSERT INTO courses(

name,

description,

status,

created\_at,

updated\_at

)

SELECT

INITCAP(TRIM(name)),

description,

COALESCE(status, TRUE),

COALESCE(created\_at, CURRENT\_TIMESTAMP),

updated\_at

FROM tmp\_courses;

RETURN TRUE;

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER COST 1000;

COMMENT ON FUNCTION public.fx\_ins\_courses(JSONB) IS

$$

OBJETIVO : Insertar registros en courses

SINTAXIS:

SELECT public.fx\_ins\_courses('[{"name":"Matemáticas","description":"Basica","status":true}]'::jsonb);

$$;

DROP FUNCTION IF EXISTS public.fx\_upd\_courses(JSONB);

CREATE FUNCTION public.fx\_upd\_courses(JSONB)

RETURNS BOOLEAN

AS $$

DECLARE

p\_json\_data ALIAS FOR $1;

BEGIN

DROP TABLE IF EXISTS tmp\_courses\_upd;

CREATE TEMPORARY TABLE tmp\_courses\_upd AS

SELECT

x.id,

x.name,

x.description,

x.status,

x.updated\_at

FROM JSONB\_TO\_RECORDSET(COALESCE(p\_json\_data,'[]'::JSONB)) AS x(

id INT,

name VARCHAR(500),

description VARCHAR(500),

status BOOLEAN,

updated\_at TIMESTAMP WITH TIME ZONE

);

UPDATE courses c

SET

name = COALESCE(INITCAP(TRIM(u.name)), c.name),

description = COALESCE(u.description, c.description),

status = COALESCE(u.status, c.status),

updated\_at = COALESCE(u.updated\_at, CURRENT\_TIMESTAMP)

FROM tmp\_courses\_upd u

WHERE c.id = u.id;

RETURN TRUE;

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER COST 1000;

COMMENT ON FUNCTION public.fx\_upd\_courses(JSONB) IS

$$

OBJETIVO : Actualizar courses (parcial)

SINTAXIS:

SELECT public.fx\_upd\_courses('[{"id":1,"name":"Matematicas Avanzadas"}]'::jsonb);

$$;

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- TEACHERS\_BY\_COURSE

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DROP FUNCTION IF EXISTS public.fx\_sel\_teachers\_by\_course(JSONB);

CREATE FUNCTION public.fx\_sel\_teachers\_by\_course(JSONB)

RETURNS TABLE (

id INT,

teacher\_id INT,

teacher\_person\_id INT,

teacher\_dni VARCHAR,

teacher\_names VARCHAR,

course\_id INT,

course\_name VARCHAR,

quantity\_hours INT,

status BOOLEAN,

created\_at TIMESTAMP WITH TIME ZONE,

updated\_at TIMESTAMP WITH TIME ZONE

)

AS $$

DECLARE

p\_json\_data ALIAS FOR $1;

BEGIN

RETURN QUERY

WITH filtros AS (

SELECT

x.id,

x.teacher\_id,

x.course\_id,

x.status,

x.created\_at

FROM JSONB\_TO\_RECORDSET(COALESCE(p\_json\_data,'[]'::JSONB)) AS x(

id INT,

teacher\_id INT,

course\_id INT,

status BOOLEAN,

created\_at TIMESTAMP WITH TIME ZONE

)

)

SELECT

tbc.id,

tbc.teacher\_id,

te.person\_id as teacher\_person\_id,

p.dni as teacher\_dni,

p.names as teacher\_names,

tbc.course\_id,

c.name as course\_name,

tbc.quantity\_hours,

tbc.status,

tbc.created\_at,

tbc.updated\_at

FROM teachers\_by\_course tbc

LEFT JOIN teachers te ON te.id = tbc.teacher\_id

LEFT JOIN people p ON p.id = te.person\_id

LEFT JOIN courses c ON c.id = tbc.course\_id

LEFT JOIN filtros f ON TRUE

WHERE (f.id IS NULL OR tbc.id = f.id)

AND (f.teacher\_id IS NULL OR tbc.teacher\_id = f.teacher\_id)

AND (f.course\_id IS NULL OR tbc.course\_id = f.course\_id)

