

BASES DE DATOS

PROYECTO PROFESIONALIZANTE

GESTIÓN DE BASE DE DATOS UNIVERSITARIA

Carrera

**Tecnicatura Universitaria en Programación
Informática**

Alumnos

**Decuzzi Emiliano
Fucceneco Valentin Pedro
Rey Brienza Agustina
Sabbatini Tatiana
Tarquini Gabriel**

Grupo

5 - McTeam

2024

Parte 1: Análisis y Limpieza de datos.....	3
1. Decisiones en base al modelo de negocio.....	3
2. Paso a Paso.....	5
3. Conclusiones y versión final.....	8
Parte 2: Selección y Justificación del Motor de Bases de Datos.....	10
1. Investigación y Selección de Motor Relacional.....	10
2. Evaluación Comparativa con una Base de Datos No Relacional.....	11
Parte 3: Diseño de Modelo de Datos y Esquema.....	13
1. Diseño del Modelo Entidad-Relación.....	13
2. Implementación del Esquema SQL.....	14
Parte 4: Carga y Manipulación de Datos.....	18
1. Carga inicial de los datos.....	18
2. Consultas Básicas y Reportes.....	21
3. Consultas Avanzadas y Optimización.....	25
Parte 5: Procedimientos, Triggers y Automatización.....	30
1. Análisis Teórico de Procedimientos Almacenados.....	30
2. Análisis Teórico de Triggers.....	32
3. Propuestas de Estrategias de Automatización.....	41
Parte 6: Anexo.....	45
Bibliografía.....	46

Parte 1: Análisis y Limpieza de datos

1. Decisiones en base al modelo de negocio

Se decidió realizar una limpieza manual de los datos, estableciendo la entidad **Alumno** como la principal. En primer lugar, se revisaron los datos en general, eliminando las entradas correspondientes a los profesores y una entrada duplicada de un alumno, con el objetivo de obtener una tabla con datos únicos que representen las reglas de negocio.

En los campos vacíos se optó por dos medidas. En Hobbie y Trabajo/Actividad los datos vacíos se dejaron en **null**, ya que estos correspondían a datos con **parcialidad** según el **diagrama de entidad-relación** planteado. Por ejemplo, un alumno que no trabaja o no tiene hobbies.

id_alumno	dni	nombre_alumno	grupo_fk	trabajo_fk
11	43245286	Valentino	3	

En cuanto al dato del lugar de procedencia, se unificaron criterios para referirse a localidades. En el caso de barrios de Capital Federal, se especificó como **C.A.B.A.**. Para los casos en que este dato no existía, se asignó un valor dummy: **'No especifica'**.

id_localidad	nombre_localidad
1	San Martin
2	Tigre
3	CABA
4	Tres de febrero
5	Escobar
6	Morón
7	La Matanza
8	No especifica

Respecto al campo de trabajo, se desdobló la información en dos entidades: **Rubro** y **Actividad**. Esta decisión permitió un mejor agrupamiento de la información que será útil en las consultas planteadas en puntos posteriores.

Rubro	Actividad
IT	Desarrollador de Software
null	null
CONTABILIDAD	Finanzas
IT	Desarrollador de Software
IT	Desarrollador de Software
IT	Ingeniero de datos
IT	Desarrollador de Software
ADMINISTRATIVO	Seguros
IT	Ingeniero de datos
IT	Desarrollador de Software
null	null
null	null
IT	Desarrollador de Software
IT	Soporte
IT	Desarrollador de Software
IT	Desarrollador de Software

Los datos relacionados con **Mascotas** y **Experiencia en Bases de Datos** no se modelaron como entidades, sino que se incluyeron como atributos del alumno, registrando estos valores como booleanos.

Experiencia BBDD Relacional	Experiencia BBDD No Relacional	Mascotas
TRUE	FALSE	FALSE
FALSE	FALSE	TRUE
FALSE	FALSE	TRUE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE
TRUE	TRUE	FALSE
TRUE	TRUE	FALSE
TRUE	TRUE	FALSE

El caso de la **Música** fue el que más controversias trajo. Lo primero que se hizo fue agrupar las bandas según el género musical. Sin embargo, se cuestionó el valor real de este dato por lo cual se decidió incluirla dentro del hobby. De esta manera se evitó generar una nueva relación que no aportaría relevancia significativa en el contexto del objetivo planteado, pero se mantuvo la información aportada en cada registro.

Como este punto sobre la música y el hobby generó dudas se decidió hacer un backup en el propio excel ya que en el caso de modificar el modelo de negocio se cambiaría el planteo y existiría la posibilidad de hacer **rollback**:

	K	L	M	N	O	P
1	Experiencia BDD Relacional	Experiencia BDD No Relacional	Localidad	Hobbie	Mascotas	Musica Genero
2	TRUE	FALSE	San Martín	Musica, viajar y videojuego	FALSE	Rock Nacional
3	FALSE	FALSE	null	null	TRUE	null
4	FALSE	FALSE	San Martín	Robotica	TRUE	Rock Internacional
5	FALSE	TRUE	null	Deporte	TRUE	Ritmos tropicales
6	TRUE	FALSE	San Martín	Lectura	TRUE	Todo
7	TRUE	TRUE	Tigre	Musica, deporte	TRUE	Trap
8	TRUE	TRUE	CABA	Lectura, Dibujo, Deporte	FALSE	Rock Internacional
9	TRUE	TRUE	null	null	FALSE	Rock Internacional
10	TRUE	TRUE	San Martín	Deporte y películas	FALSE	null
11	TRUE	TRUE	Tres de febrero	Deporte y musica	TRUE	Cuarteto
12	FALSE	FALSE	Escobar	Deporte	TRUE	Rock Internacional
13	FALSE	FALSE	Escobar	Lectura	TRUE	null
14	TRUE	TRUE	San Martín	Deporte, lectura y jardineria	TRUE	null
15	TRUE	TRUE	CABA	Deporte, lectura y electrónica	TRUE	Todo
16	TRUE	FALSE	CABA	Series, películas	FALSE	Pop, KPop, Rock Internacional
17	TRUE	FALSE	Villa Ballester	Deporte	TRUE	null
18	FALSE	FALSE	null	Videojuego	TRUE	null
19	TRUE	FALSE	CABA	Series, películas	FALSE	Musicales, Rock
20	TRUE	FALSE	Tres de febrero	null	TRUE	Rock Nacional
21	TRUE	FALSE	San Martín	Jardineria, redes sociales	TRUE	Rock Nacional

Finalmente, la columna **RoI** se optó por eliminarla ya que no aportaba información relevante en el contexto de una base de datos para la Universidad (reglas de negocio). Se entiende que, si el objetivo era trabajar en una base de datos sobre los alumnos históricos que cursaron la materia bases de datos, este dato iba a cobrar más relevancia.

2. Paso a Paso

En la **primera limpieza** de los datos solamente separamos en columnas, como se puede ver en la siguiente captura:

	A	B	C	D
1		Apellido	Nombre	Email
2	37688075	Jotallan Calvetti	Gabriel	gaabicarp@gmail.com
3	41472398	Nadine Kovinchich	Mariel	marielkov1998@gmail.com
4	38532025	Abelardo	Gastón Ezequiel	gabelardo@estudiantes.unsam.edu.ar
5	40663606	Pavon Gomez	Rodrigo Nicolas	rodrigopavongomez@gmail.com
6	38601662	Dragonetti	Maria Cecilia	mdragonetti@estudiantes.unsam.edu.ar
7	44547586	Caceffo	Juan Ignacio	jicaceffo@estudiantes.unsam.edu.ar
8	37984582	Gibelli	Julian	juligibelli@gmail.com
9	35972409	Nuñez	Emiliano Javier	ejnunez@estudiantes.unsam.edu.ar
10	43036494	Menini	Alejo	amenini@estudiantes.unsam.edu.ar
11	35726203	Barneche	Facundo	fh.barneche@gmail.com
12	43245286	Bortolussi	Valentino	valentino.bortolussi.lembo@gmail.com
13	42647332	Bouza	Santiago	sbouza@estudiantes.unsam.edu.ar
14	38703368	Lomas	Cristian	cristian.lomas.a@gmail.com
15	42472348	Neiro	Tomás	tomasneiro@hotmail.com
16	37626822	Villafañez Sobrecasa	Cristian	ccvillafanezsobrecasa@estudiantes.unsam.edu.ar
17	41067566	Borrelli	Maximiliano	maxifborrelli@gmail.com
18	44792981	Narmontas Bocci	Theo	narmontastheo@gmail.com
19	95822280	Toledo Contreras	Paola	ptoledocontreras@estudiantes.unsam.edu.ar
20	35993466	Diaz	Matias Hernan	diaz.matiash@gmail.com
21	42321002	Pagano	Annabella	apagano@estudiantes.unsam.edu.ar
22	40007189	Tarquini	Gabriel	gabi.tarquini@gmail.com
23	39800551	Fucceneco	Valentin Pedro	vfucceneco@estudiantes.unsam.edu.ar
24	41106994	Rey Brienza	Agustina	a.reybrienza@gmail.com
25	43781315	Decuzzi	Emiliano	eadeceuzzi@estudiantes.unsam.edu.ar
26	44160355	Sabbatini	Tatiana	tsabbatini@estudiantes.unsam.edu.ar
27	40395042	Ruina	Mariano	mjruiua@estudiantes.unsam.edu.ar
28	41684308	Cuellar	Lautaro	lautacuellar69@hotmail.com
29	43017353	Virgilio	Federico	fedevirgilio0@gmail.com
30	42291365	Exarchos	Alan	alanexarchos@gmail.com
31	45105469	Rossi	Florencia	frossi@estudiantes.unsam.edu.ar
32		Davogustto	Ernesto	ernesto.davogustto@gmail.com
33	45174406	Rodriguez	Lucas Gonzalo	lgrdriguez@estudiantes.unsam.edu.ar
34	39916775	Guarino	Alan	aguarino@estudiantes.unsam.edu.ar

E	G	J	K	L
GRUPO_ID	ROL_ID	Materias cursadas	Trabajo	Experiencia BBDD - NO Relacional - relacional (b
1	2	Solo base de datos.	Trabajo de frontend en una compañía de seguros	Tengo experiencia con sqlServer y mongodb
1	5			
1	1	Curso también algoritmos 3	Trabajo en una oficina de costos	No trabaje con base de datos
1	3	Curso Seminario y Bases de Datos	Trabajo de Desarrollador front-end freelance y soy	tengo conocimientos mínimos de Bases de Datos
1	4	Estoy cursando seminario	soy desarrolladora frontend	la experiencia en base de datos es la que tuve en
2	3	Base de datos	trabajo como ingeniero de datos en NTT Data	experiencia previa en MySQL, posgres, graphQL, r
2	2	bd y seminario	Trabajo como desarrollador de software	Experiencia en sql con mysql y nosql con firebase
2	5	Cursando Seminario	trabajo en empresa de seguros	uso MS-SQL
2	4	Cursando Base de datos	trabajo de ingeniero de datos	Tengo experiencia con SQL server
2	1	Base de datos, Seminario de programaci	Trabajo como desarrollador	No mucha pero experiencia en mysql y mongodb
3	4	Curso algoritmos 3, base de datos y sem	No trabajo	No tengo experiencia en base de datos
3	3	Seminario, Algoritmos 3 y Bases de dato	No trabajo	No experiencia previa
3	5	Cursando Seminario, Algo3 y BDD	Trabajo como desarrollador	tengo algo de experiencia en bases de datos rela
3	2	Curso Algo 3, Seminario y dBase de Dat	Trabajo nada fijo relacionado a mantenimiento de	experiencia en base de datos en el secundario
3	1	Cursando Base de Datos	Trabajo como desarrolladora en .NET para Accent	Tengo experiencia con base de datos, utilizando
4	3	Curso algoritmos 3 y bases de datos	soy desarrollador backend java/golang	tengo experiencia en dynamodb y sql server
4	2	las 3 materias del cuatrimestre	no estoy trabajando actualmente	
4	5	Cursando Base de Datos	Trabajo como desarrolladora en .NET para Accent	Tengo experiencia con base de datos, utilizando
4	4	Base de datos, Algoritmo 3 y Seminario	Trabajo como desarrollador	Experiencia poca en sql
4	1	Curso CASO y base de datos	Trabajo en ciberseguridad	uso SQL Developer para algunos proyectos
5	3	Cursando algoritmos 3 y bases de datos	Trabajo como empleado de comercio	No tengo experiencia en bbdd
5	2	Cursando Algoritmos 3	trabajo como administrativo en una papeleria	sin experiencia en base de datos
5	5	Curso algoritmos 3 y BBDD	Actualmente estoy trabajando como desarrollador	No tengo experiencia en BBDD
5	4	Curso algoritmos 3, base de datos y sem	Trabajo en el area de informatica de un hospital	Experiencia con MS Access
5	1	Cursando las 3 del pack	Trabajando en soporte como pasante	Sin experiencia en base de datos
6	3	Cursando Seminario, Algo 3 y Base De datos		Tengo experiencia en base de datos
6	2	Curso base de datos, algoritmos 3 y sem	Trabajo de repartidor de agua	no tengo experiencia en base de datos
6	5	Cursando Algoritmos 3, bases de datos, Trabajando con IA en pasantia		Poca experiencia con DB solo proyecto propio
6	4	Estoy cursando seminario y base de datos		Actualmente estoy ayudando a un amigo en un bi
6	1	Cursando seminario, algo 3 y base de d	Trabajando con IA	Cero experiencia en base de datos
7	3	Bases de datos, Algoritmos 3 y Seminario	trabajo como ingeniero de Datos en una Consultor	Experiencia de 1 año
7	5	Cursando el pack de 3	Actualmente no estoy trabajando	No tengo experiencia en el mundo de base de da
7	4	Cursando Algoritmos 3, seminario y base	Trabajo en comercio electronico	Experiencia en bases de datos muy poca

