ColNexus Reporte Final
Entrega de Proyecto BD

Daniel Alejandro Duarte Duarte
Mónica Castro Benítez
Juan Felipe Vela Jiménez
José Alejandro Contreras Obregón

Pontificia Universidad Javeriana
Facultad de Ingeniería
Bases de Datos

Presentado a:
Prof. Andrés Oswaldo Calderón Romero

25 de Noviembre de 2025

Índice

1 Introducción

El presente informe detalla el diseño conceptual y lógico para un sistema de base de datos (BD) destinado a recopilar datos de telemetría de juego y experiencia de usuario (UX) generados por el puerto fuente Chocolate-Doom. Este proyecto se enmarca dentro de un esfuerzo de investigación para analizar las dinámicas de juego y el rendimiento en un entorno deliberadamente diseñado para emular la tecnología de los años 90 y generar una propuesta que atienda la consultoría la cual nos fue presentada.

2 Diseño de DB

2.1 Diagrama E-R

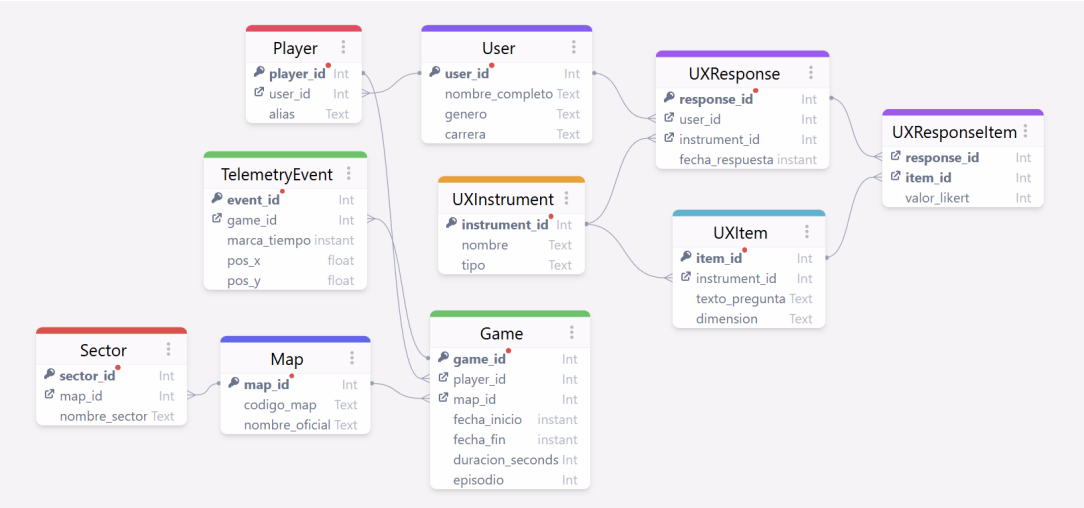

Figura 1: Diagrama Entidad-Relación.

2.2 Esquema Relacional

A continuación se detalla la estructura relacional propuesta:

- Player (player_id, user_id, alias)
- User (user_id, nombre_completo, genero, carrera)
- UXResponse (response_id, user_id, instrument_id, fecha_respuesta)
- UXResponseItem (response_id, item_id, valor_likert)
- UXItem (item_id, instrument_id, texto_pregunta)
- UXInstrument (instrument_id, nombre, tipo)
- TelemetryEvent (event_id, game_id, marca_tiempo, pos_x, pos_y)
- Game (game_id, player_id, map_id, fecha_inicio, fecha_fin, duracion_seconds, episodios)
- Map (map_id, codigo_map, nombre_oficial)
- Sector (sector_id, map_id, nombre_sector)

2.3 Raciocinio

2.3.1 Separación entre telemetría y sesiones

- GameSession actúa como tabla padre que agrupa todos los ticks de telemetría de una partida, lo que facilita consultas por sesión (duración, nivel jugado, jugador).
- TelemetryTick almacena datos por "tic" del juego (posición, acción, número de tick), lo que permite un análisis muy granular de comportamiento durante la partida.

2.3.2 Modelado de niveles (GameLevel)

- Tener una tabla GameLevel (o similar) para los niveles del juego permite normalizar la información de los niveles y evitar duplicación de datos de nivel en cada sesión.
- Además, facilita extender la información del nivel (por ejemplo, nombre, dificultad, diseño) sin repetirlo en cada sesión.

2.3.3 Instrumento UX (cuestionario PENS)

- UXItem representa preguntas o ítems del cuestionario, categorizados por dimensión (autonomía, competencia, etc.), lo que permite mantener la estructura del instrumento en la base de datos.
- UXResponse almacena las respuestas de los jugadores a cada ítem para cada sesión, vinculando la experiencia subjetiva (cuestionario) con la sesión de juego real.

2.3.4 Integridad referencial

- Las claves foráneas (session_id, item_id, level_id) aseguran que los datos de telemetría y respuestas UX siempre estén relacionados con una sesión válida y con ítems válidos del cuestionario.
- Esto previene entradas huérfanas ("orphan rows") y garantiza consistencia entre tablas.

2.3.5 Flexibilidad y escalabilidad

- Al separar los ticks en su propia tabla, el esquema puede escalar para manejar grandes volúmenes de datos sin sobrecargar la tabla de sesiones.
- También permite construir consultas complejas: por ejemplo, cruzar la posición o tipo de acción en ciertos "ticks" con las puntuaciones del cuestionario para explorar correlaciones entre comportamiento de juego y experiencia del usuario.

2.3.6 Análisis relacional enriquecido

- Tener tanto datos objetivos (telemetría) como subjetivos (respuestas PENS) en la base de datos permite análisis de tipo "¿cómo influye la experiencia de juego en el comportamiento?" o "¿existen patrones de acción para jugadores que reportan más autonomía o competencia?".

