SAKARYA ÜNİVERSİTESİ BİLGİSAYAR MÜHENDİSLİĞİ
2025-2026 GÜZ DÖNEMİ
VERİ TABANI YÖNETİM SİSTEMLERİ DERSİ PROJE RAPORU

Hazırlayan: İsmail Fatih Çolak
Numara: B241210062
Tarih: 12 Aralık 2025
Projenin GitHub Reposu: https://github.com/kuyash71/ttrpgDatabase

a. Uygulamanın Kısa Tanıtımı ve İş Kuralları

Giriş

Bu proje kapsamında, masaüstü rol yapma oyunları (Tabletop Role-Playing Games – TTRPG) için çoklu evren ve sistem desteği sunan, ilişkisel veritabanı tabanlı bir karakter ve oyun yönetim sistemi tasarlanmış ve uygulanmıştır. Projenin amacı; farklı TTRPG sistemlerine ait evrenler, kampanyalar, karakterler, sınıflar, istatistikler ve envanter bilgilerinin tutarlı, genişletilebilir ve güvenli bir biçimde saklanmasını sağlamaktır.

Bu doğrultuda PostgreSQL veritabanı yönetim sistemi kullanılarak; tablolar, birincil ve yabancı anahtarlar, stored procedure'ler ve trigger'lar aracılığıyla veri bütünlüğü sağlanmıştır. İş kuralları yalnızca uygulama seviyesinde değil, veritabanı seviyesinde de enforced edilerek hatalı veri girişlerinin önüne geçilmiştir. Ayrıca sistem, yeni oyun evrenleri ve kurallarının eklenmesine olanak tanıyacak şekilde modüler bir yapıda tasarlanmıştır.

İş Kuralları

Bu proje, masaüstü rol yapma oyunları (Tabletop Role-Playing Games – TTRPG) için birden fazla oyun sistemi ve evreni destekleyebilen bir karakter ve evren veritabanı yönetim sistemi geliştirmeyi amaçlamaktadır. Sistem; oyun evrenleri, kampanyalar, karakterler, sınıflar (class/preset), statlar, kaynaklar, yetenekler ve envanter gibi oyun bileşenlerinin düzenli, tutarlı ve genişletilebilir şekilde saklanmasını sağlar.

Aşağıda sistemin uyması gereken temel iş kuralları tanımlanmıştır.

1. Sistem ve Evren Kuralları

Sistem, birden fazla oyun sistemini (örneğin Umbra Caelis, Shwarzesonne gibi) destekleyebilmelidir.

Her oyun sistemi kendine özgü stat, kaynak ve class/preset tanımlarına sahiptir.

Bir evren, yalnızca bir oyun sistemine bağlıdır.

Bir evren birden fazla kampanya içerebilir.

Bir kampanya yalnızca bir evrene bağlıdır.

2. Karakter Kuralları

Her karakter yalnızca bir "campaign"e bağlıdır.

Her karakter yalnızca bir oyun sistemine ve bir class/preset'e sahiptir.

Bir kampanya içerisinde birden fazla karakter bulunabilir.

Karakterler oyuncu karakteri (PC) veya oyuncu olmayan karakter (NPC) olabilir.

Karakterin temel bilgileri (isim, hikâye, güçlü/zayıf yönler vb.) sistem tarafından saklanmalıdır.

3. Stat Kuralları

Statlar, oyun sistemine bağlı olarak tanımlanır.

Her oyun sistemi, kendine özgü bir stat kümesine sahiptir.

Her stat için minimum ve maksimum değerler tanımlanmalıdır.

Her karakter, bağlı olduğu oyun sistemindeki tüm statlara sahip olmak zorundadır.

Karakter stat değerleri, tanımlı minimum ve maksimum sınırların dışına çıkamaz.

Stat değerleri, karakter oluşturulurken otomatik olarak oluşturulmalıdır.

4. Class / Preset Kuralları

Her class/preset, yalnızca bir oyun sistemine bağlıdır.