Trabajo	Experiencia BBDD - NO Relacional - relacional (boolean)	Localidad
Trabajo de frontend en una compañía de seguros	Tengo experiencia con sqlServer y mongodb	San Martin
Trabajo en una oficina de costos	No trabaje con base de datos	San Martin
Trabajo de Desarrollador front-end freelance y soy	tengo conocimientos minimos de Bases de Datos	
soy desarrolladora frontend	la experiencia en base de datos es la que tuve en phm el	San Martin
trabajo como ingeniero de datos en NTT Data	experiencia previa en MySQL, posgres, graphQL, mongoDE	General Pacheco
Trabajo como desarrollador de software	Experiencia en sql con mysql y nosql con firebase	Soy de Belgrano
trabajo en empresa de seguros	uso MS-SQL	
trabajo de ingeniero de datos	Tengo experiencia con SQL server	Soy de San Andres
Trabajo como desarrollador	No mucha pero experiencia en mysql y mongodb	Santos Lugares
No trabajo	No tengo experiencia en base de datos	Soy de Escobar
No trabajo	No experiencia previa	Escobar
Trabajo como desarrollador	tengo algo de experiencia en bases de datos relacionales	N/A
Trabajo nada fijo relacionado a mantenimiento de	experiencia en base de datos en el secundario	Soy de Saavedra
Trabajo como desarrolladora en .NET para Accent	Tengo experiencia con base de datos, utilizando SQL Serv	Vivo en Boedo, CAI
soy desarrollador backend java/golang	tengo experiencia en dynamodb y sql server	Ballester
no estoy trabajando actualmente		
Trabajo como desarrolladora en .NET para Accent	Tengo experiencia con base de datos, utilizando SQL Serv	Vivo en Boedo, CAI
Trabajo como desarrollador	Experiencia poca en sql	Soy de Santos Lug
Trabajo en ciberseguridad	uso SQL Developer para algunos proyectos	San martin
Trabajo como empleado de comercio	No tengo experiencia en bbdd	Soy de Villa Bosch,
trabajo como administrativo en una papeleria	sin experiencia en base de datos	san martin
Actualmente estoy trabajando como desarrollador	No tengo experiencia en BBDD	El Palomar
Trabajo en el area de informatica de un hospital	Experiencia con MS Access	Vivo en Ciudad Jan
Trabajando en soporte como pasante	Sin experiencia en base de datos	Soy de Villa Urquizi
datos	Tengo experiencia en base de datos	
Trabajo de repartidor de agua	no tengo experiencia en base de datos	Vivo en Loma herm
Trabajando con IA en pasantia	Poca experiencia con DB solo proyecto propio	San Martin.
os	Actualmente estoy ayudando a un amigo en un backend e	villa bosch
Trabajando con IA	Cero experiencia en base de datos	Vivo en CABA
trabajo como ingeniero de Datos en una Consultor	Experiencia de 1 año	Vivo en Palermo

Localidad	Hobbie	Mascotas	Musica Genero
San Martin	Toco la guitarra, me gusta viajar y los videojuegos	No tengo mascotas	Rock nacional
San Martin	Soy técnico electrónico me gusta todo lo que es reparación de pc Arduino etc	Tengo una perrita se llama leia,	the Beatles.
San Martin	Mi hobbie es jugar y entrenar voley, estoy en el mixto de la Unsam	Tengo 3 gatos (Artyom, Sol y Tig	Me gusta los ritmos tropicales
San Martin	me gusta mucho leer	Tengo un perro Milo y un gatito f	de todo un poco
General Pacheco	Cantar, correr, gym	bobi, luis y mailo	Wos
Soy de Belgrano	leer, dibujar y correr. Serie: supernatural		arctic monkeys, radiohead
Soy de San Andres	Natacion y salir al cine. the boys y the bear	no tengo mascota	me gusta bandas como foofighter
Santos Lugares	Jugar al futbol y escuchar musica	No tengo	
Soy de Escobar	Mi hobbie es entrenar	Una gatita de mascota de 10 añ	Recomiendo cuarteto pero me gu
Escobar	Leer ficción e historia	Tengo una perra, Rita	Recomiendo Queen.
N/A	Deportes variados, ajedrez, leer novelas y dedicarle tiempo a mi huerta	3 gatos: Apolo, Coco y Zoe	
Soy de Saavedra	jugar al tenis, leer mangas, armar PC's	vivo con 3 gatos	
Vivo en Boedo, CA	ver series y películas	Tengo dos perritas.	top 50 spoty.
Ballester	Juego a la pelota	No tengo mascotas	Escucho todo lo que sea Pop, incl
	me gustan los videojuegos, el anime, manga y me ejercito, elden ring, dragon ball y jojo's mis favoritos	tengo una perra	
Vivo en Boedo, CA	ver series y películas	No tengo mascotas	Escucho todo lo que sea Pop, incl
Soy de Santos Lug	No tengo mucho tiempo para hobbies	Tengo una perra.	rock nacional.
San martin	jardineria o twitter.	Dos gatos y un perro	Musica los piojos
Soy de Villa Bosch,	juntarme con amigos a comer algo o jugar algun futbol		
san martin		tengo 2 gatos	Musica rock nacional
El Palomar	Me gustan mucho los deportes, en especial el basquet que practico desde que	Tengo dos perros Uma y Jack	Música escucho sobretodo rock n
Vivo en Ciudad Jan	Gimnasio, bici, juegos	Una perra mitad caniche mitad c	Musica: todos los generos, The hi
Soy de Villa Urquiza	Juego al voley, leo fantasía	No tengo	Musicales, pop, rock
	Tambien soy estudiante de la Lic. en Psicología en UBA. Hago musica y mi instrumento de preferencia es la guitarra.		
Vivo en Loma herm	futbol tanto practicarlo como verlo	tengo una gata que se llama lety	rock nacional
San Martin.			
a villa bosch	peliculas,series y juegos	Tengo un perro mestizo Tyson	roto entre metal, electronica y tra
Vivo en CABA		Mascotas tengo un perro que se	Miranda
Vivo en Palermo	Fotografía, entrenar y musica.	Tengo un perro.	

Luego, fue tomando forma cuando empezamos a estandarizar los valores y eliminar duplicados de cada campo.

	A	B	C	D	E	F	G	H	I
1		Apellido	Nombre	Email	GRUPO_ID	ROL_ID	Materias cursadas	Rubro	Actividad
2	37688975	Jotatan Calvetti	Gabriel	gaabircarp@gmail.com	1	2	BBDD	CONTABILIDAD	Desarrollador de Software
3	41472398	Nadine Kovinchich	Mariel	mariekov1998@gmail.com	1	5	BBDD		
4	38532025	Abelardo	Gastón Ezequiel	gabelardo@estudiantes.unsam.edu.ar	1	1	Algo 3 y BBDD		Finanzas
5	40663606	Pavon Gomez	Rodrigo Nicolas	rodrigopavongomez@gmail.com	1	3	Seminario y BBDD		Desarrollador de Software
6	38601662	Dragonetti	Maria Cecilia	mdragonetti@estudiantes.unsam.edu.ar	1	4	Seminario y BBDD		Desarrollador de Software
7	44547586	Caceffo	Juan Ignacio	jicaceffo@estudiantes.unsam.edu.ar	2	3	BBDD		Ingeniero de datos
8	37984582	Gibelli	Julían	juligibelli@gmail.com	2	2	Seminario y BBDD		Desarrollador de Software
9	35972409	Núñez	Emiliano Javier	ejunez@estudiantes.unsam.edu.ar	2	5	Seminario y BBDD	ADMINISTRATIVO	Seguros
10	43036494	Menini	Alejo	amenini@estudiantes.unsam.edu.ar	2	4	BBDD		Ingeniero de datos
11	35726203	Barneche	Facundo	fb.barneche@gmail.com	2	1	Seminario y BBDD		Desarrollador de Software
12	43245286	Bortolussi	Vaentino	vaentino.bortolussi.lemba@gmail.com	3	4	Algo 3, Seminario y BBDD		
13	42647332	Bouza	Sanitago	sbouza@estudiantes.unsam.edu.ar	3	3	Algo 3, Seminario y BBDD		
14	38703368	Lomas	Cristian	cristian.lomas.a@gmail.com	3	5	Algo 3, Seminario y BBDD		Desarrollador de Software
15	42472348	Nero	Tomás	tomasonero@hotmail.com	3	2	Algo 3, Seminario y BBDD		Soporte
16	37626822	Villafañez Sobrecasa	Cristian	ccvillafanezsobrecasa@estudiantes.unsam.edu.ar	3	1	BBDD		Desarrollador de Software
17	41067566	Borrelli	Maximiliano	maxiborrelli@gmail.com	4	3	Algo 3 y BBDD		Desarrollador de Software
18	44792981	Narmontas Bocci	Theo	narmontastheo@gmail.com	4	2	Algo 3, Seminario y BBDD		
19	95822280	Toledo Contreras	Paola	ptoledocontreras@estudiantes.unsam.edu.ar	4	5	BBDD		Desarrollador de Software
20	35993466	Diaz	Matias Hernan	diaz.matias@gmail.com	4	4	Algo 3, Seminario y BBDD		Desarrollador de Software
21	42321002	Pagano	Annabella	apagano@estudiantes.unsam.edu.ar	4	1	CASO y BBDD		Ciberseguridad
22	40007189	Tarquini	Gabriel	gabti.tarquini@gmail.com	5	3	BBDD	COMERCIO	
23	39800551	Fucceneco	Valentin Pedro	vfucceneco@estudiantes.unsam.edu.ar	5	2	Algo 3 y BBDD	ADMINISTRATIVO	
24	41106994	Rey Brienza	Agustina	a.reybrienza@gmail.com	5	5	Algo 3 y BBDD		Desarrollador de Software
25	43781315	Decuzzi	Emiliano	eadecuzzi@estudiantes.unsam.edu.ar	5	4	Algo 3, Seminario y BBDD		Soporte
26	44160355	Sabbatini	Tatiana	tsabbatini@estudiantes.unsam.edu.ar	5	1	Algo 3, Seminario y BBDD		Soporte
27	40395042	Ruina	Mariano	mruiuna@estudiantes.unsam.edu.ar	6	3	Algo 3, Seminario y BBDD		
28	41894308	Cuellar	Lautaro	lautacuellar19@hotmail.com	6	2	Algo 3, Seminario y BBDD	COMERCIO	Envios
29	43017353	Virgilio	Federico	fedevirgilio@gmail.com	6	5	Algo 3, Seminario y BBDD		IA
30	42291365	Evachos	Alan	alanevachos@gmail.com	6	4	Seminario y BBDD		
31	45105469	Rossi	Florencia	frossi@estudiantes.unsam.edu.ar	6	1	Algo 3, Seminario y BBDD		IA
32	12345678	Davogusto	Ernesto	ernesto.davogustito@gmail.com	7	3	Algo 3, Seminario y BBDD		Ingeniero de datos
33	45174406	Rodriguez	Lucas Gonzalo	lgrodriguez@estudiantes.unsam.edu.ar	7	5	Algo 3, Seminario y BBDD		
34	39916775	Guarino	Alan	aguarino@estudiantes.unsam.edu.ar	7	4	Algo 3, Seminario y BBDD	COMERCIO	Ventas
35	35768192	Mecozzi	Tamara Eleonor	mecozzi@gmail.com	7	2	Algo 3, Seminario y BBDD		
36	43253567	Hoj	Agustín	ahoj@estudiantes.unsam.edu.ar	7	3	Algo 3 y BBDD		Finanzas
37	35093145	Caballero	Matias	msebaccaballero@gmail.com	8	3	BBDD	AUTOMOTRIZ	Mecánico
38	36594617	Pazos	David	davidgpaos@gmail.com	8	2	CASO y BBDD		Ciberseguridad
39	43441575	Signorello	Fabrizio	fsignorello@estudiantes.unsam.edu.ar	8	5	Algo 3 y BBDD		Desarrollador de Software