AND (f.status IS NULL OR tbc.status = f.status)

AND (f.created\_at IS NULL OR DATE(tbc.created\_at) = DATE(f.created\_at));

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER COST 1000;

COMMENT ON FUNCTION public.fx\_sel\_teachers\_by\_course(JSONB) IS

$$

OBJETIVO : Consultar teachers\_by\_course con joins a teachers->people y courses

SINTAXIS:

SELECT \* FROM public.fx\_sel\_teachers\_by\_course(NULL);

SELECT \* FROM public.fx\_sel\_teachers\_by\_course('[{"teacher\_id":1}]'::jsonb);

$$;

DROP FUNCTION IF EXISTS public.fx\_ins\_teachers\_by\_course(JSONB);

CREATE FUNCTION public.fx\_ins\_teachers\_by\_course(JSONB)

RETURNS BOOLEAN

AS $$

DECLARE

p\_json\_data ALIAS FOR $1;

BEGIN

DROP TABLE IF EXISTS tmp\_teachers\_by\_course;

CREATE TEMPORARY TABLE tmp\_teachers\_by\_course AS

SELECT

x.teacher\_id,

x.course\_id,

x.quantity\_hours,

x.status,

x.created\_at,

x.updated\_at

FROM JSONB\_TO\_RECORDSET(COALESCE(p\_json\_data,'[]'::JSONB)) AS x(

teacher\_id INT,

course\_id INT,

quantity\_hours INT,

status BOOLEAN,

created\_at TIMESTAMP WITH TIME ZONE,

updated\_at TIMESTAMP WITH TIME ZONE

);

INSERT INTO teachers\_by\_course(

teacher\_id,

course\_id,

quantity\_hours,

status,

created\_at,

updated\_at

)

SELECT

teacher\_id,

COALESCE(course\_id, NULL),

COALESCE(quantity\_hours, 0),

COALESCE(status, TRUE),

COALESCE(created\_at, CURRENT\_TIMESTAMP),

updated\_at

FROM tmp\_teachers\_by\_course;

RETURN TRUE;

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER COST 1000;

COMMENT ON FUNCTION public.fx\_ins\_teachers\_by\_course(JSONB) IS

$$

OBJETIVO : Insertar registros en teachers\_by\_course

SINTAXIS:

SELECT public.fx\_ins\_teachers\_by\_course('[{"teacher\_id":1,"course\_id":2,"quantity\_hours":40}]'::jsonb);

$$;

DROP FUNCTION IF EXISTS public.fx\_upd\_teachers\_by\_course(JSONB);

CREATE FUNCTION public.fx\_upd\_teachers\_by\_course(JSONB)

RETURNS BOOLEAN

AS $$

DECLARE

p\_json\_data ALIAS FOR $1;

BEGIN

DROP TABLE IF EXISTS tmp\_teachers\_by\_course\_upd;

CREATE TEMPORARY TABLE tmp\_teachers\_by\_course\_upd AS

SELECT

x.id,

x.teacher\_id,

x.course\_id,

x.quantity\_hours,

x.status,

x.updated\_at

FROM JSONB\_TO\_RECORDSET(COALESCE(p\_json\_data,'[]'::JSONB)) AS x(

id INT,

teacher\_id INT,

course\_id INT,

quantity\_hours INT,

status BOOLEAN,

updated\_at TIMESTAMP WITH TIME ZONE

);

UPDATE teachers\_by\_course t

SET

teacher\_id = COALESCE(u.teacher\_id, t.teacher\_id),

course\_id = COALESCE(u.course\_id, t.course\_id),

quantity\_hours = COALESCE(u.quantity\_hours, t.quantity\_hours),

status = COALESCE(u.status, t.status),

updated\_at = COALESCE(u.updated\_at, CURRENT\_TIMESTAMP)

FROM tmp\_teachers\_by\_course\_upd u

WHERE t.id = u.id;

RETURN TRUE;

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER COST 1000;

COMMENT ON FUNCTION public.fx\_upd\_teachers\_by\_course(JSONB) IS

$$

OBJETIVO : Actualizar teachers\_by\_course (parcial)

SINTAXIS:

SELECT public.fx\_upd\_teachers\_by\_course('[{"id":1,"quantity\_hours":50}]'::jsonb);

$$;

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- ENROLLMENTS

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DROP FUNCTION IF EXISTS public.fx\_sel\_enrollments(JSONB);

CREATE FUNCTION public.fx\_sel\_enrollments(JSONB)

RETURNS TABLE (

id INT,

student\_id INT,

student\_person\_id INT,

student\_dni VARCHAR,

student\_names VARCHAR,

amount NUMERIC,

disacount NUMERIC,

quantity\_courses INT,

status BOOLEAN,

created\_at TIMESTAMP WITH TIME ZONE,

updated\_at TIMESTAMP WITH TIME ZONE

)

AS $$

DECLARE

p\_json\_data ALIAS FOR $1;

BEGIN

RETURN QUERY

WITH filtros AS (

SELECT

x.id,

x.student\_id,

x.status,

x.created\_at

FROM JSONB\_TO\_RECORDSET(COALESCE(p\_json\_data,'[]'::JSONB)) AS x(

id INT,

student\_id INT,

status BOOLEAN,

created\_at TIMESTAMP WITH TIME ZONE

)

)

SELECT

e.id,

e.student\_id,

s.person\_id as student\_person\_id,

p.dni as student\_dni,

p.names as student\_names,

e.amount,

e.disacount,

e.quantity\_courses,

e.status,

e.created\_at,

e.updated\_at

FROM enrollments e

LEFT JOIN students s ON s.id = e.student\_id

LEFT JOIN people p ON p.id = s.person\_id

LEFT JOIN filtros f ON TRUE

WHERE (f.id IS NULL OR e.id = f.id)

AND (f.student\_id IS NULL OR e.student\_id = f.student\_id)

AND (f.status IS NULL OR e.status = f.status)

AND (f.created\_at IS NULL OR DATE(e.created\_at) = DATE(f.created\_at));

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER COST 1000;

COMMENT ON FUNCTION public.fx\_sel\_enrollments(JSONB) IS

$$

OBJETIVO : Consultar enrollments (join students->people)

SINTAXIS:

SELECT \* FROM public.fx\_sel\_enrollments(NULL);

SELECT \* FROM public.fx\_sel\_enrollments('[{"student\_id":1}]'::jsonb);

$$;

DROP FUNCTION IF EXISTS public.fx\_ins\_enrollments(JSONB);

CREATE FUNCTION public.fx\_ins\_enrollments(JSONB)

RETURNS TABLE (

id INT

)

AS $$

DECLARE

p\_json\_data ALIAS FOR $1;

BEGIN

DROP TABLE IF EXISTS tmp\_enrollments;

CREATE TEMPORARY TABLE tmp\_enrollments AS

SELECT

x.student\_id,

x.amount,

x.disacount,

x.quantity\_courses,

x.status,

x.created\_at,

x.updated\_at

FROM JSONB\_TO\_RECORDSET(COALESCE(p\_json\_data,'[]'::JSONB)) AS x(

student\_id INT,

amount NUMERIC,

disacount NUMERIC,

quantity\_courses INT,

status BOOLEAN,

created\_at TIMESTAMP WITH TIME ZONE,

updated\_at TIMESTAMP WITH TIME ZONE

);

RETURN QUERY

INSERT INTO enrollments(

student\_id,

amount,

disacount,

quantity\_courses,

status,

created\_at,

updated\_at

)

SELECT

student\_id,

COALESCE(amount, 0),

COALESCE(disacount, 0),

COALESCE(quantity\_courses, 0),

COALESCE(status, TRUE),

COALESCE(created\_at, CURRENT\_TIMESTAMP),

updated\_at

FROM tmp\_enrollments

RETURNING id;

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER COST 1000;

COMMENT ON FUNCTION public.fx\_ins\_enrollments(JSONB) IS

$$

OBJETIVO : Insertar enrollments (puede insertar varios registros desde arreglo JSON)

SINTAXIS:

SELECT \* FROM public.fx\_ins\_enrollments('[{"student\_id":1,"amount":200.00,"disacount":10.00,"quantity\_courses":2}]'::jsonb);

$$;

DROP FUNCTION IF EXISTS public.fx\_upd\_enrollments(JSONB);