Bir class/preset, karakterin başlangıç statlarını etkileyen stat değiştiricilere (artı/eksi değerler) sahip olabilir.

Class/preset stat değiştiricileri, karakter oluşturulurken otomatik olarak uygulanmalıdır.

Bir karakterin class/preset'i değiştirildiğinde, stat değiştiricileri yeniden hesaplanmalıdır.

Class/preset'ler, bir veya daha fazla yetenek (ability) içerebilir.

5. Kaynak (Resource) Kuralları

Kaynaklar (örneğin sağlık, esin puanı, yiyecek vb.) oyun sistemine bağlı olarak tanımlanır.

Her karakter, oyun sisteminde tanımlı tüm kaynaklara sahip olmalıdır.

Kaynak değerleri minimum ve maksimum sınırlar içinde tutulmalıdır.

Kaynak değerleri oyun sırasında güncellenebilir.

6. Yetenek (Ability) Kuralları

Yetenekler, oyun sistemine bağlı olarak tanımlanır.

Yetenekler açıklama ve kullanım kuralları içerebilir.

Bir yetenek birden fazla class/preset tarafından kullanılabilir.

Bir karakter, class/preset'i aracılığıyla birden fazla yeteneğe sahip olabilir.

7. Envanter Kuralları

Oyun sisteminde tanımlı eşyalar karakterlerin envanterine eklenebilir.

Bir karakter birden fazla eşyaya sahip olabilir.

Envanterdeki her eşyanın bir adet (quantity) bilgisi bulunmalıdır.

Bir eşyanın adedi sıfır olduğunda envanter kaydı otomatik olarak silinmelidir.

8. Veri Bütünlüğü ve Kayıt Kuralları

Sistem üzerinde yapılan ekleme, silme ve güncelleme işlemleri kayıt altına alınmalıdır.

Tanımlı olmayan bir stat, kaynak veya class/preset bir karaktere atanamaz.

Veritabanı bütünlüğü, yabancı anahtarlar, kısıtlamalar ve tetikleyiciler (trigger) ile korunmalıdır.

9. Uygulama İşlevleri Kuralları

Sistem; arama, ekleme, silme ve güncelleme (CRUD) işlemlerini desteklemelidir.