J	K	L	M	N	O
Experiencia BBDD Relacional	Experiencia BBDD No Relacional	Localidad	Hobbie	Mascotas	Musica Genero
TRUE	FALSE	San Martín	Musica, viajar y videojuego	FALSE	Rock Nacional
FALSE	FALSE	null	null	TRUE	null
FALSE	FALSE	San Martín	Robotica	TRUE	Rock Internacional
FALSE	TRUE	null	Deporte	TRUE	Ritmos tropicales
TRUE	FALSE	San Martín	Lectura	TRUE	Todo
TRUE	TRUE	Tigre	Musica, deporte	TRUE	Trap
TRUE	TRUE	CABA	Lectura, Dibujo, Deporte	FALSE	Rock Internacional
TRUE	TRUE	null	null	FALSE	Rock Internacional
TRUE	TRUE	San Martín	Deporte y películas	FALSE	null
TRUE	TRUE	Tres de f...	Deporte y musica	TRUE	Cuarteto
FALSE	FALSE	Escobar	Deporte	TRUE	Rock Internacional
FALSE	FALSE	Escobar	Lectura	TRUE	null
TRUE	TRUE	San Martín	Deporte, lectura y jardineria	TRUE	null
TRUE	TRUE	CABA	Deporte, lectura y electrónica	TRUE	Todo
TRUE	FALSE	CABA	Series, películas	FALSE	Pop, KPop, Rock Internacional
TRUE	FALSE	Villa Bail...	Deporte	TRUE	null
FALSE	FALSE	null	Videojuego	TRUE	null
TRUE	FALSE	CABA	Series, películas	FALSE	Musicales, Rock
TRUE	FALSE	Tres de f...	null	TRUE	Rock Nacional
TRUE	FALSE	San Martín	Jardineria, redes sociales	TRUE	Rock Nacional
FALSE	FALSE	Villa Bosch	Deporte, reunion amigos	TRUE	Rock Nacional
FALSE	FALSE	San Martín	null	TRUE	Rock Nacional
FALSE	FALSE	Morón	Deporte, musica	TRUE	Rock Nacional
TRUE	FALSE	Tres de f...	Deporte, videojuego	TRUE	Todo
FALSE	FALSE	CABA	Deporte, lectura	FALSE	Musicales, Pop, Rock
TRUE	TRUE	null	Musica	FALSE	null
FALSE	FALSE	Tres de f...	Deporte	TRUE	Rock Nacional
TRUE	FALSE	San Martín	null	FALSE	null
TRUE	FALSE	Tres de f...	Series, películas, videojuego	TRUE	Metal, Electronica, Trap argentino.
FALSE	FALSE	CABA	null	TRUE	Pop
TRUE	TRUE	CABA	Fotografia, entrenar y musica.	TRUE	null
FALSE	FALSE	Villa Bail...	Series	TRUE	Rock Nacional
TRUE	FALSE	Villa Bail...	Deporte	TRUE	Rock Internacional
FALSE	FALSE	San Martín	Viajar, pelicula, deporte	TRUE	Todo
TRUE	FALSE	Villa Bail...	Videojuego, deporte	TRUE	Rock, Electronica, Rap.
TRUE	TRUE	La Matanza	Musica	TRUE	Rap
FALSE	FALSE	CABA	Deporte	FALSE	Rock Nacional, Rock Internacional, Pop
TRUE	FALSE	Villa Bail...	Deporte	TRUE	Rock Nacional, Electrónica

Una vez separadas las entidades en columnas se crearon tablas base e intermedias para aquellas que lo requerían, como **Alumno_Cursa_Materia** y **Alumno_Tiene_Hobbie**. Esta parte fue elaborada a la par que se diseñaba el DER para entender mejor cómo serían las tablas y las relaciones entre las entidades.

3. Conclusiones y versión final

En la versión final, tenemos tablas normalizadas en la primera, segunda y tercera forma normal.

Tenemos las siguientes tablas base:


- Alumno
- Localidad
- Materia
- Grupo
- Trabajo
- Rubro
- Hobbie

Luego, las tablas intermedias:

- Alumno_Cursa_Materia
- Alumno_Tiene_Hobbie

Se puede ver que en los campos donde antes teníamos 'null', ahora están vacíos. Esto se debe a que cuando luego creamos las tablas e insertamos los valores en la base de datos del punto 4, **PostgreSQL** no entendía la diferencia entre el valor null y el string 'null'. Dejándolos vacíos, el programa se encargó de llenarlos.

Se puede observar la versión final en el siguiente link:

 [TP MC TEAM Alumnas BdD 2024](#)

Parte 2: Selección y Justificación del Motor de Bases de Datos

1. Investigación y Selección de Motor Relacional

Según el modelo de negocios presentado, se requiere un sistema apuntado a la organización de grandes cantidades de información estructurada mediante tablas de manera relacional. Al ser una universidad también se espera que el sistema pueda ser **fácilmente escalable** de manera vertical y a raíz que se mantenga la **integridad, consistencia** y relaciones entre los datos. Para asegurar este último punto se puede optar por un sistema de gestión compatible con **transacciones ACID** preparado para una media-alta concurrencia ya que hay muchos datos modificables.

Teniendo en cuenta las consideraciones ya mencionadas, optamos por el motor **PostgreSQL**. Este además de cubrir las necesidades del proyecto, cuenta con soporte para almacenar datos en formato **JSON** y **JSONB**, si es que es necesario manejar datos semi o no estructurados. También cuenta con soporte especializado en consultas complejas y buena capacidad de indexación, esto en su conjunto permite gestionar operaciones de análisis intensivo.

En cuanto a la concurrencia, se maneja con **MVCC** (Multi-Version Concurrency Control), que permite el acceso a múltiples usuarios y que estos hagan modificaciones de los datos sin que se bloqueen mutuamente. Otra ventaja de PostgreSQL es que, es un sistema gratuito, de código abierto, el cual recibe frecuentes actualizaciones, cuenta con muchos recursos y variada documentación.

Sin embargo, hablando de desventajas, podemos marcar que PostgreSQL necesita de una infraestructura robusta si gestiona aplicaciones de gran volumen de datos. También al ser altamente configurable en términos de rendimiento y administración de memoria, lo cual sin dudas es una ventaja, requiere de personal técnicamente más especializado para su mantenimiento y mejoras de rendimiento. Otra limitación que podemos mencionar es la falta de soporte para una escalabilidad horizontal, ya que habría que depender del uso de herramientas externas. Aunque este motor sí soporta replicación y particionamiento si fuese necesario. De todos modos, según el modelo de negocio presentado, este punto no cuenta con mucho peso ya que se piensa en escalabilidad vertical.

2. Evaluación Comparativa con una Base de Datos No Relacional

En el caso de optar por una base de datos no relacional, el sistema debería estructurarse en colecciones de documentos y no en tablas relacionadas como sería en el caso de PostgreSQL. Por ejemplo, un documento tendría toda la información de un alumno en un solo registro (materias, notas, información personal, intereses, imágenes, etc.).

Esta implementación facilita tanto el agregado de información personalizada de cada estudiante o materia, como también la actualización de dicha información, ya que no sería necesario actualizar una secuencia de tablas y registros relacionados. Podemos decir entonces que este modelo brinda una gran flexibilidad en el diseño manejando datos no estructurados y también en el almacenamiento personalizado.

Otro beneficio de una base de datos no relacional es la posibilidad de un escalado horizontal, ya que permite replicar y distribuir los datos en distintos nodos, haciendo que la información sea más accesible desde cualquier punto y reduciendo la carga del sistema ya que esta se encuentra distribuida. Aplicar un enfoque con estas características, en el contexto de un sistema de gestión universitario, permitiría la fácil expansión del mismo.

Como ejemplo podemos optar por el uso de **MongoDB**, un motor de gestión de bases de datos no relacional. Este sistema almacena información en formato **BSON** (Binary JSON) y cumple con las características ya mencionadas de un sistema no relacional.

En cuanto a su escalabilidad horizontal específica, la realiza mediante **sharding**, que es un método para distribuir datos entre varias máquinas. *"MongoDB utiliza la fragmentación para respaldar implementaciones con conjuntos de datos muy grandes y operaciones de alto rendimiento."*¹

La velocidad de consultas también se ve beneficiada usando un sistema como MongoDB. Este al manejar datos semi o no estructurados, hace que las consultas específicas se realicen de manera más rápida, más aún cuando se trata de consultas complejas en grandes volúmenes.

Por otra parte, el costo de infraestructura también disminuye, ya que si se quiere expandir el sistema solo bastaría con agregar servidores que repliquen la información. De esta manera no habría que gastar tanto para actualizar la tecnología de alta gama que requiere un sistema más centralizado de escalado vertical.

1

<https://www.mongodb.com/docs/manual/sharding/#:~:text=Sharding%20is%20a%20method%20for,capacity%20of%20a%20single%20server.>

En cuanto a sus limitaciones, se pierde la **integridad referencial**. Es decir, que el sistema no sería capaz de mantener relaciones estructuradas entre entidades (claves foráneas), punto que es importante ya que en el modelo de negocios presentado se busca relacionar a alumnos con materias. Al mismo tiempo si la base de datos escala y se agregan nuevas entradas como calificaciones de parciales o trabajos este punto se profundizará aún más.

La **consistencia eventual** de MongoDB también es un punto débil tal vez no conveniente para el modelo de negocios. Si bien es adecuada en sistemas con alta concurrencia y necesidad de disponibilidad, también representa un inconveniente al momento de realizar transacciones críticas simultáneas, como sería la actualización de registros de calificaciones. Se pierde integridad de los datos lo cual puede llevar a errores.

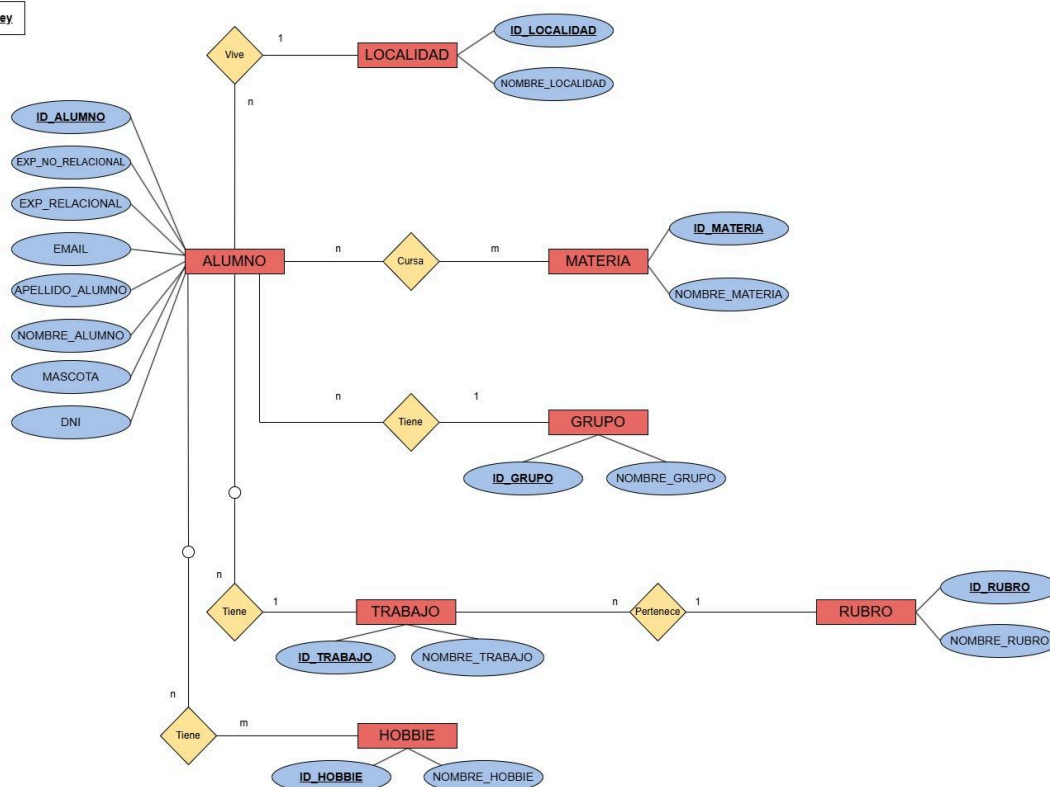
En conclusión, a pesar de los beneficios que otorgaría implementar MongoDB si se piensa en la expansión del sistema a futuro, en la implementación de entidades no estructuradas y, teniendo en cuenta sus puntos fuertes (escalabilidad horizontal, infraestructura barata, registros personalizados, etc.), sigue siendo más beneficioso optar por un sistema relacional como sería el de PostgreSQL, el cual se adapta más al modelo de negocios planteado. Con una base de datos relacional nos aseguramos que las transacciones se realicen de manera segura especialmente porque se implementa ACID, existen relaciones estrictas, no como en MongoDB, y se mantiene la veracidad de los registros una vez que son actualizados ya que PostgreSQL cuenta con consistencia alta en vez de consistencia eventual.