- Permite usar SQL para agregaciones, filtrados, métricas por sesión o por jugador, posibilitando también la generación de dashboards o reportes.

3 DDL Constraints Appendix

A continuación, una serie de fragmentos del archivo `db_struct.sql` que define tablas, claves primarias y foráneas a modo de ejemplificación para los DDL Constraints.

```sql
CREATE TABLE "User" (
  user_id SERIAL PRIMARY KEY,
  nombre_completo TEXT NOT NULL,
  genero TEXT NOT NULL,
  carrera TEXT NOT NULL
);

CREATE TABLE Player (
  player_id SERIAL PRIMARY KEY,
  user_id INTEGER NOT NULL REFERENCES "User"(user_id),
  alias TEXT NOT NULL
);

CREATE TABLE Map (
  map_id SERIAL PRIMARY KEY,
  codigo_map VARCHAR(50) NOT NULL,
  nombre_oficial VARCHAR(200) NOT NULL
);

CREATE TABLE Game (
  game_id SERIAL PRIMARY KEY,
  player_id INTEGER NOT NULL REFERENCES Player(player_id),
  map_id INTEGER NOT NULL REFERENCES Map(map_id),
  fecha_inicio TIMESTAMP NOT NULL,
  fecha_fin TIMESTAMP NOT NULL,
  duracion_seconds INTEGER NOT NULL,
  episodio INTEGER NOT NULL
);
```

Figura 2: Creación de tablas Inicial Parte 1.

```sql
CREATE TABLE Sector (
  sector_id SERIAL PRIMARY KEY,
  map_id INTEGER NOT NULL REFERENCES Map(map_id),
  nombre_sector VARCHAR(150) NOT NULL
);

CREATE TABLE TelemetryEvent (
  event_id SERIAL PRIMARY KEY,
  game_id INTEGER NOT NULL REFERENCES Game(game_id),
  marca_tiempo TIMESTAMP NOT NULL,
  pos_x NUMERIC NOT NULL,
  pos_y NUMERIC NOT NULL
);

CREATE TABLE UXInstrument (
  instrument_id SERIAL PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  tipo VARCHAR(50) NOT NULL
);

CREATE TABLE UXItem (
  item_id SERIAL PRIMARY KEY,
  instrument_id INTEGER NOT NULL REFERENCES UXInstrument(instrument_id),
  texto_pregunta TEXT NOT NULL,
  dimension VARCHAR(100)
);

CREATE TABLE UXResponse (
  response_id SERIAL PRIMARY KEY,
  user_id INTEGER NOT NULL REFERENCES "User"(user_id),
  instrument_id INTEGER NOT NULL REFERENCES UXInstrument(instrument_id),
  fecha_respuesta TIMESTAMP NOT NULL,
  respuestas_json TEXT DEFAULT '{}'
);

CREATE TABLE UXResponseItem (
  response_id INTEGER NOT NULL REFERENCES UXResponse(response_id),
  item_id INTEGER NOT NULL REFERENCES UXItem(item_id),
  valor_likert INTEGER NOT NULL CHECK (valor_likert BETWEEN 1 AND 7),
  PRIMARY KEY (response_id, item_id)
);
```

Figura 3: Creación de tablas Inicial Parte 2.

4 Descripción ETL

El proceso ETL implementado para ColNexus sigue las recomendaciones del enunciado oficial del proyecto, las cuales establecen que la ingestión debe seguir un flujo *staging* → *validación* → *carga en tablas core*. En este repositorio, dicho proceso se encuentra implementado principalmente en el archivo loader_script.cxx dentro de la carpeta Code.

A continuación se describe el funcionamiento detallado del pipeline ETL, basado exclusivamente en los archivos del repositorio:

4.1 Extract (Extracción de datos crudos)

El motor modificado de Chocolate-Doom genera un archivo de telemetría por sesión de juego utilizando la redirección estándar:

```
chocolate-doom -iwad DOOM.WAD > Data/NombreJugador_IDPartida.txt
```

El archivo es un **log secuencial por tic** que contiene información mínima utilizada por el loader:

- Timestamp del motor (formato YYYY-MM-DD HH:MM:SS)
- Tic del juego
- Posición X
- Posición Y

Adicionalmente, el log incluye líneas meta como:

```
When: 2025-11-22 14:35:10 Episode: 1 Map: 1
```

Las cuales permiten al sistema:

- Identificar la fecha de inicio.
- Determinar episodio.
- Determinar el mapa.
- Calcular la duración total de la partida.

El archivo crudo se deposita en la carpeta `Data/`.

4.2 Transform (Validación, limpieza y normalización)

El archivo `loader_script.cxx` implementa la etapa de transformación de la siguiente forma:

a. Detección de metadatos de sesión: El script utiliza expresiones regulares (`std::regex`) para extraer:

- Fecha de inicio.
- Episodio jugado.
- Número de mapa.
- Primer y último timestamp.
- Duración calculada en segundos.

b. Lectura y validación línea por línea: Cada línea de telemetría se compara con el patrón:

```
YYYY-MM-DD HH:MM:SS tic pos_x pos_y
```

Se valida que:

- El timestamp sea correcto.
- El tic sea numérico.
- Las coordenadas estén en rango válido.
- La fila no esté vacía ni truncada.

c. Normalización: El ETL convierte:

- timestamps tipo `2025-11-22T14:35:12` → `2025-11-22 14:35:12`.
- Coordenadas extraídas del motor → valores numéricos SQL.
- Nombres del jugador → alias definidos en la ejecución del programa.

d. Construcción incremental del SQL de carga: El loader no carga directamente en la base de datos. Genera un archivo estándar: `Code/data_loader.sql`. Este archivo contiene:

```
1. INSERT INTO User
2. INSERT INTO Player
3. INSERT INTO Map
4. INSERT INTO Game
5. INSERT INTO Sector (placeholder único para el mapa actual)
6. INSERT múltiples en TelemetryEvent
7. INSERT de UXInstrument y UXResponse (estructura vacía por defecto)
```

La estructura generada respeta las claves foráneas definidas en `db_struct.sql`.