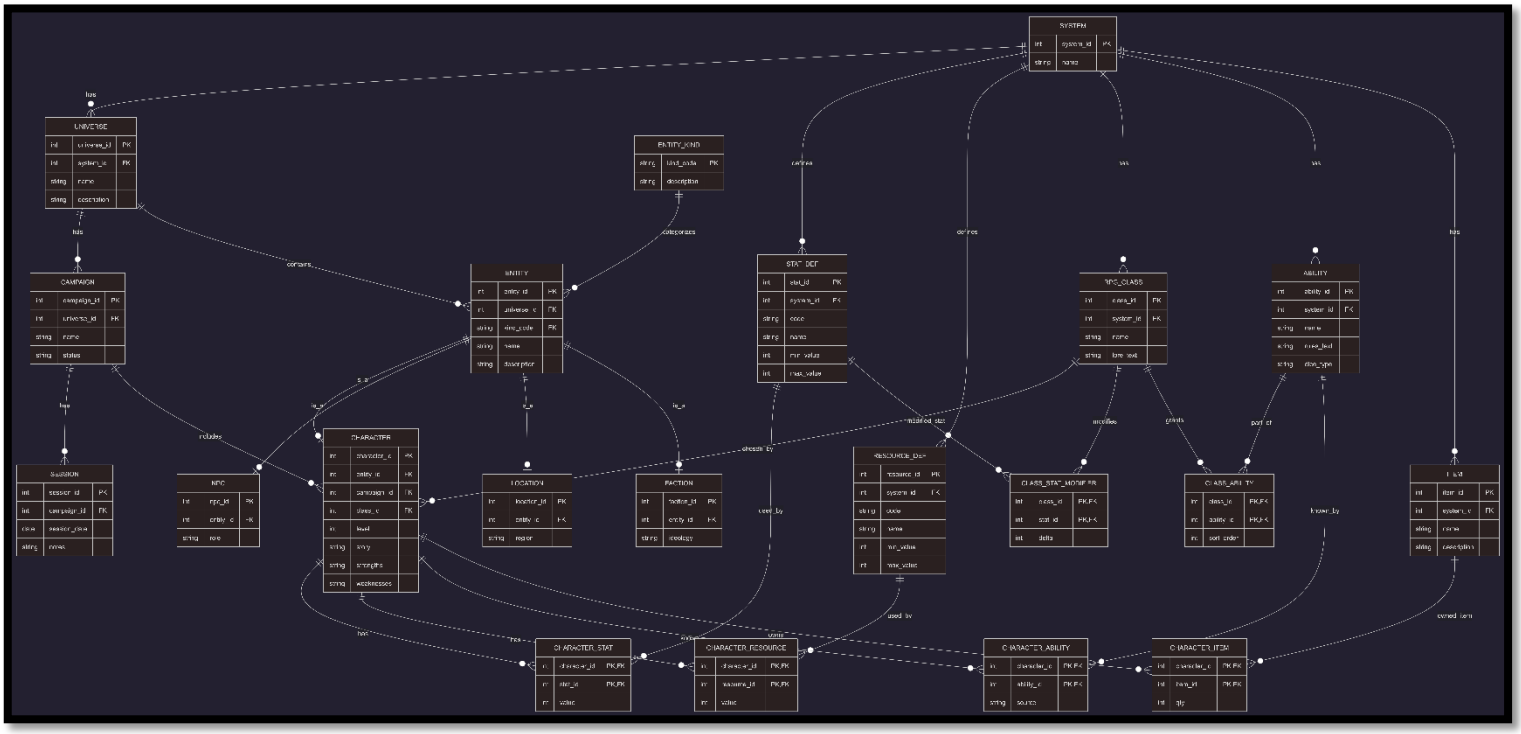
Kullanıcı, karakter bilgilerini ve statlarını görüntüleyebilmelidir.

Karakter oluşturma işlemleri veritabanı saklı yordamları (stored procedure) aracılığıyla gerçekleştirilmelidir.

İş kuralları, mümkün olduğunca veritabanı seviyesinde uygulanmalıdır.

b. Varlık Bağıntı Modeli

Projenin ER diyagramı aşağıda mevcuttur, görselin orijinali github reposunda yer almaktadır.