Parte 3: Diseño de Modelo de Datos y Esquema

1. Diseño del Modelo Entidad-Relación

MCTEAM - 5

PK = Primary Key



Ver el DER con mejor calidad : [MCTeam-TP SQL.jpg](#)

MLR (Modelo Lógico Relacional) - PK / FK

Alumno(ID_ALUMNO, DNI, Nombre_Alumno, Apellido_Alumno, Email, Exp_No_Relacional, Exp_Relacional, Mascota, LOCALIDAD_ID, GRUPO_ID, TRABAJO_ID)

Localidad(ID_LOCALIDAD, Nombre_Localidad)

Materia(ID_MATERIA, Nombre_Materia)

Grupo (ID_GRUPO, Nombre_Grupo)

Trabajo(ID_TRABAJO, Nombre_Trabajo, RUBRO_ID)

Rubro(ID_Rubro, Nombre_Rubro)

Hobbie(ID_HOBBIE, Nombre_Hobbie)

ALUMNO_CURSA_MATERIA (ALUMNO_ID, MATERIA_ID)

ALUMNO_TIENE_HOBBIE (ALUMNO_ID, HOBBIE_ID)

Restricciones

ALUMNO VS GRUPO:

GRUPO.ID_GRUPO debe estar en ALUMNO.GRUPO_ID

ALUMNO.GRUPO_ID no puede ser NULL

ALUMNO VS HOBBIE:

ALUMNO.ID_ALUMNO puede no estar en ALUMNO_TIENE_HOBBIE.ALUMNO_ID

TRABAJO VS RUBRO:

RUBRO.ID_RUBRO debe estar en TRABAJO.RUBRO_ID

TRABAJO.RUBRO_ID no puede ser NULL

ALUMNO VS LOCALIDAD:

LOCALIDAD.ID_LOCALIDAD debe estar en ALUMNO.LOCALIDAD_ID

ALUMNO.LOCALIDAD_ID no puede ser NULL

ALUMNO VS TRABAJO:

TRABAJO.ID_TRABAJO debe estar en ALUMNO.TRABAJO_ID

ALUMNO.TRABAJO_ID puede ser NULL

ALUMNO VS MATERIA:

ALUMNO_CURSA_MATERIA.MATERIA_ID debe estar en MATERIA.ID_MATERIA

ALUMNO_CURSA_MATERIA.ALUMNO_ID debe estar en ALUMNO.ID_ALUMNO

2. Implementación del Esquema SQL

Teniendo como referencia el **DER** y los datos ya limpios del excel procedemos a crear las tablas de nuestra base de datos.

En la creación de las mismas agregamos la relación entre ellas y seteamos valores '**DEFAULT**' en ciertas columnas. Por ejemplo a la hora de cargar un alumno si se deja vacío el campo de experiencia en base de datos tanto relacional como no relacional, así como el campo de mascotas el valor por defecto será FALSE.

Además, especificamos el comportamiento que debería tomar la tabla al eliminar un dato que esté siendo referenciado en otra tabla. Veamos los casos planteados:

- **ON DELETE SET NULL:** Esta restricción se puede ver en la tabla alumnos con respecto a la referencia que apunta a la localidad, el trabajo y el grupo. Lo que ocurriría al eliminar algún dato es que el alumno no quede apuntando a una referencia que ya no existe sino que esta se actualice con un valor **NULL**.
- **ON DELETE CASCADE:** Esta restricción se puede ver en las tablas intermedias **ALUMNO_TIENE_HOBBIE** y **ALUMNO_CURSA_MATERIA**. Su función es la de eliminar una asociación si alguna de las partes deja de existir. Es decir si se tiene al *alumno X* asociado con *Bases de datos* en la tabla de **ALUMNO_CURSA_MATERIA**, al eliminar al *alumno X* el **ON DELETE CASCADE** eliminará esta asociación en la tabla intermedia.
- **Restricciones en tablas intermedias:** En las dos tablas intermedias de nuestra base de datos se creó una restricción **UNIQUE** que asegura que ninguna relación entre las dos entidades relacionadas se repita más de una vez.

Al usar el **'CREATE TABLE'** agregamos **'IF NOT EXISTS'** para que el programa avise en caso de que la tabla ya exista y no se ejecute nuevamente.

```
CREATE TABLE IF NOT EXISTS MATERIA (  
  id_materia SERIAL PRIMARY KEY,  
  nombre_materia VARCHAR (25)  
);  
  
CREATE TABLE IF NOT EXISTS LOCALIDAD (  
  id_localidad SERIAL PRIMARY KEY,  
  nombre_localidad VARCHAR (25)  
);  
  
CREATE TABLE IF NOT EXISTS GRUPO (  
  id_grupo SERIAL PRIMARY KEY,  
  nombre_grupo VARCHAR (25)  
);  
  
CREATE TABLE IF NOT EXISTS HOBBIE (  
  id_hobbie SERIAL PRIMARY KEY,  
  nombre_hobbie VARCHAR (50)  
);
```



```
CREATE TABLE IF NOT EXISTS RUBRO (  
  id_rubro SERIAL PRIMARY KEY,  
  nombre_rubro VARCHAR (25)  
);  
  
CREATE TABLE IF NOT EXISTS TRABAJO (  
  id_trabajo SERIAL PRIMARY KEY,  
  nombre_trabajo VARCHAR(50),  
  rubro_fk INT,  
  FOREIGN KEY (rubro_fk) REFERENCES RUBRO(id_rubro) ON DELETE CASCADE  
);
```

Ahora que tenemos todas las tablas base, podemos crear la tabla **ALUMNO**, con las **foreign keys** correspondientes.

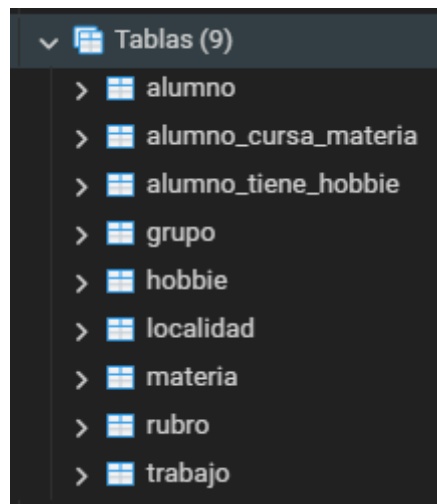
```
CREATE TABLE IF NOT EXISTS ALUMNO (  
  id_alumno SERIAL PRIMARY KEY,  
  dni INT,  
  nombre_alumno VARCHAR (50),  
  apellido VARCHAR(50),  
  email VARCHAR(50),  
  grupo_fk INT,  
  FOREIGN KEY (grupo_fk) REFERENCES GRUPO(id_grupo) ON DELETE SET NULL,  
  trabajo_fk INT,  
  FOREIGN KEY (trabajo_fk) REFERENCES TRABAJO(id_trabajo) ON DELETE SET NULL,  
  experiencia_relacional BOOLEAN DEFAULT FALSE,  
  experiencia_no_relacional BOOLEAN DEFAULT FALSE,  
  localidad_fk INT,  
  FOREIGN KEY (localidad_fk) REFERENCES LOCALIDAD(id_localidad) ON DELETE SET NULL,  
  mascotas BOOLEAN DEFAULT FALSE  
);
```

Teniendo la tabla '**ALUMNO**', podemos pasar a crear las tablas intermedias '**ALUMNO_CURSA_MATERIA**' y '**ALUMNO_TIENE_HOBBIE**', utilizando dos foreign keys en cada una, que referencian a las dos tablas que la componen.

```
CREATE TABLE IF NOT EXISTS ALUMNO_CURSA_MATERIA (  
  alumno_materia_id SERIAL PRIMARY KEY,  
  materia_id INT,  
  FOREIGN KEY (materia_id) REFERENCES MATERIA(id_materia) ON DELETE CASCADE,  
  alumno_id INT,
```

```
FOREIGN KEY (alumno_id) REFERENCES ALUMNO(id_alumno) ON DELETE CASCADE,  
CONSTRAINT uc_alumno_materia UNIQUE (materia_id, alumno_id)  
);  
  
CREATE TABLE IF NOT EXISTS ALUMNO_TIENE_HOBBIE (  
alumno_hobbie_id SERIAL PRIMARY KEY,  
hobbie_id INT,  
FOREIGN KEY (hobbie_id) REFERENCES HOBBIE(id_hobbie) ON DELETE CASCADE,  
alumno_id INT,  
FOREIGN KEY (alumno_id) REFERENCES ALUMNO (id_alumno) ON DELETE CASCADE,  
CONSTRAINT uc_alumno_hobbie UNIQUE (hobbie_id, alumno_id)  
);
```

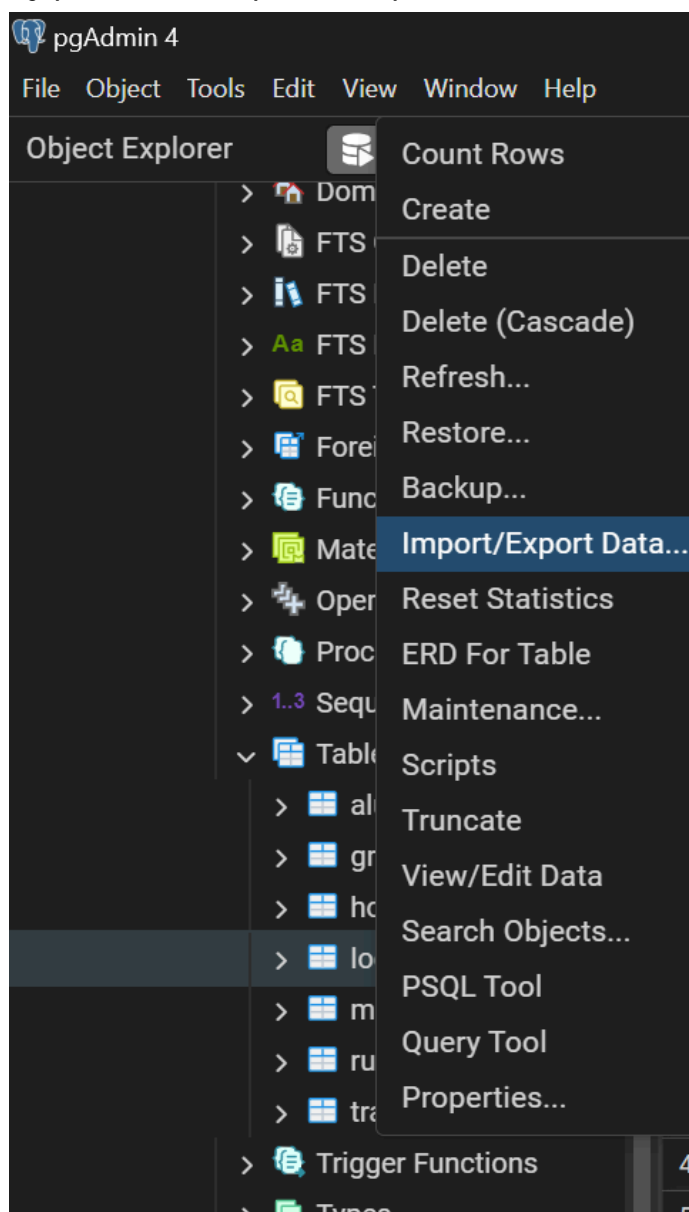
Vemos entonces en el panel de **pgAdmin** como se ve reflejado la creación de las tablas:



Parte 4: Carga y Manipulación de Datos

1. Carga inicial de los datos

Para importar los datos desde nuestro excel de alumnos, exportamos cada hoja, donde tenemos las tablas de cada entidad, como csv. Luego cliqueamos en la tabla en pgAdmin y ponemos “Importar/Exportar Data...”



Seleccionamos la ruta donde está guardado nuestro archivo y luego chequeamos que esté ingresando las columnas correctas en las columnas correspondientes de la tabla de la base de datos.

Import/Export data - table 'localidad' ✕

General Options Columns

Import/Export ✓ Import Export

Filename C:\Users\... \Downloads\TP Final BBDD\localidad.csv 📁

Format csv ▾

Encoding Select an item... ▾

i ? ✕ Close ↺ Reset ✓ OK

Import/Export data - table 'localidad' ✕

General Options Columns

Columns to export id_localidad ✕ nombre_localidad ✕ ▾

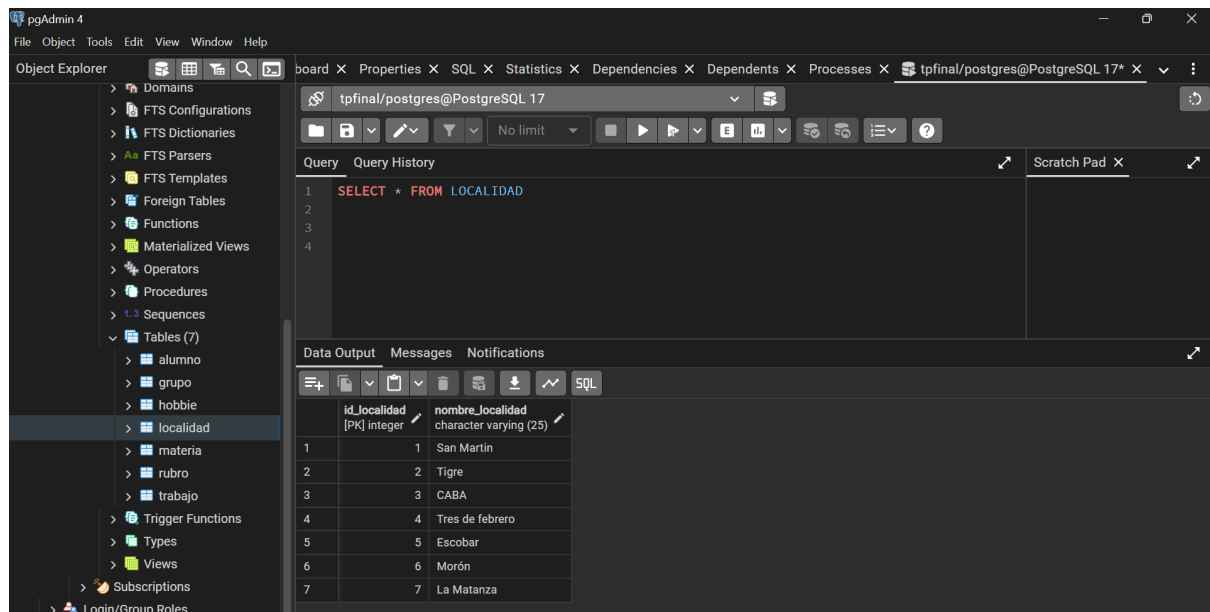
An optional list of columns to be copied. If no column list is specified, all columns of the table will be copied.