4.3 Load (Carga en tablas core de PostgreSQL)

Una vez generado `data_loader.sql`, la carga se ejecuta manualmente:

```
psql -U usuario -d colnexus -f Code/db_struct.sql
psql -U usuario -d colnexus -f Code/ux_instrument_pens.sql
psql -U usuario -d colnexus -f Code/data_loader.sql
psql -U usuario -d colnexus -f Code/index_creation.sql
```

4.4 Resumen:

La ejecución realiza los siguientes aspectos:

1. `db_struct.sql` crea el esquema relacional (User, Player, Game, Map, Sector, TelemetryEvent, UX...).
2. `ux_instrument_pens.sql` inserta los 21 items del cuestionario PENS.
3. `data_loader.sql` ingesta la sesión de telemetría procesada.
4. `index_creation.sql` crea los índices recomendados para análisis:
   - Índices compuestos (game_id, player_id, tic).
   - índices por sector/mapa.
   - índice espacial GiST basado en (pos_x, pos_y).
5. views.sql

Este orden garantiza integridad referencial y rendimiento óptimo.

5 Ejecuciones

5.1 Definición de Vistas y Optimizaciones

Para facilitar el análisis de datos y optimizar consultas frecuentes, se implementaron las siguientes vistas y vistas materializadas. A continuación se presenta el código de definición y la justificación de su implementación:

```
-- Limpieza previa
DROP VIEW IF EXISTS View_Player_Demographics CASCADE;
DROP VIEW IF EXISTS View_UX_Dimension_Scores CASCADE;
DROP MATERIALIZED VIEW IF EXISTS MatView_Session_Stats CASCADE;

-- 1. Vista de Demografia de Jugadores
CREATE VIEW View_Player_Demographics AS
```

```sql
SELECT
    p.player_id,
    u.user_id, -- Columna clave para JOINS con UX
    p.alias,
    u.nombre_completo,
    u.genero,
    u.carrera
FROM Player p
JOIN "User" u ON p.user_id = u.user_id;

-- 2. Vista de Scores UX Agrupados
CREATE VIEW View_UX_Dimension_Scores AS
SELECT
    r.user_id,
    r.response_id,
    i.dimension,
    ROUND(AVG(ri.valor_likert), 2) as promedio_dimension
FROM UXResponse r
JOIN UXResponseItem ri ON r.response_id = ri.response_id
JOIN UXItem i ON ri.item_id = i.item_id
GROUP BY r.user_id, r.response_id, i.dimension;

-- 3. Vista Materializada de Estadisticas de Sesion
CREATE MATERIALIZED VIEW MatView_Session_Stats AS
SELECT
    g.game_id,
    m.nombre_oficial AS mapa,
    g.duracion_seconds,
    COUNT(t.event_id) AS total_tics_registrados,
    MIN(t.pos_x) AS min_x,
    MAX(t.pos_x) AS max_x,
    MIN(t.pos_y) AS min_y,
    MAX(t.pos_y) AS max_y
FROM Game g
JOIN Map m ON g.map_id = m.map_id
JOIN TelemetryEvent t ON g.game_id = t.game_id
GROUP BY g.game_id, m.nombre_oficial, g.duracion_seconds;

CREATE INDEX idx_matview_game ON MatView_Session_Stats(game_id);
```

5.1.1 Justificación de las Vistas

- **View_Player_Demographics:** Esta vista abstrae la complejidad de unir las tablas ‘Player‘ y ‘User‘. Dado que múltiples consultas requieren filtrar o agrupar por atributos demográficos (carrera, género) junto con el alias del juego, esta vista simplifica la escritura de queries y garantiza consistencia en el acceso a la identidad del jugador.
- **View_UX_Dimension_Scores:** El cuestionario PENS se compone de múltiples ítems que pertenecen a dimensiones psicológicas específicas (autonomía, competencia, etc.). Esta vista pre-calcula el promedio de la escala Likert agrupado por dimensión para cada respuesta. Esto permite realizar análisis de correlación entre telemetría y psicología sin tener que recalcular los promedios en cada consulta.
- **MatView_Session_Stats:** Esta es una **Vista Materializada** diseñada por razones de rendimiento. La tabla ‘TelemetryEvent‘ contiene millones de registros (tics

de juego). Calcular agregaciones como el conteo total de tics o los límites espaciales (min/max X, Y) en tiempo real es computacionalmente costoso. Al materializar estos datos, el resultado se almacena físicamente, permitiendo reportes instantáneos sobre las estadísticas generales de las sesiones sin volver a escanear la tabla gigante de eventos.

5.2 Queries y Resultados

A continuación se presentan las consultas SQL desarrolladas para el análisis de los datos. Se incluyen dos versiones para cada análisis principal: la consulta compleja original (con *Joins* explícitos) y la consulta simplificada implementando las *Vistas* definidas anteriormente.