b.1. Projenin metinsel gösterim ile ilişkisel şeması:

```
CHARACTER_RESOURCE(
    character_id PK, FK → CHARACTER(character_id),
    resource_id PK, FK → RESOURCE_DEF(resource_id),
    value
)
RPG_CLASS(
    class_id PK,
    system_id FK → SYSTEM(system_id),
    name,
    lore_text
)
ABILITY(
    ability_id PK,
    system_id FK → SYSTEM(system_id),
    name,
    usage_text,
    dice_type
)
CLASS_STAT_MODIFIER(
    class_id PK, FK → RPG_CLASS(class_id),
    stat_id PK, FK → STAT_DEF(stat_id),
    delta
)
CLASS_ABILITY(
    class_id PK, FK → RPG_CLASS(class_id),
    ability_id PK, FK → ABILITY(ability_id),
    sort_order
)
CHARACTER_ABILITY(
    character_id PK, FK → CHARACTER(character_id),
    ability_id PK, FK → ABILITY(ability_id),
    source
)
ITEM(
    item_id PK,
    system_id FK → SYSTEM(system_id),
    name,
    description
)
CHARACTER_ITEM(
    character_id PK, FK → CHARACTER(character_id),
    item_id PK, FK → ITEM(item_id),
    qty
)
```

```
ENTITY(
    entity_id PK,
    universe_id FK → UNIVERSE(universe_id),
    kind_code FK → ENTITY_KIND(kind_code),
    name,
    description
)
CHARACTER(
    character_id PK,
    entity_id FK → ENTITY(entity_id),
    campaign_id FK → CAMPAIGN(campaign_id),
    class_id FK → RPG_CLASS(class_id),
    level,
    story,
    starting_loc,
    weaknesses
)
NPC(
    npc_id PK,
    entity_id FK → ENTITY(entity_id),
    role
)
LOCATION(
    location_id PK,
    entity_id FK → ENTITY(entity_id),
    region
)
```

```
SYSTEM(
    system_id PK,
    name
)
UNIVERSE(
    universe_id PK,
    system_id FK → SYSTEM(system_id),
    name,
    description
)
CAMPAIGN(
    campaign_id PK,
    universe_id FK → UNIVERSE(universe_id),
    name,
    status
)
SESSION(
    session_id PK,
    campaign_id FK → CAMPAIGN(campaign_id),
    session_date,
    notes
)
ENTITY_KIND(
    kind_code PK,
    description
)
```

```
FACTION(
    faction_id PK,
    entity_id FK → ENTITY(entity_id),
    ideology
)
STAT_DEF(
    stat_id PK,
    system_id FK → SYSTEM(system_id),
    code,
    name,
    min_value,
    max_value
)
RESOURCE_DEF(
    resource_id PK,
    system_id FK → SYSTEM(system_id),
    code,
    name,
    min_value,
    max_value
)
CHARACTER_STAT(
    character_id PK, FK → CHARACTER(character_id),
    stat_id PK, FK → STAT_DEF(stat_id),
    value
)
```

c.Veritabanını, içindeki veriler ile oluşturmayı sağlayan SQL ifadeleri

Veritabanını oluşturan tüm SQL ifadelerine “01_schema.sql” adlı dosyanın içerisinden erişilebilmektedir. Oluşturulan veritabanının içindeki başlangıç verilerini ise “02_seed.sql” adlı dosyanın içerisindeki SQL ifadeleri ile oluşturulur.