NOT NULL columns Not null columns... ▾

Do not match the specified column values against the null string. In the default case where the null string is empty, this means that empty values will be read as zero-length strings rather than nulls, even when they are not quoted. This option is allowed only in import, and only when using CSV format.

i ? ✕ Close ↺ Reset ✓ OK

Si hacemos un **'SELECT * FROM LOCALIDAD'** podemos ver que se cargaron correctamente los datos desde el archivo csv.



Aquí se puede ver como queda la tabla 'ALUMNOS'.

	id_alumno [PK] integer	dni integer	nombre_alumno character varying (50)	apellido character varying (50)	email character varying (50)	grupo_fk integer	trabajo_fk integer	experiencia_ boolean
1	1	37688075	Gabriel	Jotallan Calvetti	gaabicarp@gmail.com	1	1	true
2	2	41472398	Mariel	Nadine Kovinchich	marielkov1998@gmail.com	1	[null]	false
3	3	38532025	Gastón Ezequiel	Abelardo	gabelardo@estudiantes.unsam.edu.ar	1	3	false
4	4	40663606	Rodrigo Nicolas	Pavon Gomez	rodrigopavongomez@gmail.com	1	1	false
5	5	38601662	Maria Cecilia	Dragonetti	mdragonetti@estudiantes.unsam.edu.ar	1	1	true
6	6	44547586	Juan Ignacio	Caceffo	jicaceffo@estudiantes.unsam.edu.ar	2	4	true
7	7	37984582	Julián	Gibelli	juligibelli@gmail.com	2	1	true
8	8	35972409	Emiliano Javier	Núñez	ejnunez@estudiantes.unsam.edu.ar	2	5	true
9	9	43036494	Alejo	Menini	amenini@estudiantes.unsam.edu.ar	2	4	true
10	10	35726203	Facundo	Barneche	fh.barneche@gmail.com	2	1	true
11	11	43245286	Valentino	Bortolussi	valentino.bortolussi.lembo@gmail.com	3	[null]	false
12	12	42647332	Santiago	Bouza	sbouza@estudiantes.unsam.edu.ar	3	[null]	false
13	13	38703368	Cristian	Lomas	cristian.lomas.a@gmail.com	3	1	true
14	14	42472348	Tomás	Neiro	tomasneiro@hotmail.com	3	6	true
15	15	37626822	Cristian	Villafanez Sobrecasa	ccvillafanezsobrecasa@estudiantes.unsam.edu.ar	3	1	true

2. Consultas Básicas y Reportes

a) Localidad - Hobbie Predominante - Total de Alumnos

```
SELECT localidad, hobbie, total_alumnos
FROM (
    SELECT DISTINCT ON (LOCALIDAD.nombre_localidad)
        LOCALIDAD.nombre_localidad AS localidad,
        HOBBIE.nombre_hobbie AS hobbie,
        COUNT(ALUMNO_TIENE_HOBBIE.alumno_id) AS total_alumnos
    FROM ALUMNO_TIENE_HOBBIE
    INNER JOIN HOBBIE ON ALUMNO_TIENE_HOBBIE.hobbie_id = HOBBIE.id_hobbie
    INNER JOIN ALUMNO ON ALUMNO_TIENE_HOBBIE.alumno_id = ALUMNO.id_alumno
    INNER JOIN LOCALIDAD ON ALUMNO.localidad_fk = LOCALIDAD.id_localidad
    GROUP BY localidad, hobbie
    ORDER BY localidad, total_alumnos DESC
)
ORDER BY total_alumnos DESC;
```

Esta consulta tiene como eje dos puntos claves:

- Por un lado una consulta global que envuelve a una subconsulta. La subconsulta selecciona el hobbie más popular en cada localidad y la consulta externa organiza esos resultados por popularidad total a nivel global.
- Por otro lado, el uso de **DISTINCT ON**, una cláusula exclusiva de PostgreSQL.² Esta cláusula permite obtener un registro único por grupo, basado en las columnas especificadas en el paréntesis, seleccionando el primer registro según el orden definido en la cláusula **ORDER BY**.

Output:

	localidad character varying (25) 🔒	hobbie character varying (50) 🔒	total_alumnos bigint 🔒
1	San Martin	Deporte	9
2	CABA	Deporte	7
3	Tres de febrero	Deporte	5
4	Tigre	Deporte	1
5	Morón	Musica Rock Nacional	1
6	Escobar	Musica Rock Internacional	1
7	La Matanza	Musica Rap	1

² También hay cláusulas similares en otros sistema de gestión de bases de datos (SGBD), como por ejemplo **ROW_NUMBER** en SQL Server.
<https://learn.microsoft.com/en-us/sql/t-sql/functions/row-number-transact-sql?view=sql-server-ver16>

Análisis de datos. Luego de la consulta podemos hacer algunas observaciones:

- A nivel general vemos que el hobby '**Deporte**' es de los más practicados en la mayoría de las localidades.
- Se ve también la predominancia de los alumnos que viven en **San Martín, CABA y Tres de Febrero**, siendo lógico por la ubicación geográfica de la universidad.

b) Alumnos - Materias - Experiencia en base de datos

```
SELECT
    apellido || ', ' || nombre_alumno AS "Alumno" ,
    COUNT(ALUMNO_CURSA_MATERIA.materia_id) AS "Materias en curso",
    experiencia_relacional AS "Experiencia BD Relacional",
    experiencia_no_relacional AS "Experiencia BD No Relacional"
FROM ALUMNO
INNER JOIN ALUMNO_CURSA_MATERIA ON ALUMNO.id_alumno =
ALUMNO_CURSA_MATERIA.alumno_id
GROUP BY "Alumno", experiencia_relacional, experiencia_no_relacional
ORDER BY "Materias en curso" DESC
LIMIT 5;
```

Output:

	Alumno text	Materias en curso bigint	Experiencia BD Relacional boolean	Experiencia BD No Relacional boolean
1	Sabbatini, Tatiana	3	false	false
2	Narmontas Bocci, Theo	3	false	false
3	Neiro , Tomás	3	true	true
4	Ruina, Mariano	3	true	true
5	Lentz, Diego	3	true	false

Análisis de datos. Luego de la consulta podemos hacer algunas observaciones:

- En cuanto a las materias cursadas el máximo es 3. Este dato hay que leerlo sabiendo que los valores que manejamos son alumnos de la Tecnicatura en Programación Informática, lo cual el máximo de materias que se pueden inscribir por cuatrimestre es justamente 3. Si luego el sistema comienza a escalar y se obtienen datos de alumnos de otras carreras la tendencia debería ser un poco más variable.
- Con respecto a la experiencia en **Base de datos Relacional** y **No Relacional** notamos que en la comparativa entre ellas es muy pareja, con una leve tendencia hacia las relacionales. Sin embargo debemos tener en cuenta que estamos aplicando un **LIMIT 5** por lo que las conclusiones acerca de este tema no pueden ser muy certeras. Lo que sí podemos ver

es que si no discriminamos por relacional y no relacional todos tendrían experiencia en bases de datos. Una explicación posible es que los alumnos que alimentan nuestra base de datos son alumnos que cursan la materia **Bases de Datos**, valga la redundancia. Dichos alumnos están en una etapa avanzada en la carrera y por ello ya habrían tenido contacto con esta disciplina, ya sea en otras materias o en sus respectivos empleos.

c) Materia popular por grupo - Porcentaje inscripto en esa materia

```
SELECT DISTINCT ON (GRUPO.nombre_grupo)
    GRUPO.nombre_grupo AS "Grupo",
    MATERIA.nombre_materia AS "Materia",
    COUNT(ALUMNO_CURSA_MATERIA.alumno_id) AS "Alumnos inscriptos",
    (COUNT(ALUMNO_CURSA_MATERIA.alumno_id)*100/5)|| '%' AS "% del Grupo"
FROM ALUMNO
INNER JOIN GRUPO ON GRUPO.id_grupo = ALUMNO.grupo_fk
INNER JOIN ALUMNO_CURSA_MATERIA ON ALUMNO.id_alumno =
ALUMNO_CURSA_MATERIA.alumno_id
INNER JOIN MATERIA ON MATERIA.id_materia = ALUMNO_CURSA_MATERIA.materia_id
GROUP BY "Grupo", "Materia"
ORDER BY "Grupo", "Alumnos inscriptos" DESC
```

Output:

	Grupo character varying (25) 🔒	Materia character varying (25) 🔒	Alumnos inscriptos bigint 🔒	% del Grupo text 🔒
1	DataMasters	BBDD	5	100%
2	DreamTeam	BBDD	5	100%
3	DropTable	BBDD	5	100%
4	Enrutados	BBDD	5	100%
5	Mandarina	BBDD	5	100%
6	MCTeam	BBDD	5	100%
7	nullpointer	BBDD	5	100%
8	Okupas	BBDD	5	100%
9	Undefined	ALGO 3	5	100%

Análisis de datos. Luego de la consulta podemos hacer algunas observaciones:

- Nuevamente esta consulta tiene un techo que no permite sacarle todo el jugo posible debido a que nuestros datos corresponden a alumnos de la carrera de Programación Informática que cursan como mínimo Bases de Datos. Por esta razón en la mayoría de los grupos **BBDD** es la materia más popular salvo para el grupo Undefined que tiene **ALGO 3**. La razón de esto

es que el **DISTINCT ON** se queda el primer valor según el **ORDER BY** establecido y para Undefined este dato era **ALGO 3**.

d) Alumnos - Experiencia en BBDD - Actividad/Trabajo

En esta consulta entra en juego el desdoblamiento que realizamos en el **Data Processing** ya que pudimos agrupar a los alumnos según el rubro en el que trabajan, el rubro IT.

```
SELECT
    apellido || ', ' || nombre_alumno AS "Alumno",
    experiencia_relacional AS "Experiencia BD Relacional",
    experiencia_no_relacional AS "Experiencia BD No Relacional",
    TRABAJO.nombre_trabajo AS "Actividad"
FROM ALUMNO
INNER JOIN TRABAJO ON TRABAJO.id_trabajo = ALUMNO.trabajo_fk
INNER JOIN RUBRO ON RUBRO.id_rubro = TRABAJO.rubro_fk
WHERE RUBRO.id_rubro = 6 -- "IT"
ORDER BY experiencia_relacional DESC, experiencia_no_relacional DESC
LIMIT 5;
```

Output:

	Alumno text	Experiencia BD Relacional boolean	Experiencia BD No Relacional boolean	Actividad character varying (50)
1	Menini, Alejo	true	true	Ingeniero de datos
2	Barneche, Facundo	true	true	Desarrollador de Software
3	Caceffo, Juan Ignacio	true	true	Ingeniero de datos
4	Gibelli, Julián	true	true	Desarrollador de Software
5	Lomas, Cristian	true	true	Desarrollador de Software

Análisis de datos. Luego de la consulta podemos hacer algunas observaciones:

- Si bien es una muestra de 5 alumnos notamos un poco la tendencia de las consultas anteriores. Los alumnos que forman esta base de datos ya se encuentran en una etapa avanzada en la carrera por lo cual ya muchos se encuentran trabajando en el rubro **IT** y han tenido experiencia en bases de datos de ambos tipos.

3. Consultas Avanzadas y Optimización

a) Alumnos por localidad - Promedio de inscripción a materias

```
SELECT
    LOCALIDAD.nombre_localidad AS "Localidad",
    ROUND(AVG(alumno_materias.total_materias), 2) AS "Promedio de
materias",
    COUNT(DISTINCT ALUMNO.id_alumno) AS "Total Alumnos"
FROM ALUMNO
INNER JOIN LOCALIDAD ON ALUMNO.localidad_fk = LOCALIDAD.id_localidad
INNER JOIN (
    SELECT alumno_id, COUNT(materia_id) AS total_materias
    FROM ALUMNO_CURSA_MATERIA
    GROUP BY alumno_id
) AS alumno_materias ON ALUMNO.id_alumno = alumno_materias.alumno_id
GROUP BY "Localidad"
ORDER BY "Total Alumnos" DESC;
```

Output:

	Localidad character varying (25) 🔒	Promedio de materias numeric 🔒	Total Alumnos bigint 🔒
1	San Martín	2.29	17
2	CABA	2.20	10
3	Tres de febrero	2.29	7
4	Escobar	3.00	2
5	Morón	2.00	1
6	La Matanza	1.00	1
7	Tigre	1.00	1

Análisis de datos. Luego de la consulta podemos hacer algunas observaciones:

- Nuevamente la presencia geográfica se hace presente, viendo una predominancia de **San Martín**, **CABA** y **Tres de Febrero** como las localidades que más alumnos aportan a la universidad.
- Con respecto al promedio las muestras más significativas con las que podemos sacar conclusiones son las de estas tres localidades ya que en las cuatro restantes los alumnos son muy pocos.