CREATE FUNCTION public.fx\_upd\_enrollments(JSONB)

RETURNS BOOLEAN

AS $$

DECLARE

p\_json\_data ALIAS FOR $1;

BEGIN

DROP TABLE IF EXISTS tmp\_enrollments\_upd;

CREATE TEMPORARY TABLE tmp\_enrollments\_upd AS

SELECT

x.id,

x.student\_id,

x.amount,

x.disacount,

x.quantity\_courses,

x.status,

x.updated\_at

FROM JSONB\_TO\_RECORDSET(COALESCE(p\_json\_data,'[]'::JSONB)) AS x(

id INT,

student\_id INT,

amount NUMERIC,

disacount NUMERIC,

quantity\_courses INT,

status BOOLEAN,

updated\_at TIMESTAMP WITH TIME ZONE

);

UPDATE enrollments e

SET

student\_id = COALESCE(u.student\_id, e.student\_id),

amount = COALESCE(u.amount, e.amount),

disacount = COALESCE(u.disacount, e.disacount),

quantity\_courses = COALESCE(u.quantity\_courses, e.quantity\_courses),

status = COALESCE(u.status, e.status),

updated\_at = COALESCE(u.updated\_at, CURRENT\_TIMESTAMP)

FROM tmp\_enrollments\_upd u

WHERE e.id = u.id;

RETURN TRUE;

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER COST 1000;

COMMENT ON FUNCTION public.fx\_upd\_enrollments(JSONB) IS

$$

OBJETIVO : Actualizar enrollments (parcial)

SINTAXIS:

SELECT public.fx\_upd\_enrollments('[{"id":1,"amount":180.00}]'::jsonb);

$$;

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- ENROLLMENT\_DETAILS

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DROP FUNCTION IF EXISTS public.fx\_sel\_enrollment\_details(JSONB);

CREATE FUNCTION public.fx\_sel\_enrollment\_details(JSONB)

RETURNS TABLE (

id INT,

enrollment\_id INT,

course\_assigned\_id INT,

course\_name VARCHAR,

teacher\_person\_id INT,

teacher\_dni VARCHAR,

teacher\_names VARCHAR,

score NUMERIC,

status BOOLEAN,

created\_at TIMESTAMP WITH TIME ZONE,

updated\_at TIMESTAMP WITH TIME ZONE

)

AS $$

DECLARE

p\_json\_data ALIAS FOR $1;

BEGIN

RETURN QUERY

WITH filtros AS (

SELECT

x.id,

x.enrollment\_id,

x.course\_assigned\_id,

x.status,

x.created\_at

FROM JSONB\_TO\_RECORDSET(COALESCE(p\_json\_data,'[]'::JSONB)) AS x(

id INT,

enrollment\_id INT,

course\_assigned\_id INT,

status BOOLEAN,

created\_at TIMESTAMP WITH TIME ZONE

)

)

SELECT

ed.id,

ed.enrollment\_id,

ed.course\_assigned\_id,

c.name as course\_name,

te.person\_id as teacher\_person\_id,

p.dni as teacher\_dni,

p.names as teacher\_names,

ed.score,

ed.status,

ed.created\_at,

ed.updated\_at

FROM enrollment\_details ed

LEFT JOIN teachers\_by\_course tbc ON tbc.id = ed.course\_assigned\_id

LEFT JOIN teachers te ON te.id = tbc.teacher\_id

LEFT JOIN people p ON p.id = te.person\_id

LEFT JOIN courses c ON c.id = tbc.course\_id

LEFT JOIN filtros f ON TRUE

WHERE (f.id IS NULL OR ed.id = f.id)

AND (f.enrollment\_id IS NULL OR ed.enrollment\_id = f.enrollment\_id)

AND (f.course\_assigned\_id IS NULL OR ed.course\_assigned\_id = f.course\_assigned\_id)

AND (f.status IS NULL OR ed.status = f.status)

AND (f.created\_at IS NULL OR DATE(ed.created\_at) = DATE(f.created\_at));

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER COST 1000;

COMMENT ON FUNCTION public.fx\_sel\_enrollment\_details(JSONB) IS

$$

OBJETIVO : Consultar enrollment\_details con joins a teachers\_by\_course->teachers->people y courses

