SpaceSim Database

Yevhen Savchuk, Denys Kyrychenko

1. Podstawowe założenia projektu

Cel:
Celem projektu jest stworzenie bazy danych, która obsługuje świat gry symulującej, eksplorację i handel w kosmosie (SpaceSim). Baza gromadzi informacje o systemach gwiezdnych, planetach, stacjach, statkach (zarówno graczy, jak i NPC), transakcjach handlowych, misjach, frakcjach i innych elementach istotnych dla rozgrywki.

Główne założenia i możliwości:

- Obsługa wielu systemów gwiezdnych wraz z ich pozycjami (koordynaty 3D).
- Przechowywanie planet i stacji w konkretnych systemach.
- Rozróżnienie statków graczy (PlayerShips) oraz statków NPC (NPCShips), z dziedziczeniem od tabeli bazowej Ships.
- Zarządzanie graczem i jego zasobami (kredyty, ładunki w statkach).
- Rejestrowanie transakcji (kupno i sprzedaż towarów).
- Przechowywanie misji wraz z możliwością ich ukończenia i przyznania nagrody.
- Integracja funkcji i procedur obliczania trasy, odległości, zarządzania ładunkiem itp.
- Mechanizmy integralności (triggery) automatycznie weryfikujące i logujące działania (np. kupno/sprzedaż, niszczenie statku, walidację nazw graczy).
- Widoki (views) prezentujące najważniejsze zestawienia (np. aktywni gracze, transakcje).

Ograniczenia przyjęte przy projektowaniu:

- Każdy statek może należeć tylko do jednego właściciela (gracz lub NPC).
- Dla uproszczenia, wszystkie operacje finansowe rozgrywają się w jednej walucie, gracze mają stan kredytów typu NUMERIC(18,2).
- Nie uwzględniamy złożonych mechanik bojowych poza prostą, przykładową procedurą `proc_pvp_combat`.
- Odległości i koordynaty zapisywane w typach numerycznych INT z ograniczoną dokładnością (zamiast np. typów float/double).

2. Diagram ER

Diagram ER znajduje się w pliku **er.png**

3. Schemat bazy danych

Schemat bazy znajduje się w pliku **schemat.png**

4. Integralność danych

Przykłady dodatkowych więzów integralności (niezapisane wyłącznie w schemacie, ale w triggerach/funkcjach):

1. **Wyzwalacze** (triggery) sprawdzające:
   - Dostępność kredytów gracza przy **transakcjach** (BUY).
   - Pojemność ładowni statku przy dodawaniu towaru.
   - Własność statku (czy należy do danego gracza).
   - Zmianę pola `is_destroyed` (logowanie faktu zniszczenia statku).
   - Walidację nazwy gracza (tylko alfanumeryczne i `_`).
2. **Sprawdzanie stanu ładunku**:
   - Funkcje `fn_add_cargo` i `fn_remove_cargo` rzucające wyjątki, jeśli ilość towaru jest niewystarczająca.
3. **Mechanizmy ON DELETE**:
   - ON DELETE CASCADE (np. usunięcie systemu → usunięcie powiązanych planet).
   - ON DELETE SET NULL (np. stacja traci frakcję, jeśli dana frakcja zostanie usunięta).

---

5. Indeksy

Tworzone są indeksy na kluczach głównych (PRIMARY KEY).
Przykłady (z pliku **tables.sql**):

- `PRIMARY KEY` na kolumnach typu `id` (np. `system_id`, `player_id`, `ship_id`, `deal_id`).
- `PRIMARY KEY(ship_id, good_id)` w tabeli **ShipCargo**.

W razie potrzeby można tworzyć dodatkowe indeksy, np. po kolumnie `deal_type`, `player_id` w `Deals`, ale w tym projekcie wystarczą indeksy domyślne tworzone przy kluczach głównych.

---

6. Widoki i procedury

Widoki

W pliku **views.sql** zdefiniowano m.in.:

- **vw_active_players** — filtruje graczy z dodatnim saldem kredytów.
- **vw_stations_with_faction** — wyświetla stacje wraz z nazwą kontrolującej je frakcji.
- **vw_ships_details** — łączy informacje o statkach graczy i NPC, pokazując je w jednej tabeli logicznej.
- **vw_deals_summary** — podsumowanie transakcji (typ transakcji, stacja, gracz, towar, cena).

- **vw_goods_prices** — ostatnia znana cena towaru na danej stacji (z historii cen).

Procedury

1. **proc_generate_random_systems_and_planets**
   Generuje losowo podaną liczbę systemów oraz planet w każdym systemie.
2. **proc_transfer_credits**
   Przenosi określoną kwotę kredytów z jednego gracza na drugiego, walidując stan konta.
3. **proc_finish_mission**
   Kończy misję i w zależności od powodzenia — usuwa wymagany towar, przyznaje nagrodę, ustawia status misji.
4. **proc_create_system**, **proc_create_planet**, **proc_create_station**, **proc_create_mission**, **proc_finish_mission**
   Proste procedury wstawiające rekordy do odpowiednich tabel.
5. **proc_pvp_combat**
   Przykład procedury symulującej proste PvP — losowo decyduje o zwycięzcy i przenosi stawkę kredytów.