1. Average duration of game sessions per map.

a. Query Inicial

```sql
SELECT
    m.map_id,
    m.nombre_oficial,
    ROUND(AVG(g.duracion_seconds)::numeric, 2) AS
   duracion_promedio_segundos
FROM Game g
JOIN Map m ON g.map_id = m.map_id
GROUP BY m.map_id, m.nombre_oficial
ORDER BY duracion_promedio_segundos DESC;
```

	map_id [PK] integer	nombre_oficial character varying (200)	duracion_promedio_segundos numeric
1	3	E1M3: Toxin Refinery	1459.00
2	4	E1M4: Command Control	1417.00
3	5	E1M5: Phobos Lab	1117.00

Figura 4: Query 1

b. Query con View (MatView_Session_Stats)

```sql
SELECT
    mapa AS nombre_oficial,
    ROUND(AVG(duracion_seconds)::numeric, 2) AS
   duracion_promedio_segundos
FROM MatView_Session_Stats
GROUP BY mapa
ORDER BY duracion_promedio_segundos DESC;
```

	nombre_oficial character varying (200)	duracion_promedio_segundos numeric
1	E1M3: Toxin Refinery	1459.00
2	E1M4: Command Control	1417.00
3	E1M5: Phobos Lab	1117.00

Figura 5: Query 1

2. Players with the highest average proximity.

a. Query Inicial

```sql
WITH paired_events AS (
    SELECT
        g1.player_id AS player_a,
        g2.player_id AS player_b,
        g1.map_id,
        SQRT(POWER(te1.pos_x - te2.pos_x, 2) + POWER(te1.pos_y - te2.pos_y, 2)) AS distancia
    FROM TelemetryEvent te1
    JOIN Game g1 ON te1.game_id = g1.game_id
    JOIN TelemetryEvent te2 ON te1.game_id <> te2.game_id
    JOIN Game g2 ON te2.game_id = g2.game_id
    WHERE g1.map_id = g2.map_id
      AND g1.player_id < g2.player_id
)
SELECT
    player_a,
    player_b,
    ROUND(AVG(distancia)::numeric, 2) AS promedio_proximidad
FROM paired_events
WHERE distancia <= 200
GROUP BY player_a, player_b
ORDER BY promedio_proximidad ASC;
```

	player_a integer	player_b integer	promedio_proximidad numeric
1	3	4	4.51

Figura 6: Query 2

b. Query con View (View_Player_Demographics)

```sql
WITH paired_events AS (
    SELECT
        g1.player_id AS id_a,
        g2.player_id AS id_b,
        g1.map_id,
        SQRT(POWER(te1.pos_x - te2.pos_x, 2) + POWER(te1.pos_y - te2.
    pos_y, 2)) AS distancia
    FROM TelemetryEvent te1
    JOIN Game g1 ON te1.game_id = g1.game_id
    JOIN TelemetryEvent te2 ON te1.game_id <> te2.game_id
    JOIN Game g2 ON te2.game_id = g2.game_id
    WHERE g1.map_id = g2.map_id AND g1.player_id < g2.player_id
)
SELECT
    vp1.alias AS jugador_a,
    vp2.alias AS jugador_b,
    ROUND(AVG(pe.distancia)::numeric, 2) AS promedio_proximidad
FROM paired_events pe
JOIN View_Player_Demographics vp1 ON pe.id_a = vp1.player_id
JOIN View_Player_Demographics vp2 ON pe.id_b = vp2.player_id
WHERE pe.distancia <= 200
GROUP BY vp1.alias, vp2.alias
ORDER BY promedio_proximidad ASC;
```

	jugador_a text	jugador_b text	promedio_proximidad numeric
1	mysticfriday	elito1322	4.51

Figura 7: Query 2

3. Shortest and longest trajectory distances per player.

a. Query Inicial

```sql
WITH distancia_tramos AS (
    SELECT
        g.player_id,
        te1.game_id,
        SQRT(
            POWER(te2.pos_x - te1.pos_x, 2) +
            POWER(te2.pos_y - te1.pos_y, 2)
        ) AS distancia_parcial
    FROM TelemetryEvent te1
    JOIN TelemetryEvent te2
        ON te1.game_id = te2.game_id
        AND te2.marca_tiempo = (
            SELECT MIN(te3.marca_tiempo)
            FROM TelemetryEvent te3
```

```
            WHERE te3.game_id = te1.game_id
              AND te3.marca_tiempo > te1.marca_tiempo
        )
    JOIN Game g ON te1.game_id = g.game_id
),
trayectorias AS (
    SELECT
        player_id,
        game_id,
        SUM(distancia_parcial) AS distancia_total
    FROM distancia_tramos
    GROUP BY player_id, game_id
)
SELECT
    player_id,
    ROUND(distancia_total::numeric, 2) AS distancia_total
FROM trayectorias
ORDER BY distancia_total DESC;
```

	player_id integer	distancia_total numeric
1	3	483804.79
2	3	230639.34
3	4	187786.78
4	3	178268.81
5	3	168005.63
6	3	162146.73
7	4	141367.50
8	3	84621.11
9	4	72239.45
10	3	55499.88
11	3	50109.00
12	4	47334.64
13	4	45179.69

Figura 8: Query 3

b. Query con View (View_Player_Demographics)

```sql
WITH distancia_tramos AS (
    SELECT
        g.player_id,
        te1.game_id,
        SQRT(
            POWER(te2.pos_x - te1.pos_x, 2) +
            POWER(te2.pos_y - te1.pos_y, 2)
        ) AS distancia_parcial
    FROM TelemetryEvent te1
    JOIN TelemetryEvent te2
        ON te1.game_id = te2.game_id
        AND te2.marca_tiempo = (
            SELECT MIN(te3.marca_tiempo)
            FROM TelemetryEvent te3
            WHERE te3.game_id = te1.game_id
              AND te3.marca_tiempo > te1.marca_tiempo
        )
    JOIN Game g ON te1.game_id = g.game_id
),
trayectorias AS (
    SELECT
        player_id,
        game_id,
        SUM(distancia_parcial) AS distancia_total
    FROM distancia_tramos
    GROUP BY player_id, game_id
)
SELECT
    vp.alias,
    ROUND(t.distancia_total::numeric, 2) AS distancia_total
FROM trayectorias t
JOIN View_Player_Demographics vp ON t.player_id = vp.player_id
ORDER BY distancia_total DESC;
```

	alias text	distancia_total numeric
1	mysticfriday	483804.79
2	mysticfriday	230639.34
3	elito1322	187786.78
4	mysticfriday	178268.81
5	mysticfriday	168005.63
6	mysticfriday	162146.73
7	elito1322	141367.50
8	mysticfriday	84621.11
9	elito1322	72239.45
10	mysticfriday	55499.88
11	mysticfriday	50109.00
12	elito1322	47334.64
13	elito1322	45179.69