Lo que podemos decir es que los valores son muy similares, y nuevamente sabiendo que solo tenemos datos de los alumnos de Programación, concluimos que hay una tendencia por seguir el programa de la carrera (3 materias por cuatrimestre). Si el sistema escala y tenemos alumnos de otras carreras donde se manejan otros programas las conclusiones seguro sean distintas. Esta consulta, con una buena muestra de datos, puede ser muy útil para entender la relación entre cercanía a la facultad e inscripción a materias por ejemplo. Si además las materias tendrán un atributo **VIRTUAL** de tipo booleano se podrían sacar conclusiones si aumenta la inscripción en localidades más lejanas a la universidad.

b) Alumnos por localidad - Promedio de inscripción a materias

```
SELECT DISTINCT ON (A1.id_alumno)
  A1.apellido AS "Alumno",
  A2.apellido AS "Potencial Mentor",
  H1.nombre_hobbie AS "Interés Común",
  A2.experiencia_relacional AS "Experiencia Mentor BD Relacional",
  A2.experiencia_no_relacional AS "Experiencia Mentor BD NO Relacional"
FROM
  ALUMNO A1
INNER JOIN ALUMNO_TIENE_HOBBIE AH1 ON A1.id_alumno = AH1.alumno_id
INNER JOIN HOBBIE H1 ON AH1.hobbie_id = H1.id_hobbie
INNER JOIN ALUMNO_TIENE_HOBBIE AH2 ON H1.id_hobbie = AH2.hobbie_id
INNER JOIN ALUMNO A2 ON AH2.alumno_id = A2.id_alumno
WHERE
  A1.id_alumno <> A2.id_alumno -- Evitar que el alumno sea su propio mentor
  AND (
    A2.experiencia_relacional = TRUE
    OR A2.experiencia_no_relacional = TRUE
  )
ORDER BY
  A1.id_alumno
LIMIT 10;
```

Output:

	Alumno character varying (50) 	Potencial Mentor character varying (50) 	Interés Común character varying (50) 	Experiencia Mentor BD Relacional boolean 	Experiencia Mentor BD NO Relacional boolean 
1	Jotallan Calveti	Lentz	Musica Rock Nacional	true	false
2	Abelardo	Hoj	Musica Rock Internacional	true	false
3	Pavon Gomez	Neiro	Lectura	true	true
4	Dragonetti	Pugliese	Musica	true	false
5	Caceffo	Pugliese	Deporte	true	false
6	Gibelli	Neiro	Lectura	true	true
7	Nuñez	Hoj	Musica Rock Internacional	true	false
8	Menini	Pugliese	Deporte	true	false
9	Barneche	Pugliese	Deporte	true	false
10	Bortolussi	Pugliese	Deporte	true	false

Esta consulta tiene ciertos puntos claves a destacar:

- Dado que es necesario relacionar entidades de una misma tabla, resulta indispensable utilizar alias para diferenciarlas. Por un lado, la tabla de alumnos **A1** identifica al alumno que recibirá el mentoreo, mientras que la tabla **A2** selecciona al potencial mentor.
- El uso de los **INNER JOIN** es fundamental para establecer las relaciones necesarias entre tablas y lograr el emparejamiento solicitado (interés común). Además, se incluye una restricción mediante el **operador relacional <>** para evitar que un alumno sea seleccionado como su propio mentor.
- Como nuestra base de datos contaba con varios alumnos con experiencia en bases de datos, decidimos ser más exigentes con la elección del mentor y emparejar únicamente aquellos alumnos que poseen experiencia en ambos tipos de bases de datos (relacionales y no relaciones).
- Como los emparejamientos por hobby generaban múltiples coincidencias, utilizamos nuevamente **DISTINCT ON**. De esta manera obtenemos solo el primer registro de cada alumno, asegurándonos de quedarnos con un único mentor potencial para cada uno.

Uso de índices y mejora de la performance

Aunque con el volumen actual de datos de nuestra base de datos los indicadores de performance no deberían ser negativos, debemos pensar en el futuro. Si este sistema será utilizado por toda la Universidad, es probable que la tabla de alumnos crezca considerablemente con múltiples entradas.

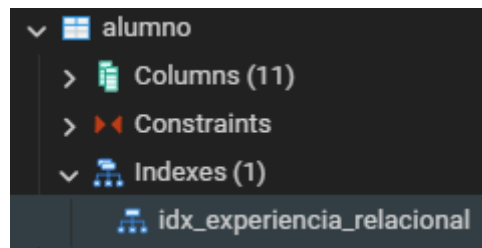
Partiendo de este hipotético caso veamos como los índices podrían ser claves para tener una mejor performance.

Como vimos anteriormente esta consulta utiliza múltiples **INNER JOIN** en la cual está involucrada la tabla de **ALUMNO** y si no indexamos algunos datos impacta en la performance ya que los emparejamientos los haría de forma secuencial.

Por ello agregar índices en las columnas involucradas es vital para mejorar el rendimiento. Si pensamos en el grueso de alumnos de toda la Universidad es muy probable que sólo una minoría tenga experiencia en bases de datos relacional y no relacional por lo cual crear índices en campos como **ALUMNO.experiencia_relacional** y **ALUMNO.experiencia_no_relacional** sería clave para mejorar los tiempos de consulta.

Veamos cómo crear este índice en PostgreSQL:

```
CREATE INDEX idx_experiencia_relacional
ON ALUMNO (id_alumno)
WHERE experiencia_relacional = TRUE;
```



Un punto a tener en cuenta es el **DISTINCT ON** que decidimos usar para solo ver un mentor. Como se mencionó anteriormente esta cláusula necesita de un **ORDER BY** por lo que este ordenamiento en una tabla de múltiples datos podría resultar poco performante. Por lo que en este caso podría ser útil crear un índice compuesto para **ALUMNO_TIENE_HOBBIE (alumno_id, hobbie_id)**.

Todas estas mejoras se pueden ir verificando en **PostgreSQL** con la herramienta **EXPLAIN** y **EXPLAIN ANALYZE** que brindan estadísticas y tiempos de una consulta. Comparando podemos saber si la implementación de algún índice mejoró la performance o no fue tan significativo



Botones EXPLAIN y EXPLAIN ANALYZE en PostgreSQL

4. Consulta Extra

Nos propusimos agregar una consulta que pueda servirle al cliente, la Universidad, para cuando la bases de datos escale y maneje un mayor volumen de datos.

- Rubros de trabajo de los alumnos y su relación con la experiencia en bases de datos: Con esta consulta podremos analizar los rubros en los que los alumnos ya poseen experiencia en bases de datos y en los que no. De esta manera se pueden sacar conclusiones y diseñar por ejemplo, programas de formación o mentoría según los rubros que lo requieran.

```
SELECT
  R.nombre_rubro AS "Rubro",
  COUNT(A.id_alumno) AS "Cantidad de alumnos",
```

```
SUM(CASE WHEN A.experiencia_relacional THEN 1 ELSE 0 END) AS
"Alumnos con Experiencia Relacional",
SUM(CASE WHEN A.experiencia_no_relacional THEN 1 ELSE 0 END) AS
"Alumnos con Experiencia No Relacional"
FROM ALUMNO A
JOIN TRABAJO T ON A.trabajo_fk = T.id_trabajo
JOIN RUBRO R ON T.rubro_fk = R.id_rubro
GROUP BY R.nombre_rubro
ORDER BY "Cantidad de alumnos" DESC;
```

Output:

	Rubro character varying (25)	Cantidad de alumnos bigint	Alumnos con Experiencia Relacional bigint	Alumnos con Experiencia No Relacional bigint
1	IT	23	18	8
2	COMERCIO	5	1	0
3	CONTABILIDAD	2	1	0
4	ADMINISTRATIVO	2	2	1
5	AUTOMOTRIZ	1	1	1
6	EDUCACION	1	0	0

Algunas explicaciones sobre el código

Para esta consulta se usó el **CASE WHEN**. Esta línea actúa como un IF/ELSE que servirá para contar la cantidad de alumnos con experiencia en bases de datos. Es decir que si este campo es TRUE suma 1 y en caso contrario 0.

Análisis de datos. Luego de la consulta podemos hacer algunas observaciones:

- Se cumple con la lógica de que quienes trabajan en IT son los que cuentan con experiencia en base de datos. La conclusión que se puede sacar es la diferencia que hay entre relacional y no relacional. Una medida que podría tomar la Universidad sería planificar cursos que enfoquen su enseñanza en las bases de datos no relacionales.

Parte 5: Procedimientos, Triggers y Automatización

1. Análisis Teórico de Procedimientos Almacenados

Un procedimiento almacenado es una función que se utiliza para realizar tareas específicas de manera automatizada. Se ejecuta directamente en el servidor, mejorando la eficiencia y reduciendo la carga en las aplicaciones cliente.

Beneficios:

- **Automatización:** Simplifican tareas repetitivas, como validaciones o actualizaciones masivas.
- **Rendimiento:** Reducen el tráfico entre cliente y servidor al agrupar múltiples operaciones en una sola ejecución.
- **Consistencia:** Garantiza que las reglas de negocio se apliquen de forma uniforme en cada operación.
- **Seguridad:** Restringen el acceso directo a los datos y delegan operaciones controladas al procedimiento almacenado.

Propuesta Teórica:

Necesitamos diseñar un procedimiento que automatice la inscripción de un alumno en una materia, asegurando que se cumplan las validaciones de cupo disponible y evitando inscripciones duplicadas.

Pasos del Procedimiento

1. **Validaciones iniciales:**
 - Verificar si el alumno ya está inscripto en la materia.
 - Comprobar que haya cupos disponibles en la materia.
2. **Inscripción:**
 - En caso que las validaciones sean exitosas, se deberá registrar al alumno en la materia.
3. **Actualización de estado:**
 - Reducir el número de cupos disponibles en la materia.
4. **Registro de actividad:**
 - Generar reporte con los detalles de la inscripción.

Beneficios del Procedimiento

- **Eficiencia:** Automatiza el registro en tiempo real, reduciendo la intervención manual.
- **Reducción de errores:** Garantiza que se cumplan las reglas de negocio, evitando duplicaciones y problemas de capacidad.

- **Adaptabilidad:** Si cambian los requerimientos (por ejemplo, incluir una validación adicional), el diseño del procedimiento permite ajustes rápidos sin afectar la funcionalidad existente.

Funcionamiento del Procedimiento

El procedimiento almacenado propuesto para la inscripción de alumnos en una materia funciona de la siguiente manera:

1. **Recepción de Parámetros:**
El procedimiento recibe como entrada los identificadores del alumno y de la materia (por ejemplo, `id_alumno` e `id_materia`).
2. **Validaciones Previas:**
 - **Inscripción existente:** Verifica en la tabla de inscripciones si el alumno ya está registrado en la materia. Si es así, se detiene la ejecución y retorna un mensaje de error propuesto por el negocio
 - **Disponibilidad de cupos:** En caso de que la primera validación se ejecute de manera correcta, comprueba si la materia tiene cupos disponibles, accediendo al campo correspondiente en la tabla de materias. Si no hay cupos, también se detiene el proceso.
3. **Inscripción:**
 - Si las validaciones son satisfactorias, se inserta un registro en la tabla de inscripciones, asociando al alumno con la materia.
4. **Actualización del Estado de Cupos:**
 - Se reduce en 1 el número de cupos disponibles en la tabla de materia, asegurando consistencia con el registro de la inscripción.
5. **Generación de reporte:**
 - Se almacena un registro en una tabla excel con detalles de la inscripción (fecha, id del alumno y materia).

Implementación Ideal en el Motor de Base de Datos Seleccionado

Como anteriormente mencionamos, elegimos el motor relacional **PostgreSQL** que puede ser ideal para implementar este procedimiento debido a su soporte robusto para procedimientos almacenados, triggers y manejo eficiente de transacciones.

1. **Transacciones:**
El procedimiento debe ejecutarse dentro de una transacción para garantizar la consistencia de los datos. Esto asegura que, si alguna validación falla o se produce un error, todos los cambios se pueden revertir.
2. **Manejo de Excepciones:**
El procedimiento debe manejar errores, como intentar registrar un

alumno duplicado o una materia sin cupos, devolviendo mensajes claros al cliente o sistema que invoque la operación.

3. Optimización mediante Índices:

- Índices en las columnas **id_alumno** e **id_materia** de las tablas correspondientes mejorarán la velocidad de las consultas y validaciones.

4. Configuración de Permisos:

Solo los usuarios autorizados deben tener permisos para ejecutar este procedimiento, garantizando la seguridad y evitando modificaciones no deseadas.

2. Análisis Teórico de Triggers

Los **Triggers** o **Disparadores** son herramientas de automatización que realizan tareas en respuesta a eventos específicos de sentencias o comandos.

Existen distintos niveles de trigger:

- **Nivel DML:** Reaccionan a comandos como **INSERT**, **DELETE** o **UPDATE**
- **Nivel DDL:** Reaccionan a comandos como **CREATE**, **ALTER** o **DROP**. Son útiles para controlar los cambios de esquema, auditar las modificaciones de la base de datos y aplicar políticas de seguridad.
- **Nivel de inicio de sesión:** Los triggers de inicio de sesión suelen ejecutarse en respuesta a un evento **LOGON**. Suelen utilizarse para controlar o supervisar las sesiones de usuario, aplicar políticas de inicio de sesión o registrar la actividad de los usuarios. Por ejemplo, un trigger de inicio de sesión puede limitar el acceso a determinadas horas o registrar la hora de inicio de sesión de cada usuario y su dirección IP.