Funkcje

- **func_get_distance(systemA, systemB)** — oblicza odległość 3D między dwoma systemami gwiezdnymi.
- **func_find_path(systemA, systemB, ship_id)** — wyszukuje trasę uwzględniając maksymalny zasięg skoku statku.
- **func_calc_player_profit(player_id)** — wylicza zysk netto (suma SELL - suma BUY) dla danego gracza.
- **func_upgrade_ship(ship_id, module_name)** — dodaje nowy moduł do statku lub zwiększa poziom istniejącego.
- **fn_add_cargo** — dodaje ladunek do statku, jezeli juz istnieje, zwieksza ilosc
- **fn_remove_cargo** — suwa ladunek ze statku; rzuca wyjatek, jezeli ilosc jest niewystarczajaca

---

7. Wyzwalacze

W pliku **triggers.sql**:

1. **deals_after_insert**
   - Weryfikuje, czy gracz ma wystarczającą ilość kredytów przy zakupie (BUY).
   - Sprawdza pojemność statku.
   - Dodaje lub usuwa ładunek w funkcjach `fn_add_cargo` / `fn_remove_cargo`.
   - Aktualizuje stan konta gracza.
   - Loguje zakup/sprzedaż w tabeli **Logs**.

2. **ship_destroyed_trigger**
    - Po aktualizacji statku sprawdza, czy `is_destroyed` zmieniło się z FALSE na TRUE — jeżeli tak, wstawia log do **Logs**.
3. **check_player_name_trigger**
    - Przed wstawieniem/aktualizacją w **Players** sprawdza, czy nazwa gracza spełnia wyrażenie regularne (tylko `[a-zA-Z0-9_]`).
4. **goods_price_history_trigger**
    - Po wstawieniu nowego wpisu z ceną towaru na stacji loguje zdarzenie w tabeli **Logs**.
5. **block_delete_open_mission_trigger**
    - Blokuje usuwanie misji, która jest w statusie `Open` lub `InProgress`.
6. **validate_ship_in_deals** i **validate_ship_in_cargo**
    - Przed wstawieniem/aktualizacją sprawdzają, czy dany `ship_id` istnieje w tabeli nadrzędnej **Ships**.

---

8. Skrypt tworzący bazę danych

W pliku **create_database.sql** znajdują się polecenia:

- **DROP DATABASE IF EXISTS spacesimdb;**
- **CREATE DATABASE spacesimdb ...**

Następnie (po zmianie kontekstu w psql: `\c spacesimdb`) można wykonać kolejne pliki .sql zawierające definicje tabel, funkcji, procedur, triggerów oraz wstawić dane przykładowe.

Opis działania `setup_from_files.py`:
Skrypt (niewidoczny tu w całości, ale docelowo) wykonuje serię poleceń psql lub korzysta z biblioteki psycopg2, by:

1. Utworzyć bazę (o ile nie istnieje) lub ją wyczyścić.
2. Wczytać pliki .sql w odpowiedniej kolejności:
    - tables.sql (struktura tabel)
    - triggers.sql (definicje wyzwalaczy)
    - functionsandprocedures.sql (funkcje i procedury)
    - views.sql (widoki)
    - inserts.sql (dane przykładowe)
3. W rezultacie baza `spacesimdb` będzie gotowa do użycia przez aplikację **app.py**.

---

9. Zapytania (przykładowe, typowe)

W pliku **examplequeries.sql** znajdziemy m.in.:

```
-- lista aktywnych graczy
SELECT * FROM vw_active_players;

-- stacje z frakcja
SELECT * FROM vw_stations_with_faction;

-- szczegoly statkow
SELECT * FROM vw_ships_details;

-- podsumowanie transakcji
SELECT * FROM vw_deals_summary;

-- ostatnie ceny towarow
SELECT * FROM vw_goods_prices;

-- oblicz odleglosc miedzy Sol(1) a AlphaCentauri(2)
SELECT func_get_distance(1, 2) AS dist;

-- test wyszukiwania trasy (dla statku id=2)
SELECT func_find_path(1, 3, 2) AS route;
```

Możemy również wykonywać inne zapytania np. do tabel logów, misji, ładunku statku itp.

Uruchomienie projektu

Poniżej przykładowe polecenia, zakładając że mamy zainstalowanego Pythona, pip oraz serwer PostgreSQL (wraz z utworzonym użytkownikiem `postgres` i hasłem `postgres`):

1. **Zainstaluj zależności**:

   ```
   pip install -r requirements.txt
   ```

2. **Uruchom skrypt setup_from_files.py** (przykładowo):

   ```
   python setup_from_files.py
   ```

   - Ten skrypt utworzy bazę `spacesimdb`, załaduje pliki .sql z definicjami tabel, widoków, procedur, triggerów i wstawi dane testowe.

3. **Uruchom aplikację Flask**:

   ```
   python app.py
   ```

    - Aplikacja będzie dostępna pod adresem http://127.0.0.1:5000/.

W ten sposób możemy przetestować funkcjonalność bazy, tworzyć nowe rekordy w interfejsie webowym, przeglądać istniejące dane (systemy, planety, stacje, statki, gracze itd.) oraz wywoływać procedury (np. generowanie nowych systemów, finalizowanie misji, obliczanie odległości, itd.).

Strategia pielęgnacji bazy danych:

- Codzienne automatyczne kopie zapasowe wykonywane w godzinach nocnych.
- W przypadku awarii możliwość odtworzenia danych z logów.
- Backup przechowywany lokalnie, na serwerze zapasowym oraz dodatkowo w chmurze.