Figura 9: Query 3

4. UX responses for players with above-average trajectory.

a. Query Inicial

```
WITH duraciones AS (
    SELECT
        g.player_id,
        g.game_id,
        (
            EXTRACT(DAY FROM (MAX(te.marca_tiempo) - MIN(te.
marca_tiempo))) * 86400 +
            EXTRACT(HOUR FROM (MAX(te.marca_tiempo) - MIN(te.
marca_tiempo))) * 3600 +
            EXTRACT(MINUTE FROM (MAX(te.marca_tiempo) - MIN(te.
marca_tiempo))) * 60 +
            EXTRACT(SECOND FROM (MAX(te.marca_tiempo) - MIN(te.
marca_tiempo)))
        ) AS duracion_segundos
    FROM TelemetryEvent te
    JOIN Game g ON te.game_id = g.game_id
    GROUP BY g.player_id, g.game_id
```

```
),
promedio AS (
    SELECT AVG(duracion_segundos) AS duracion_promedio
    FROM duraciones
)
SELECT
    d.player_id,
    ROUND(d.duracion_segundos::numeric, 2) AS duracion_total,
    ui.dimension,
    AVG(uri.valor_likert) AS promedio_ux
FROM duraciones d
JOIN promedio p ON d.duracion_segundos > p.duracion_promedio
JOIN UXResponse ux ON ux.user_id = d.player_id
JOIN UXResponseItem uri ON uri.response_id = ux.response_id
JOIN UXItem ui ON uri.item_id = ui.item_id
GROUP BY d.player_id, duracion_total, ui.dimension
ORDER BY duracion_total DESC;
```

	player_id integer	duracion_total numeric	dimension character varying (100)	promedio_ux numeric
1	3	3455.00	AutonomÃa	6.3333333333333333
2	3	3455.00	Competencia	6.6666666666666667
3	3	3455.00	Controles intuitivos	6.6666666666666667
4	3	3455.00	RelaciÃ³n	6.0000000000000000
5	3	2324.00	AutonomÃa	6.3333333333333333
6	3	2324.00	Competencia	6.6666666666666667
7	3	2324.00	Controles intuitivos	6.6666666666666667
8	3	2324.00	RelaciÃ³n	6.0000000000000000
9	4	2236.00	AutonomÃa	6.3333333333333333
10	4	2236.00	Competencia	5.6666666666666667
11	4	2236.00	Controles intuitivos	6.6666666666666667
12	4	2236.00	RelaciÃ³n	5.0000000000000000
13	3	1459.00	AutonomÃa	6.3333333333333333
14	3	1459.00	Competencia	6.6666666666666667
15	3	1459.00	Controles intuitivos	6.6666666666666667
16	3	1459.00	RelaciÃ³n	6.0000000000000000
17	3	1417.00	AutonomÃa	6.3333333333333333
18	3	1417.00	Competencia	6.6666666666666667
19	3	1417.00	Controles intuitivos	6.6666666666666667
20	3	1417.00	RelaciÃ³n	6.0000000000000000
21	3	1273.00	AutonomÃa	6.3333333333333333

Total rows: 24 | Query complete 00:00:00.237

Figura 10: Query 4

5. Most visited sector (hotspot) per episode and map.

a. Query Inicial

```
WITH sectorized AS (
    SELECT
        g.episodio,
        g.map_id,
        FLOOR(te.pos_x / 250) AS sector_x,
        FLOOR(te.pos_y / 250) AS sector_y
    FROM TelemetryEvent te
    JOIN Game g ON te.game_id = g.game_id
)
SELECT
    episodio,
    map_id,
```

```
    sector_x,
    sector_y,
    COUNT(*) AS visitas
FROM sectorized
GROUP BY episodio, map_id, sector_x, sector_y
ORDER BY visitas DESC
LIMIT 10;
```

	episodio integer	map_id integer	sector_x numeric	sector_y numeric	visitas bigint
1	1	5	-1	-3	209
2	1	5	3	1	192
3	1	5	-1	0	114
4	1	5	1	1	111
5	1	5	-2	-1	110
6	1	5	0	0	108
7	1	5	5	1	108
8	1	5	-8	-11	107
9	1	5	0	1	102
10	1	5	-2	3	101

Figura 11: Query 5

6. Number of Tics Where Players Were Together in a Sector.

a. Query Inicial

```
WITH sectorized AS (
    SELECT
        g.player_id,
        te.game_id,
        FLOOR(te.pos_x / 250) AS sector_x,
        FLOOR(te.pos_y / 250) AS sector_y
    FROM TelemetryEvent te
    JOIN Game g ON te.game_id = g.game_id
)
SELECT
    a.player_id AS jugador_a,
    b.player_id AS jugador_b,
    a.sector_x,
    a.sector_y,
    COUNT(*) AS coincidencias_sector
FROM sectorized a
```

```sql
JOIN sectorized b
ON a.sector_x = b.sector_x
   AND a.sector_y = b.sector_y
   AND a.player_id < b.player_id
GROUP BY jugador_a, jugador_b, a.sector_x, a.sector_y
ORDER BY coincidencias_sector DESC;
```

	jugador_a integer	jugador_b integer	sector_x numeric	sector_y numeric	coincidencias_sector bigint
1	3	4	-8	-11	8360
2	3	4	-6	-13	5394
3	3	4	-1	-3	4830
4	3	4	3	1	4800
5	3	4	-7	-13	4371
6	3	4	1	1	3618
7	3	4	2	1	3321
8	3	4	0	1	3036
9	3	4	5	1	2835
10	3	4	-9	-8	2730
11	3	4	0	2	2680
12	3	4	-8	-9	2660
13	3	4	-1	0	2576
14	3	4	-9	-9	2550
15	3	4	-5	-11	2144
16	3	4	-8	-10	2112
17	3	4	-9	-7	1890
18	3	4	4	1	1876
19	3	4	6	0	1848
20	3	4	-8	-8	1690
21	3	4	-8	-7	1682