Un trigger también puede verse como un tipo de procedimiento almacenado. Este se activará automáticamente en respuesta a la ejecución de alguno de estos eventos – **INSERT**, **DELETE**, **UPDATE**, etc. – en una tabla o vista específica de una base de datos. Se puede establecer su ejecución **antes** o **después** de los eventos.

Para definir un trigger se tiene que **especificar la tabla o vista** a la que se va a asociar, el **evento que lo activa** y **cuándo ocurre** si antes – **BEFORE** – o después – **AFTER** –. Además, con los triggers se pueden manejar pseudo-tablas para manejar los datos, **NEW** que contiene los valores nuevos del registro en los eventos **INSERT** o **UPDATE** y **OLD** que contiene los valores antiguos del registro en los eventos **DELETE** o **UPDATE**.

Funcionalidad de un trigger

La función principal de los triggers es **automatizar operaciones**, ahorrando tiempo y reduciendo la intervención manual. Mejoran la seguridad e integridad al implementar restricciones y verificaciones, minimizando errores y sincronizando datos. Su principal ventaja es que **operan directamente desde la base de datos**, sin necesidad de lenguajes externos.

Los podemos ver en casos como:

- Auditorías
- Validaciones de datos antes de guardarlos en las tablas finales
- Sincronización entre tablas relacionadas
- Generación de reportes automáticos basados en cambios
- Restringir operaciones no permitidas

Buenas prácticas

Como se fue demostrando los triggers pueden ser muy ventajosos para nuestras bases de datos pero es bueno seguir buenas prácticas para garantizar que el efecto no sea negativo. Veamos alguna de ellas:

1. **Triggers simples y eficientes:** No deben tener lógica compleja o tareas pesadas para no afectar el rendimiento y así evitar el cuello de botella.
2. **Evitar la lógica de negocio compleja:** Dejar que estas reglas se apliquen en el código de las aplicaciones o procedimientos almacenados. Esto mejorará la claridad y manejabilidad del trigger.
3. **Documentación:** La documentación adecuada es fundamental para mantener y comprender los triggers. Esta práctica que atraviesa toda la disciplina es vital para que otras personas puedan comprender y mantener los triggers de forma eficaz.
4. **Analizar ventajas y desventajas:** Siempre es bueno hacer este análisis antes de iniciar una tarea, algunas variables a analizar podrían ser³:

³ <https://www.datacamp.com/es/tutorial/sql-triggers>

Ventajas	Desventajas
Ejecución automatizada de tareas	Posible sobrecarga de rendimiento
Mayor integridad de los datos	Complejidad en la resolución de problemas
Tratamiento de errores y registro	Riesgo de crear bucles infinitos

5. **Alternativas a los triggers:** Si decidimos no utilizarlos por alguna razón hay que saber que contamos con otras herramientas que podrían suplantar la tarea del trigger, estos son los procedimientos almacenados, restricciones de comprobación y las claves externas.

Creando un Trigger en PostgreSQL

La creación de un trigger en PostgreSQL cuenta con dos partes, en primer lugar **programar una función** que indicará la acción del trigger propiamente dicho:

```
CREATE FUNCTION funcion_programada()  
    RETURNS trigger AS  
$BODY$  
BEGIN  
    -- En esta línea se escribe la operación que se desea automatizar  
    RETURN NEW;  
END;  
$BODY$  
LANGUAGE plpgsql;
```

Una vez que esta función esté creada el paso siguiente es **crear el trigger** del cual la sintaxis es bastante clara como se ve a continuación:

```
CREATE TRIGGER nombre_trigger  
{BEFORE | AFTER | INSTEAD OF} {event [OR ...]}  
    ON tabla  
[FOR [EACH] {ROW | STATEMENT}]  
    EXECUTE PROCEDURE función_programada;
```

Propuesta de Trigger para nuestra base de datos

Inicialmente se planteó un trigger que registre un log automáticamente cuando se modifique algún dato de la información personal de un alumno. Sin embargo se decidió dar un pequeño paso más y que esa misma tabla de log registre también cuando alguien elimina o agrega un nuevo alumno. Para ello la función creada será la misma que actuará según la operación que el trigger reciba. Vemos a continuación cómo se implementa:

```
CREATE OR REPLACE FUNCTION trigger_log_alumno()  
RETURNS TRIGGER AS $$  
BEGIN  
    IF TG_OP = 'UPDATE' THEN  
        INSERT INTO log_alumno (  
            id_alumno, dni, nombre_alumno, apellido, email, grupo_fk,  
            trabajo_fk, experiencia_relacional, experiencia_no_relacional, localidad_fk,  
            mascotas, fecha_log, descripcion, usuario  
        )  
        VALUES (  
            OLD.id_alumno, OLD.dni, OLD.nombre_alumno, OLD.apellido, OLD.email,  
            OLD.grupo_fk, OLD.trabajo_fk, OLD.experiencia_relacional,  
            OLD.experiencia_no_relacional, OLD.localidad_fk, OLD.mascotas,  
            NOW(), 'Update', current_user);  
    ELSIF TG_OP = 'INSERT' THEN  
        INSERT INTO log_alumno (  
            id_alumno, dni, nombre_alumno, apellido, email, grupo_fk,  
            trabajo_fk, experiencia_relacional, experiencia_no_relacional, localidad_fk,  
            mascotas, fecha_log, descripcion, usuario)  
        VALUES (  
            NEW.id_alumno, NEW.dni, NEW.nombre_alumno, NEW.apellido, NEW.email,  
            NEW.grupo_fk, NEW.trabajo_fk, NEW.experiencia_relacional,  
            NEW.experiencia_no_relacional, NEW.localidad_fk, NEW.mascotas,  
            NOW(), 'Insert', current_user);  
    ELSIF TG_OP = 'DELETE' THEN  
        INSERT INTO log_alumno (  
            id_alumno, dni, nombre_alumno, apellido, email, grupo_fk,  
            trabajo_fk, experiencia_relacional, experiencia_no_relacional, localidad_fk,  
            mascotas, fecha_log, descripcion, usuario)  
        VALUES (  
            OLD.id_alumno, OLD.dni, OLD.nombre_alumno, OLD.apellido, OLD.email,  
            OLD.grupo_fk, OLD.trabajo_fk, OLD.experiencia_relacional,  
            OLD.experiencia_no_relacional, OLD.localidad_fk, OLD.mascotas,  
            NOW(), 'Delete', current_user);  
    END IF;  
    RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;
```

Además de la función se creó la tabla **LOG_ALUMNO** que registrará los datos involucrados y además la hora en la que se registró ese cambio, la descripción, es decir si fue un **INSERT**, un **DELETE** o un **UPDATE** y el usuario que lo realizó.

```
CREATE TABLE LOG_ALUMNO (  
  id_alumno INT,  
  dni INT,  
  nombre_alumno VARCHAR(50),  
  apellido VARCHAR(50),  
  email VARCHAR(50),  
  grupo_fk INT,  
  trabajo_fk INT,  
  experiencia_relacional BOOLEAN,  
  experiencia_no_relacional BOOLEAN,  
  localidad_fk INT,  
  mascotas BOOLEAN,  
  fecha_log TIMESTAMP,  
  descripcion VARCHAR(10),  
  usuario VARCHAR(30)  
);
```

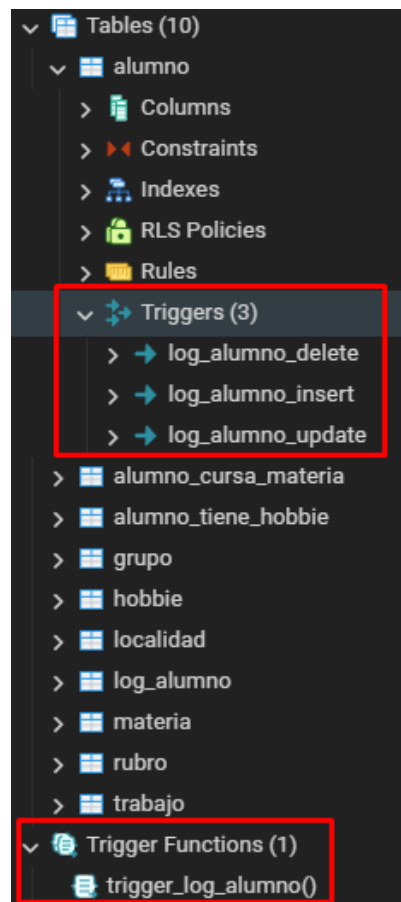

El siguiente paso es crear los triggers asociados a la tabla **ALUMNO**:

```
CREATE TRIGGER log_alumno_update
AFTER UPDATE ON alumno
FOR EACH ROW
EXECUTE FUNCTION trigger_log_alumno ();

CREATE TRIGGER log_alumno_delete
AFTER DELETE ON alumno
FOR EACH ROW
EXECUTE FUNCTION trigger_log_alumno ();

CREATE TRIGGER log_alumno_insert
AFTER INSERT ON alumno
FOR EACH ROW
EXECUTE FUNCTION trigger_log_alumno ();
```

Vemos que en el panel de PostgreSQL tenemos la función que utilizan los tres triggers y también los tres triggers asociados a la tabla **ALUMNO**:



Con esto es suficiente, ahora solo queda probarlo y ver los resultados:

CASO UPDATE

```
UPDATE ALUMNO SET email = 'pruebatrigger@gmail.com'
WHERE id_alumno = 3;

UPDATE ALUMNO SET email = 'gabelardo@estudiantes.unsam.edu.ar'
WHERE id_alumno = 3;

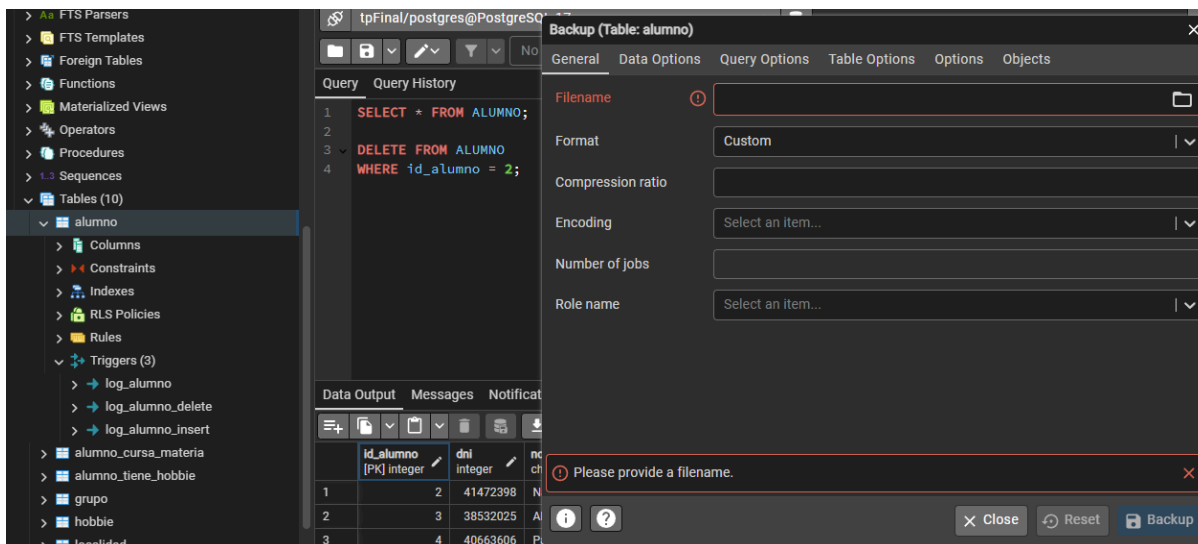
SELECT id_alumno, email, fecha_log, descripcion, usuario FROM
LOG_ALUMNO;
```

Output tabla LOG:

	id_alumno integer	email character varying (50)	fecha_log timestamp without time zone	descripcion character varying (10)	usuario character varying (30)
1	3	gabelardo@estudiantes.unsam.edu.ar	2024-11-22 17:18:14.200318	Update	postgres
2	3	pruebatrigger@gmail.com	2024-11-22 17:18:16.888327	Update	postgres

CASO DELETE

Antes de realizar un delete hacemos un backup:



```
DELETE FROM ALUMNO
WHERE id_alumno = 4;

SELECT id_alumno, email, fecha_log, descripcion, usuario FROM
LOG_ALUMNO;
```

Output tabla LOG:

	id_alumno integer	email character varying (50)	fecha_log timestamp without time zone	descripcion character varying (10)	usuario character varying (30)
1	3	gabelardo@estudiantes.unsam.edu.ar	2024-11-22 17:18:14.200318	Update	postgres
2	3	pruebatrigger@gmail.com	2024-11-22 17:18:16.888327	Update	postgres
3	4	rodrigopavongomez@gmail.com	2024-11-22 17:19:45.247675	Delete	postgres

CASO INSERT

Ahora probamos agregando un alumno con un INSERT

```
INSERT INTO ALUMNO (
dni,
nombre_alumno,
apellido,
email,
grupo_fk,
trabajo_fk,
experiencia_relacional,
experiencia_no_relacional,
localidad_fk,
mascotas )
VALUES (40663606, 'Rodrigo Nicolas', 'Pavon Gomez',
'rodrigopavongomez@gmail.com', 1,1,false, true, null, true);

SELECT id_alumno, email, fecha_log, descripcion, usuario FROM LOG_ALUMNO;
```

Output tabla LOG:

	id_alumno integer	email character varying (50)	fecha_log timestamp without time zone	descripcion character varying (10)	usuario character varying (30)
1	3	gabelardo@estudiantes.unsam.edu.ar	2024-11-22 17:18:14.200318	Update	postgres
2	3	pruebatrigger@gmail.com	2024-11-22 17:18:16.888327	Update	postgres
3	4	rodrigopavongomez@gmail.com	2024-11-22 17:19:45.247675	Delete	postgres
4	49	rodrigopavongomez@gmail.com	2024-11-22 17:26:11.286758	Insert	postgres

Beneficio de este trigger

En nuestra implementación presentamos un trigger de auditoría. Pensándolo en el contexto de una base de datos para la universidad las ventajas que este da se podrían resumir en los siguientes puntos:

- Registra automáticamente acciones importantes en una tabla vital como la de **ALUMNO**. Dejando registro de quién realizó esa acción, cuál fue la acción y sobre que datos.
- En cuestiones de seguridad también es clave ya que se podrían **detectar accesos no autorizados o actividades sospechosas** si sabemos quiénes son los usuarios habilitados a tratar esa tabla.