SINTAXIS:

SELECT \* FROM public.fx\_sel\_enrollment\_details(NULL);

SELECT \* FROM public.fx\_sel\_enrollment\_details('[{"enrollment\_id":1}]'::jsonb);

$$;

DROP FUNCTION IF EXISTS public.fx\_ins\_enrollment\_details(JSONB);

CREATE FUNCTION public.fx\_ins\_enrollment\_details(JSONB)

RETURNS BOOLEAN

AS $$

DECLARE

p\_json\_data ALIAS FOR $1;

BEGIN

DROP TABLE IF EXISTS tmp\_enrollment\_details;

CREATE TEMPORARY TABLE tmp\_enrollment\_details AS

SELECT

x.enrollment\_id,

x.course\_assigned\_id,

x.score,

x.status,

x.created\_at,

x.updated\_at

FROM JSONB\_TO\_RECORDSET(COALESCE(p\_json\_data,'[]'::JSONB)) AS x(

enrollment\_id INT,

course\_assigned\_id INT,

score NUMERIC,

status BOOLEAN,

created\_at TIMESTAMP WITH TIME ZONE,

updated\_at TIMESTAMP WITH TIME ZONE

);

INSERT INTO enrollment\_details(

enrollment\_id,

course\_assigned\_id,

score,

status,

created\_at,

updated\_at

)

SELECT

enrollment\_id,

course\_assigned\_id,

COALESCE(score, NULL),

COALESCE(status, TRUE),

COALESCE(created\_at, CURRENT\_TIMESTAMP),

updated\_at

FROM tmp\_enrollment\_details;

RETURN TRUE;

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER COST 1000;

COMMENT ON FUNCTION public.fx\_ins\_enrollment\_details(JSONB) IS

$$

OBJETIVO : Insertar registros en enrollment\_details

SINTAXIS:

SELECT public.fx\_ins\_enrollment\_details('[{"enrollment\_id":1,"course\_assigned\_id":2,"score":15.5}]'::jsonb);

$$;

DROP FUNCTION IF EXISTS public.fx\_upd\_enrollment\_details(JSONB);

CREATE FUNCTION public.fx\_upd\_enrollment\_details(JSONB)

RETURNS BOOLEAN

AS $$

DECLARE

p\_json\_data ALIAS FOR $1;

BEGIN

DROP TABLE IF EXISTS tmp\_enrollment\_details\_upd;

CREATE TEMPORARY TABLE tmp\_enrollment\_details\_upd AS

SELECT

x.id,

x.enrollment\_id,

x.course\_assigned\_id,

x.score,

x.status,

x.updated\_at

FROM JSONB\_TO\_RECORDSET(COALESCE(p\_json\_data,'[]'::JSONB)) AS x(

id INT,

enrollment\_id INT,

course\_assigned\_id INT,

score NUMERIC,

status BOOLEAN,

updated\_at TIMESTAMP WITH TIME ZONE

);

UPDATE enrollment\_details ed

SET

enrollment\_id = COALESCE(u.enrollment\_id, ed.enrollment\_id),

course\_assigned\_id = COALESCE(u.course\_assigned\_id, ed.course\_assigned\_id),

score = COALESCE(u.score, ed.score),

status = COALESCE(u.status, ed.status),

updated\_at = COALESCE(u.updated\_at, CURRENT\_TIMESTAMP)

FROM tmp\_enrollment\_details\_upd u

WHERE ed.id = u.id;

RETURN TRUE;

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER COST 1000;

COMMENT ON FUNCTION public.fx\_upd\_enrollment\_details(JSONB) IS

$$

OBJETIVO : Actualizar enrollment\_details (parcial)

SINTAXIS:

SELECT public.fx\_upd\_enrollment\_details('[{"id":1,"score":18}]'::jsonb);

$$;

* **Crear un usuario con permisos limitados (ejemplo: secretaria solo inserta y consulta alumnos).**

-- Creacion de Usuarios

CREATE USER secretary WITH PASSWORD '123456';

GRANT SELECT ON TABLE students TO secretary;

GRANT INSERT ON TABLE students TO secretary;