Total rows: 198 Query complete 00:00:00.713

Figura 12: Query 6

b. Query con View (View_Player_Demographics)

```sql
WITH sectorized AS (
    SELECT
        g.player_id, te.game_id,
        FLOOR(te.pos_x / 250) AS sector_x,
        FLOOR(te.pos_y / 250) AS sector_y
```

```
    FROM TelemetryEvent te
    JOIN Game g ON te.game_id = g.game_id
)
SELECT
    vp1.alias AS jugador_a,
    vp2.alias AS jugador_b,
    a.sector_x, a.sector_y,
    COUNT(*) AS coincidencias_sector
FROM sectorized a
JOIN sectorized b ON a.sector_x = b.sector_x AND a.sector_y = b.
   sector_y AND a.player_id < b.player_id
JOIN View_Player_Demographics vp1 ON a.player_id = vp1.player_id
JOIN View_Player_Demographics vp2 ON b.player_id = vp2.player_id
GROUP BY vp1.alias, vp2.alias, a.sector_x, a.sector_y
ORDER BY coincidencias_sector DESC;
```

	jugador_a text	jugador_b text	sector_x numeric	sector_y numeric	coincidencias_sector bigint
1	mysticfriday	elito1322	-8	-11	8360
2	mysticfriday	elito1322	-6	-13	5394
3	mysticfriday	elito1322	-1	-3	4830
4	mysticfriday	elito1322	3	1	4800
5	mysticfriday	elito1322	-7	-13	4371
6	mysticfriday	elito1322	1	1	3618
7	mysticfriday	elito1322	2	1	3321
8	mysticfriday	elito1322	0	1	3036
9	mysticfriday	elito1322	5	1	2835
10	mysticfriday	elito1322	-9	-8	2730
11	mysticfriday	elito1322	0	2	2680
12	mysticfriday	elito1322	-8	-9	2660
13	mysticfriday	elito1322	-1	0	2576
14	mysticfriday	elito1322	-9	-9	2550
15	mysticfriday	elito1322	-5	-11	2144
16	mysticfriday	elito1322	-8	-10	2112
17	mysticfriday	elito1322	-9	-7	1890
18	mysticfriday	elito1322	4	1	1876

Total rows: 198 | Query complete 00:00:00.861

Figura 13: Query 6

7. Puntuación UX Promedio para Jugadores con la Trayectoria Más Corta

por Episodio.

a. Query Inicial

```
WITH eventos_ordenados AS (
    SELECT
        g.episodio,
        g.player_id,
        te1.pos_x,
        te1.pos_y,
        te2.pos_x AS next_x,
        te2.pos_y AS next_y
    FROM telemetryevent te1
    JOIN game g ON te1.game_id = g.game_id
    LEFT JOIN telemetryevent te2
        ON te1.game_id = te2.game_id
        AND te2.marca_tiempo = (
            SELECT MIN(te3.marca_tiempo)
            FROM telemetryevent te3
            WHERE te3.game_id = te1.game_id
              AND te3.marca_tiempo > te1.marca_tiempo
        )
),
distancias AS (
    SELECT
        episodio,
        player_id,
        SUM(
            SQRT(
                POWER(next_x - pos_x, 2) +
                POWER(next_y - pos_y, 2)
            )
        ) AS total_distancia
    FROM eventos_ordenados
    WHERE next_x IS NOT NULL
    GROUP BY episodio, player_id
),
jugadores_validos AS (
    SELECT DISTINCT r.user_id
    FROM uxresponse r
    JOIN uxresponseitem ri ON ri.response_id = r.response_id
),
minimos AS (
    SELECT DISTINCT ON (d.episodio)
        d.episodio,
        d.player_id,
        d.total_distancia
    FROM distancias d
    JOIN jugadores_validos j ON j.user_id = d.player_id
    ORDER BY d.episodio, d.total_distancia ASC
)
SELECT
    m.episodio,
    m.player_id,
    ROUND(AVG(ri.valor_likert),2) AS promedio_ux
FROM minimos m
JOIN uxresponse r ON r.user_id = m.player_id
JOIN uxresponseitem ri ON ri.response_id = r.response_id
GROUP BY m.episodio, m.player_id
```

```
ORDER BY m.episodio;
```

	episodio integer	player_id integer	promedio_ux numeric
1	1	4	5.92

Figura 14: Query 7

b. Query con View (View_UX_Dimension_Scores)

```
WITH eventos_ordenados AS (
    SELECT
        g.episodio, g.player_id, te1.pos_x, te1.pos_y, te2.pos_x AS
    next_x, te2.pos_y AS next_y
    FROM telemetryevent te1
    JOIN game g ON te1.game_id = g.game_id
    LEFT JOIN telemetryevent te2 ON te1.game_id = te2.game_id
        AND te2.marca_tiempo = (SELECT MIN(t3.marca_tiempo) FROM
    telemetryevent t3 WHERE t3.game_id = te1.game_id AND t3.marca_tiempo
    > te1.marca_tiempo)
),
distancias AS (
    SELECT episodio, player_id, SUM(SQRT(POWER(next_x - pos_x, 2) +
    POWER(next_y - pos_y, 2))) AS total_distancia
    FROM eventos_ordenados WHERE next_x IS NOT NULL GROUP BY episodio,
    player_id
),
minimos AS (
    SELECT DISTINCT ON (d.episodio) d.episodio, d.player_id, d.
    total_distancia
    FROM distancias d ORDER BY d.episodio, d.total_distancia ASC
)
SELECT
    m.episodio,
    vp.alias,
    vus.dimension,
    vus.promedio_dimension AS promedio_ux
FROM minimos m
JOIN View_Player_Demographics vp ON m.player_id = vp.player_id
JOIN View_UX_Dimension_Scores vus ON vp.user_id = vus.user_id
ORDER BY m.episodio;
```

	episodio integer	alias text	dimension character varying (100)	promedio_ux numeric
1	1	elito1322	RelaciÃ³n	5.00
2	1	elito1322	Controles intuitivos	6.67
3	1	elito1322	Competencia	5.67
4	1	elito1322	AutonomÃa	6.33