- Tener estos registros puede ayudar a **solucionar discrepancias o errores** en los datos como cambios inesperados en los registros de alumnos.
- Con el log quienes administren la base de datos de la universidad pueden **analizar tendencias en las modificaciones** y en base a esto tomar ciertas decisiones o realizar algún proceso administrativo.

En resumen los triggers son muy ventajosos si queremos tener una base de datos que **asegure la integridad de los registros** y además **crea un entorno seguro y transparente**, lo cual es crucial para una institución educativa.

3. Propuestas de Estrategias de Automatización

Se llama automatizaciones en base de datos a los conjuntos de acciones o transacciones configuradas para que se realicen de manera repetitiva y automática en un momento determinado. Este momento puede ser un horario, como por ejemplo suele suceder en los backups, o cuando se efectúa un cambio específico en registros o configuraciones dentro de la base de datos.

Los objetivos de estas prácticas suelen ser asegurar la integridad de datos, reducir la carga de mano de obra realizando tareas repetitivas de manera programada y eficiente o mejorar la disponibilidad de datos. En resumen, optimizan el manejo general de la base.

Automatización con Backups

Una de las automatizaciones más comunes y esenciales en el manejo de bases de datos es la programación de **backups** automáticos. En el contexto del modelo de negocio planteado, estos pueden configurarse para realizarse de manera incremental cada hora y de manera completa diariamente durante horarios no laborales. Esto asegura que, ante un error humano o un fallo en el sistema, sea posible recuperar los datos hasta el último punto de respaldo, garantizando así la continuidad operativa y la integridad de los registros.

Herramientas como **pg_dump**, en el caso de PostgreSQL, permiten automatizar esta tarea generando archivos de respaldo en formatos comprimidos o scripts SQL para reconstruir la base de datos fácilmente. Los scripts son archivos de texto sin formato que contienen los comandos SQL necesarios para reconstruir la base de datos y sus estados al momento en el que se realizó el guardado.

Automatización con Vistas

Además de los backups, otro ejemplo relevante de automatización en bases de datos es el uso de **vistas**. Las vistas son tablas virtuales que representan un conjunto de datos generados a partir de consultas SQL. Su principal beneficio radica en que se actualizan automáticamente cada vez que cambian los datos de las tablas subyacentes. Esto asegura que los usuarios accedan siempre a información actualizada sin tener que realizar consultas complejas de manera manual.

En el contexto del modelo de negocio planteado, las vistas pueden ser utilizadas para **simplificar la gestión de información** de alumnos y materias. Por ejemplo, se puede crear una vista que combine automáticamente datos de distintas tablas para que el personal académico o administrativo acceda de manera eficiente a información relevante, como el historial académico de un alumno o estadísticas generales.

Las vistas también contribuyen a la seguridad de los datos, ya que permiten controlar el acceso a cierta información. Por ejemplo, se pueden crear vistas para que excluyan datos sensibles, como identificadores únicos o información confidencial, permitiendo que ciertos usuarios consulten únicamente los campos relevantes para su función.

Optimización de Consultas con Índices

Por otro lado, los **índices** son estructuras de datos destinadas a mejorar el rendimiento de las consultas en una base de datos. Estos permiten realizar búsquedas rápidas sin necesidad de recorrer todas las filas de las tablas. En un entorno donde se gestionan datos de alumnos y materias, los índices pueden implementarse automáticamente en campos clave, como `id_alumnos`, `localidad` o `id_trabajo`, para acelerar las consultas más frecuentes. Por ejemplo, buscar todos los alumnos de una localidad específica o los registros asociados a un trabajo particular.

La automatización del uso de índices puede mejorar significativamente el rendimiento del sistema al optimizar las consultas recurrentes de manera dinámica.

Impacto de estas prácticas

- **Eficiencia:** Las vistas simplifican consultas complejas, los índices reducen el tiempo de ejecución de las operaciones y los backups programados aseguran que los datos estén disponibles sin interrupciones.
- **Seguridad:** Las vistas limitan el acceso a información sensible, mientras que los backups protegen contra pérdidas de datos, permitiendo una recuperación eficiente.
- **Disponibilidad:** Automatizar tareas como la creación de vistas, el mantenimiento de índices y los backups periódicos asegura que el sistema se mantenga actualizado, operativo y protegido ante contingencias.
- **Integridad:** La combinación de estas estrategias garantiza la consistencia y confiabilidad de la información, incluso en casos de fallos técnicos o errores humanos.

Propuestas para nuestra base de datos:

Automatización mediante Backups Programados

Diseñar un sistema de backup automatizado que utilice **pg_dump** para generar respaldos diarios. Esto se puede configurar en PostgreSQL utilizando **cron jobs** en sistemas basados en Unix o tareas programadas en Windows. Los backups se

almacenarán en un sistema externo o en la nube para garantizar la seguridad y la recuperación ante desastres.

Ejemplo de código para backups programados usando **pg_dump**:

```
# Backup incremental (cada hora):
pg_dump -U usuario -h localhost -d base_datos --data-only --format=c >
/ruta/incremental_$(date +%Y-%m-%d_%H-%M).dump

# Backup completo (una vez al día):
pg_dump -U usuario -h localhost -d base_datos --format=custom >
/ruta/completo_$(date +%Y-%m-%d).dump
```

Estos comandos se pueden integrar en un **cron job** para automatizarlos:

```
# Cron job para backups incrementales y completos
0 * * * * /path/to/backup_incremental.sh # Cada hora
0 2 * * * /path/to/backup_completo.sh # Diario a las 2:00 AM
```

Uso de Vistas para Optimizar Consultas Recurrentes

Crear vistas que agrupen información comúnmente solicitada. Por ejemplo, una vista que combine datos de alumnos con su localidad, trabajos, y materias cursadas.

Ejemplo de creación de una vista:

```
CREATE VIEW vista_alumno_detalle AS
SELECT
    A.id_alumno,
    A.nombre_alumno,
    L.nombre_localidad,
    T.nombre_trabajo,
    COUNT(ACM.materia_id) AS total_materias
FROM
    ALUMNO A
LEFT JOIN LOCALIDAD L ON A.localidad_fk = L.id_localidad
LEFT JOIN TRABAJO T ON A.trabajo_fk = T.id_trabajo
LEFT JOIN ALUMNO_CURSA_MATERIA ACM ON A.id_alumno = ACM.alumno_id
GROUP BY A.id_alumno, L.nombre_localidad, T.nombre_trabajo;
```

Ahora, en lugar de escribir consultas complejas repetidamente, se puede consultar la vista:

```
SELECT * FROM vista_alumno_detalle WHERE nombre_localidad = 'San  
Martín';
```

Implementación de Índices para Consultas Más Eficientes

Crear índices de manera automática en las columnas que son frecuentemente utilizadas en las consultas, como *id_alumno*, *localidad_fk* y *id_trabajo*. Esto optimizará la velocidad de búsqueda, especialmente para consultas que involucran grandes volúmenes de datos en tablas como **ALUMNO** y **TRABAJO**, y mejorará las operaciones JOIN entre tablas relacionadas.

Ejemplo de creación de índices:

```
-- Índice en el identificador de alumno  
CREATE INDEX idx_id_alumno ON ALUMNO(id_alumno);  
  
-- Índice en la relación de localidades  
CREATE INDEX idx_localidad_fk ON ALUMNO(localidad_fk);  
  
-- Índice en la relación de trabajos  
CREATE INDEX idx_trabajo_fk ON ALUMNO(trabajo_fk);
```

También se pueden usar índices compuestos si las consultas involucran combinaciones de columnas:

```
-- Índice compuesto en `localidad_fk` e `id_trabajo`  
CREATE INDEX idx_localidad_trabajo ON ALUMNO(localidad_fk, trabajo_fk);
```

Impacto General en el Sistema:

1. Automatización de Backups:

- Garantiza la disponibilidad de datos ante desastres.
- Ofrece una solución proactiva a errores humanos y fallos del sistema.
- Incrementa la confianza en la seguridad de la base de datos en entornos reales.

2. Vistas:

- Centralizan lógica compleja, simplificando el acceso a los datos.
- Reducen errores en consultas repetitivas y mejoran la claridad del diseño del sistema.

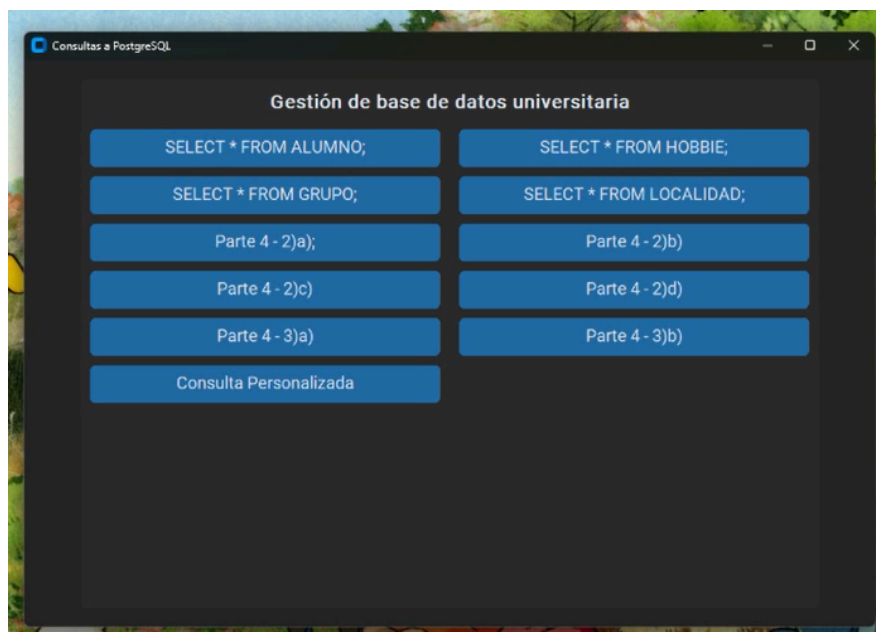
3. Índices:

- Mejoran la velocidad de las consultas.

- Reducen el uso de recursos al evitar búsquedas completas en tablas grandes.
- Proporcionan una experiencia más rápida y fluida para los usuarios finales.

Parte 6: Anexo

Adjuntamos a la carpeta general (LINK A LA CARPETA) del proyecto un **archivo ejecutable** que permite realizar consultas dinámicas, desarrollado utilizando Python.



En su implementación se utilizaron las siguientes librerías:

- **psycopg2**: para la conexión y manejo de bases de datos PostgreSQL.
- **pandas**: para la manipulación y análisis de datos.
- **custom tkinter (ctk)** y **tkinter**: para crear la interfaz gráfica de usuario.
- **pandasgui**: para la visualización interactiva de los datos en un entorno gráfico.

Bibliografía

MONGODB

- <https://www.mongodb.com/docs/manual/sharding/#:~:text=Sharding%20is%20a%20method%20for,capacity%20of%20a%20single%20server.>

PROCEDIMIENTOS

- https://www.w3schools.com/sql/sql_stored_procedures.asp
- <https://learn.microsoft.com/es-es/sql/relational-databases/stored-procedures/create-a-stored-procedure?view=sql-server-ver16>

TRIGGERS

- <https://www.todopostgresql.com/postgresql-create-trigger-disparador-postgresql/>
- <https://www.postgresql.org/docs/current/index.html>
- <https://www.datasunrise.com/es/centro-de-conocimiento/auditoria-de-base-de-datos-en-postgresql/>
- <https://www.datacamp.com/es/tutorial/sql-triggers>
- <https://ayudaleyprotecciondatos.es/bases-de-datos/trigger/#:~:text=La%20principal%20funci%C3%B3n%20de%20los,e%20integridad%20de%20la%20informaci%C3%B3n.>

AUTOMATIZACIÓN

- <https://www.postgresql.org/docs/current/indexes.html>
- <https://www.postgresql.org/docs/current/rules-materializedviews.html>
- <https://www.postgresql.org/docs/current/mvcc.html>
- <https://www.digitalocean.com/community/tutorials/how-to-automate-postgresql-database-backups-in-linux>