📊 GUÍA DE BASE DE DATOS - OBULLSHIT BACKEND

> **Para:** Fundador/CEO de OBullshit
> **Propósito:** Entender completamente cómo funciona nuestra base de datos
> **Fecha:** Julio 2025
> **Estado:** Base de datos implementada y verificada ✅

🎯 RESUMEN EJECUTIVO

Esta base de datos está diseñada para almacenar **información ilimitada** sobre proyectos de startup y hacer **matching inteligente** con inversores. La arquitectura flexible te permite añadir cualquier tipo de información sin cambiar la estructura de la base de datos.

Principio clave: Columnas fijas para lo esencial + columnas JSONB para todo lo demás.

🏗️ ARQUITECTURA GENERAL

FLUJO DE DATOS PRINCIPAL:

Usuario → Proyecto → Conversaciones → Extracción de IA → Búsqueda de Inversores → Campañas → Respuestas

TABLAS CORE (las más importantes):

- `users` - Quién usa la plataforma
- `projects` - Cada startup/proyecto que busca inversión
- `conversations` - Todo el historial de chat con IA
- `investors` - Base de datos unificada de todos los inversores

TABLAS DE OPERACIONES:

- `outreach_campaigns` - Campañas de LinkedIn
- `outreach_targets` - Cada mensaje individual
- `linkedin_responses` - Respuestas recibidas

TABLAS DE NEGOCIO:

- `subscriptions` - Planes de pago
- `credit_transactions` - Historial de créditos
- `webhook_events` - Eventos de servicios externos

📋 TABLA POR TABLA: QUÉ HACE CADA UNA

🥷 USERS - Los usuarios de la plataforma

sql

users:
- email, password_hash (autenticación básica)
- stripe_customer_id (para facturación)
- unipile_account_id (LinkedIn conectado)
- credits_balance (créditos disponibles)
- plan ('free', 'pro', 'outreach')

Casos de uso comunes:

- Verificar si usuario tiene créditos suficientes
- Obtener su cuenta de LinkedIn conectada
- Actualizar su plan de suscripción

🚀 PROJECTS - El corazón de cada startup

sql

projects:
- name, description (básico)
- categories (JSONB) - ["fintech", "saas"] - OBLIGATORIO para búsquedas
- stage (text) - "seed", "series_a" - OBLIGATORIO para búsquedas
- project_data (JSONB) - ¡AQUÍ VIVE TODO LO DEMÁS!
- context_summary (text) - Resumen generado por "bot bibliotecario"

⭐ LA COLUMNA MÁGICA: project_data

Esta columna puede contener CUALQUIER información sobre el proyecto:

json

```
{
  "funding_amount": 500000,
  "team_size": 4,
  "business_model": "subscription",
  "target_market": "pymes_espana",
  "competitors": ["stripe", "paypal"],
  "key_metrics": {
    "mrr": 15000,
    "customers": 120,
    "churn_rate": 0.05
  },
  "team_background": ["ex_stripe", "ex_visa"],
  "pitch_deck_insights": {
    "problem_statement": "Los pagos online son complicados",
    "solution_uniqueness": "API más simple que Stripe"
  },
  "investor_preferences": {
    "check_size_preferred": "€25k-€100k",
    "geographic_preference": "madrid_based_preferred"
  }
}
```

¿Cómo crece esta información?

- Cada mensaje del usuario se analiza con IA
- La IA extrae nueva información y la añade automáticamente
- Nunca necesitas crear nuevas columnas

💬 CONVERSATIONS - Todo el historial de chat

sql

```
conversations:
- project_id (a qué proyecto pertenece)
- role ('user' o 'assistant')
- content (el mensaje en sí)
- ai_extractions (JSONB) - Qué extrajo la IA de este mensaje específico
- gemini_prompt_used (texto) - El prompt exacto enviado a Gemini
- gemini_response_raw (texto) - La respuesta completa de Gemini
```

Ejemplo de ai_extractions:

json

```
{
  "new_funding_info": {
    "amount": 500000,
    "currency": "EUR"
  },
  "team_update": {
    "size": 4,
    "new_hire": "CTO ex-Stripe"
  },
  "metrics_mentioned": {
    "mrr": 15000,
    "growth_rate": "20% monthly"
  }
}
```

🎯 INVESTORS - Base de datos unificada

sql

```
investors:
- type ('business_angel', 'fund', 'employee')
- full_name, email, linkedin_url (contacto)
- categories_strong (JSONB) - Categorías donde son expertos
- categories_general (JSONB) - Categorías de interés general
- stages_strong (JSONB) - Stages donde invierten principalmente
- stages_general (JSONB) - Stages donde pueden invertir
```

Datos consolidados de 4 fuentes diferentes:

- Business Angels (CSV original)
- Fondos de inversión (CSV original)
- Empleados de fondos (CSV original)
- Empresas/corporates (CSV original)

Algoritmo de matching:

```
Puntuación =
  + 20 puntos por coincidencia en stages_strong
  + 10 puntos por coincidencia en stages_general
  + 15 puntos por coincidencia en categories_strong
  + 5 puntos por coincidencia en categories_general
```

📧 OUTREACH_CAMPAIGNS - Campañas de LinkedIn

sql

outreach_campaigns:
- project_id (qué proyecto lanza la campaña)
- name, message_template
- status ('draft', 'running', 'completed')
- total_targets, sent_count, reply_count (métricas)

🎯 OUTREACH_TARGETS - Cada mensaje individual

sql

outreach_targets:
- campaign_id, investor_id (quién y a quién)
- personalized_message (mensaje personalizado)
- status ('queued', 'sent', 'replied', 'failed')
- unipile_message_id (referencia externa)
- sent_at, replied_at (timestamps)