Figura 15: Query 7

8. Distancia Total Recorrida y Velocidad Promedio por Jugador.

a. Query Inicial

```
WITH puntos AS (
    SELECT
        g.game_id,
        g.player_id,
        g.episodio,
        te1.marca_tiempo,
        te1.pos_x,
        te1.pos_y,
        te2.pos_x AS next_x,
        te2.pos_y AS next_y
    FROM TelemetryEvent te1
    JOIN Game g ON te1.game_id = g.game_id
    LEFT JOIN TelemetryEvent te2 ON
        te1.game_id = te2.game_id
        AND te2.marca_tiempo = (
            SELECT MIN(te3.marca_tiempo)
            FROM TelemetryEvent te3
            WHERE te3.game_id = te1.game_id
              AND te3.marca_tiempo > te1.marca_tiempo
        )
),
distancias AS (
    SELECT
        episodio,
        player_id,
        marca_tiempo,
        SQRT(POWER(next_x - pos_x, 2) + POWER(next_y - pos_y, 2)) AS
   distancia
    FROM puntos
    WHERE next_x IS NOT NULL
),
resumen AS (
    SELECT
        episodio,
        player_id,
        SUM(distancia) AS distancia_total,
        MIN(marca_tiempo) AS inicio,
```

```
        MAX(marca_tiempo) AS fin
    FROM distancias
    GROUP BY episodio, player_id
)
SELECT
    episodio,
    player_id,
    ROUND(distancia_total::numeric, 2) AS distancia_total,
    (
        date_part('day',    justify_hours(fin - inicio)) * 86400 +
        date_part('hour',   justify_hours(fin - inicio)) * 3600 +
        date_part('minute',justify_hours(fin - inicio)) * 60 +
        date_part('second',justify_hours(fin - inicio))
    ) AS tiempo_total_segundos,
    ROUND(
        distancia_total /
        NULLIF(
            (
                date_part('day',    justify_hours(fin - inicio)) * 86400 +
                date_part('hour',   justify_hours(fin - inicio)) * 3600 +
                date_part('minute',justify_hours(fin - inicio)) * 60 +
                date_part('second',justify_hours(fin - inicio))
            ),
            0
        )::numeric,
        2
    ) AS velocidad_promedio
FROM resumen
ORDER BY episodio, velocidad_promedio DESC;
```

	episodio integer	player_id integer	distancia_total numeric	tiempo_total_segundos numeric	velocidad_promedio numeric
1	1	4	493908.06	10636.000000	46.44
2	1	3	1413095.29	94957.000000	14.88

Figura 16: Query 8

b. Query con View (View_Player_Demographics)

```
WITH puntos AS (
    SELECT
        g.game_id, g.player_id, g.episodio, te1.marca_tiempo, te1.pos_x
    , te1.pos_y, te2.pos_x AS next_x, te2.pos_y AS next_y
    FROM TelemetryEvent te1
    JOIN Game g ON te1.game_id = g.game_id
    LEFT JOIN TelemetryEvent te2 ON te1.game_id = te2.game_id
        AND te2.marca_tiempo = (SELECT MIN(t3.marca_tiempo) FROM
    TelemetryEvent t3 WHERE t3.game_id = te1.game_id AND t3.marca_tiempo
    > te1.marca_tiempo)
),
```

```sql
distancias AS (
    SELECT episodio, player_id, marca_tiempo, SQRT(POWER(next_x - pos_x, 2) + POWER(next_y - pos_y, 2)) AS distancia
    FROM puntos WHERE next_x IS NOT NULL
),
resumen AS (
    SELECT episodio, player_id, SUM(distancia) AS distancia_total, MIN(marca_tiempo) AS inicio, MAX(marca_tiempo) AS fin
    FROM distancias GROUP BY episodio, player_id
)
SELECT
    r.episodio,
    vp.alias,
    ROUND(r.distancia_total::numeric, 2) AS distancia_total,
    (
        date_part('day',    justify_hours(fin - inicio)) * 86400 +
        date_part('hour',   justify_hours(fin - inicio)) * 3600 +
        date_part('minute',justify_hours(fin - inicio)) * 60 +
        date_part('second',justify_hours(fin - inicio))
    ) AS tiempo_total_segundos,
    ROUND(
        r.distancia_total /
        NULLIF((
            date_part('day',    justify_hours(fin - inicio)) * 86400 +
            date_part('hour',   justify_hours(fin - inicio)) * 3600 +
            date_part('minute',justify_hours(fin - inicio)) * 60 +
            date_part('second',justify_hours(fin - inicio))
        ), 0)::numeric, 2
    ) AS velocidad_promedio
FROM resumen r
JOIN View_Player_Demographics vp ON r.player_id = vp.player_id
ORDER BY r.episodio, velocidad_promedio DESC;
```

	episodio integer	alias text	distancia_total numeric	tiempo_total_segundos numeric	velocidad_promedio numeric
1	1	elito1322	493908.06	10636.000000	46.44
2	1	mysticfriday	1413095.29	94957.000000	14.88

Figura 17: Query 8

5.3 Index Analysis

Query Comparison: Q1 - Q8

	Before Indexation (ms)	After Indexation (ms)
Q1	0.217	0.059
Q2	0.151	0.077
Q3	0.106	0.070
Q4	1.144	0.189
Q5	0.145	0.065
Q6	0.094	0.088
Q7	0.519	0.389
Q8	0.109	0.088

Cuadro 1: Comparación de tiempos de ejecución antes y después de indexar.