💭 LINKEDIN_RESPONSES - Respuestas recibidas

sql

linkedin_responses:
- outreach_target_id (a qué mensaje responden)
- response_text (el texto de la respuesta)
- response_sentiment ('positive', 'neutral', 'negative')
- interest_level ('high', 'medium', 'low', 'rejection')
- ai_analysis (JSONB) - Análisis completo de Gemini

🔄 FLUJOS DE TRABAJO PRINCIPALES

1. USUARIO CHATEA CON IA

1. Usuario envía mensaje → conversations (role: 'user')
2. Backend envía a Gemini para respuesta
3. Backend envía a Gemini para extracción de datos
4. Respuesta se guarda → conversations (role: 'assistant')
5. Datos extraídos se añaden → projects.project_data
6. Cada 5 mensajes → actualizar context_summary

2. BÚSQUEDA DE INVERSORES

1. Usuario pide "buscar inversores fintech seed"
2. Backend usa projects.categories y projects.stage
3. Busca en investors donde categories/stages coincidan
4. Calcula puntuación para cada inversor
5. Devuelve top 20 ordenados por puntuación

3. CAMPAÑA DE OUTREACH

1. Usuario selecciona 30 inversores
2. Se crea outreach_campaign
3. Se crean 30 outreach_targets (status: 'queued')
4. Worker en background envía mensajes uno por uno
5. Actualiza status a 'sent' cuando se envía
6. Webhook de Unipile actualiza a 'replied' cuando responden

4. PROCESAMIENTO DE PAGOS

1. Usuario paga → Stripe webhook → webhook_events
2. Se crea subscription o se añaden créditos
3. Cada acción (búsqueda, campaña) consume créditos
4. Se registra en credit_transactions

🛡️ SEGURIDAD Y PERMISOS

ROW LEVEL SECURITY (RLS) HABILITADO:

- Cada usuario solo ve SUS datos
- No pueden acceder a proyectos de otros usuarios
- No pueden ver conversaciones de otros
- No pueden ver campañas de otros

DATOS PÚBLICOS (sin RLS):

- `investors` - Base de datos compartida
- `webhook_events` - Eventos del sistema

⚡ PERFORMANCE Y ÍNDICES

ÍNDICES CRÍTICOS CREADOS:

- Búsquedas por categorías: `projects.categories`, `investors.categories_*`
- Búsquedas por stage: `projects.stage`, `investors.stages_*`

- Relaciones FK: todos los `*_id` tienen índices
- Búsquedas JSON: índices GIN en todas las columnas JSONB

TRIGGERS AUTOMÁTICOS:

- `updated_at` se actualiza automáticamente en users, projects, investors

---

🎯 CASOS DE USO PARA PROGRAMAR

Obtener todo sobre un proyecto:

```sql
SELECT p.*,
       array_agg(c.content ORDER BY c.created_at) as conversation_history
FROM projects p
LEFT JOIN conversations c ON c.project_id = p.id
WHERE p.user_id = $user_id AND p.id = $project_id
GROUP BY p.id;
```

Buscar inversores que coincidan:

```sql
SELECT *,
       (categories_strong ?| array['fintech', 'saas']) as strong_match,
       (stages_strong ? 'seed') as stage_match
FROM investors
WHERE categories_general ?| array['fintech', 'saas']
   OR categories_strong ?| array['fintech', 'saas']
ORDER BY relevance_score DESC;
```

Obtener estado de campaña:

```sql
SELECT
    c.name,
    c.total_targets,
    COUNT(CASE WHEN t.status = 'sent' THEN 1 END) as sent_count,
    COUNT(CASE WHEN t.status = 'replied' THEN 1 END) as reply_count
FROM outreach_campaigns c
LEFT JOIN outreach_targets t ON t.campaign_id = c.id
WHERE c.id = $campaign_id
GROUP BY c.id;
```

Verificar créditos de usuario:

```sql
SELECT SUM(amount) as current_balance
FROM credit_transactions
WHERE user_id = $user_id;
```

🚨 NOTAS IMPORTANTES PARA RECORDAR

✅ HACER SIEMPRE:

- Usar `project_data` para nueva información (no crear columnas)
- Verificar créditos antes de operaciones costosas
- Guardar prompts y respuestas completas de Gemini
- Usar transacciones para operaciones críticas

❌ NUNCA HACER:

- Modificar directamente las tablas (estructura ya final)
- Asumir que campos JSONB están presentes (pueden ser null)
- Hacer queries sin WHERE user_id (RLS lo bloquea de todos modos)
- Enviar información sensible en logs

🧠 ARQUITECTURA MENTAL:

- Esta base de datos APRENDE de cada interacción
- project_data es como un cerebro que crece
- conversations es la memoria completa
- investors es el directorio de contactos
- Todo lo demás son operaciones encima de estos datos

📞 INTEGRACIÓN CON SERVICIOS EXTERNOS

GEMINI (Google AI):

- Se llama en cada mensaje del usuario
- Respuestas se guardan completas en conversations
- Extracciones se guardan en project_data y ai_extractions

UNIPILE (LinkedIn):

- IDs se guardan en users.unipile_account_id
- Message IDs se guardan en outreach_targets.unipile_message_id
- Webhooks se procesan via webhook_events

STRIPE (Pagos):

- Customer IDs en users.stripe_customer_id
- Subscription IDs en subscriptions.stripe_subscription_id
- Webhooks se procesan via webhook_events

🎉 **¡Esta base de datos está diseñada para escalar desde 1 hasta 100,000 usuarios sin cambios estructurales!**

Última actualización: Al implementar el backend completo