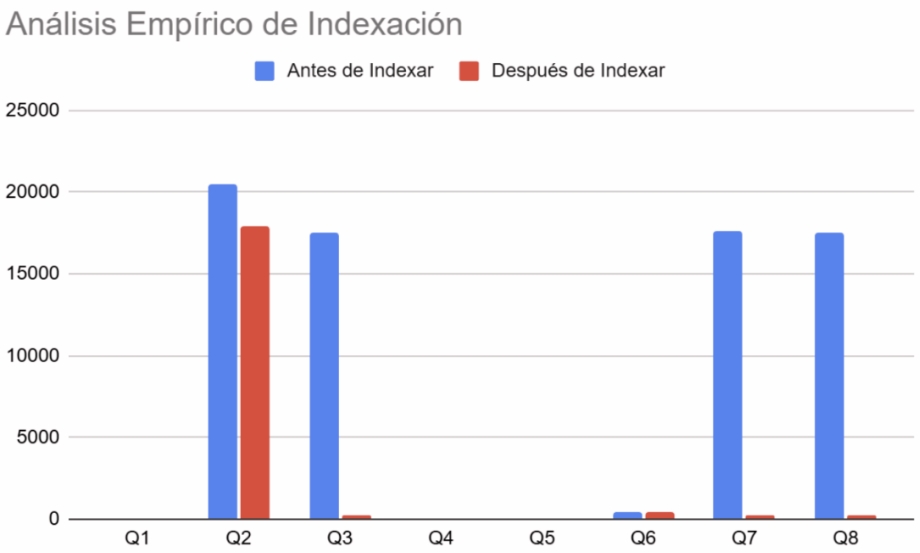

Figura 18: Queries: Empiric Analysis.

Un análisis de los resultados demuestra conclusivamente la efectividad de la indexación a fin de incrementar la eficiencia y velocidad de la base de datos. La tendencia de decrementó en tiempos de ejecución se mantuvo a través de los ocho queries y se manifestó en una reducción de 74.12 puntos porcentuales. La diferencia fue más pronunciada en el caso de los queries tres, siete y ocho, con una reducción del 98.79 %. Esto se debe a que dichos queries trabajan con mayor cantidad de entidades indexadas, haciendo más evidente su impacto.

6 Nota Ética

Este proyecto analiza telemetría de juego proveniente del motor modificado Chocolate-Doom con el objetivo de diseñar e implementar una base de datos relacional que permita la ingestión, validación y análisis de datos de juego por tick, junto con respuestas del cuestionario de experiencia del jugador (PENS: Player Experience of Need Satisfaction). Los datos incluyen archivos de telemetría de sesiones de juego (TSV) y resultados del instrumento PENS suministrados por participantes voluntarios.

- **Privacidad y anonimato:** No se almacena información personal identificable. Cualquier nombre presente en los archivos originales se reemplaza por códigos anónimos.
- **Minimización:** Solo se recopila la información necesaria para el análisis académico del comportamiento de juego.
- **Responsabilidad:** El equipo se compromete a manejar todos los datos de forma responsable, garantizando privacidad, transparencia y respeto por los participantes.

7 Diccionario de Datos

En la Tabla 2 se describe la estructura de la base de datos implementada.

Tabla	Atributo	Tipo	Restricciones / Comentarios
User	user_id	SERIAL	PK. Identificador único del usuario.
	nombre_completo	TEXT	No nulo.
	genero	TEXT	Control semántico.
	carrera	TEXT	Información académica.
Player	player_id	SERIAL	PK. Identificador del alias.
	user_id	INTEGER	FK - User.
	alias	TEXT	Alias único.
Map	map_id	SERIAL	PK. Identificador del mapa.
	codigo_map	VARCHAR	Código interno del motor.
	nombre_oficial	VARCHAR	Nombre descriptivo.
Game	game_id	SERIAL	PK. Identificador de sesión.
	player_id	INTEGER	FK - Player.
	map_id	INTEGER	FK - Map.
	duracion	INTEGER	Duración en segundos.
Telemetry Event	event_id	SERIAL	PK. Identificador del evento.
	game_id	INTEGER	FK - Game.
	marca_tiempo	TIMESTAMP	Orden temporal.
	pos_x / pos_y	NUMERIC	Coordenadas espaciales.
UXItem	item_id	SERIAL	PK. Identificador del ítem.
	texto	TEXT	Texto de la pregunta.
	dimension	VARCHAR	Dimensión psicológica.
UXResponseItem	response_id	INTEGER	FK - UXResponse.
	item_id	INTEGER	FK UXItem.
	valor_likert	INTEGER	CHECK (1-7).

Cuadro 2: Diccionario de Datos del Sistema ColNexus

8 Conclusiones

ColNexus logró integrar datos de telemetría de DOOM y respuestas del cuestionario PENS dentro de una base de datos relacional diseñada para garantizar consistencia, integridad y capacidad analítica. A través de un proceso ETL propio, los datos crudos fueron limpiados, normalizados y transformados en un conjunto estructurado apto para consultas tanto descriptivas como exploratorias. El sistema resultante permite vincular métricas objetivas de comportamiento en el juego con evaluaciones subjetivas de la experiencia del jugador, habilitando análisis más completos sobre patrones de interacción y rendimiento. El proyecto demuestra la eficacia de combinar técnicas de ingeniería de datos y modelado relacional en un entorno académico, y deja una base sólida para futuras ampliaciones orientadas al análisis avanzado o la visualización de resultados.

Referencias

[1] A. O. Calderón Romero, "Project: Designing a Telemetry and UX Database for Chocolate-Doom Research (DBS Semester Project Brief)," Pontificia Universidad Javeriana, Bogotá, Guía de Proyecto, Nov. 2025.