c.1. İlgili “01_schema.sql” dosyasının içindeki SQL ifadeleri:

```
CREATE TABLE campaign (
  campaign_id  INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  universe_id  INTEGER NOT NULL REFERENCES universe(universe_id) ON DELETE CASCADE,
  name         VARCHAR(160) NOT NULL,
  status       VARCHAR(40),
  CONSTRAINT uq_campaign__universe_name UNIQUE (universe_id, name)
);

CREATE TABLE session (
  session_id    INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  campaign_id   INTEGER NOT NULL REFERENCES campaign(campaign_id) ON DELETE CASCADE,
  session_date  DATE,
  notes         TEXT
);
-- ENTITY KIND (lookup) + ENTITY
CREATE TABLE entity_kind (
  kind_code   VARCHAR(24) PRIMARY KEY,
  description VARCHAR(200)
);

CREATE TABLE entity (
  entity_id    INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  universe_id  INTEGER NOT NULL REFERENCES universe(universe_id) ON DELETE CASCADE,
  kind_code    VARCHAR(24) NOT NULL REFERENCES entity_kind(kind_code) ON DELETE RESTRICT,
  name         VARCHAR(160) NOT NULL,
  description  TEXT
);
-- KALITIM TABLOLARI
CREATE TABLE character (
  character_id  INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  entity_id     INTEGER NOT NULL UNIQUE REFERENCES entity(entity_id) ON DELETE CASCADE,
  campaign_id   INTEGER NOT NULL REFERENCES campaign(campaign_id) ON DELETE CASCADE,
  class_id      INTEGER NOT NULL,
  level         INTEGER NOT NULL DEFAULT 0 CHECK (level >= 0),
  story         TEXT,
  strengths     TEXT,
  weaknesses    TEXT
);
```

```
CREATE TABLE npc (
  npc_id     INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  entity_id  INTEGER NOT NULL UNIQUE REFERENCES entity(entity_id) ON DELETE CASCADE,
  role       VARCHAR(120)
);

CREATE TABLE location (
  location_id  INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  entity_id    INTEGER NOT NULL UNIQUE REFERENCES entity(entity_id) ON DELETE CASCADE,
  region       VARCHAR(120)
);

CREATE TABLE faction (
  faction_id  INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  entity_id   INTEGER NOT NULL UNIQUE REFERENCES entity(entity_id) ON DELETE CASCADE,
  ideology    VARCHAR(160)
);
-- STAT / RESOURCE DEFINITIONS
CREATE TABLE stat_def (
  stat_id     INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  system_id   INTEGER NOT NULL REFERENCES system(system_id) ON DELETE CASCADE,
  code        VARCHAR(40) NOT NULL,
  name        VARCHAR(120) NOT NULL,
  min_value   INTEGER NOT NULL,
  max_value   INTEGER NOT NULL,
  CONSTRAINT ck_stat_def_minmax CHECK (min_value <= max_value),
  CONSTRAINT uq_stat_def__system_code UNIQUE (system_id, code),
  CONSTRAINT uq_stat_def__system_name UNIQUE (system_id, name)
);

CREATE TABLE resource_def (
  resource_id  INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  system_id    INTEGER NOT NULL REFERENCES system(system_id) ON DELETE CASCADE,
  code         VARCHAR(40) NOT NULL,
  name         VARCHAR(120) NOT NULL,
  min_value    INTEGER NOT NULL,
  max_value    INTEGER NOT NULL,
  CONSTRAINT ck_resource_def_minmax CHECK (min_value <= max_value),
  CONSTRAINT uq_resource_def__system_code UNIQUE (system_id, code),
  CONSTRAINT uq_resource_def__system_name UNIQUE (system_id, name)
);
```

```sql
CREATE TABLE character_stat (
  character_id  INTEGER NOT NULL REFERENCES character(character_id) ON DELETE CASCADE,
  stat_id       INTEGER NOT NULL REFERENCES stat_def(stat_id) ON DELETE RESTRICT,
  value         INTEGER NOT NULL DEFAULT 0,
  PRIMARY KEY (character_id, stat_id)
);

CREATE TABLE character_resource (
  character_id  INTEGER NOT NULL REFERENCES character(character_id) ON DELETE CASCADE,
  resource_id   INTEGER NOT NULL REFERENCES resource_def(resource_id) ON DELETE RESTRICT,
  value         INTEGER NOT NULL DEFAULT 0,
  PRIMARY KEY (character_id, resource_id)
);
-- CLASS / ABILITY / ITEM
CREATE TABLE rpg_class (
  class_id    INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  system_id   INTEGER NOT NULL REFERENCES system(system_id) ON DELETE CASCADE,
  name        VARCHAR(120) NOT NULL,
  lore_text   TEXT,
  CONSTRAINT uq_rpg_class__system_name UNIQUE (system_id, name)
);

ALTER TABLE character
  ADD CONSTRAINT fk_character_class
  FOREIGN KEY (class_id) REFERENCES rpg_class(class_id)
  ON DELETE RESTRICT;

CREATE TABLE class_stat_modifier (
  class_id  INTEGER NOT NULL REFERENCES rpg_class(class_id) ON DELETE CASCADE,
  stat_id   INTEGER NOT NULL REFERENCES stat_def(stat_id) ON DELETE RESTRICT,
  delta     INTEGER NOT NULL,
  PRIMARY KEY (class_id, stat_id)
);
```

```sql
CREATE TABLE ability (
  ability_id   INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  system_id    INTEGER NOT NULL REFERENCES system(system_id) ON DELETE CASCADE,
  name         VARCHAR(160) NOT NULL,
  rules_text   TEXT,
  dice_type    VARCHAR(40),
  CONSTRAINT uq_ability__system_name UNIQUE (system_id, name)
);

CREATE TABLE class_ability (
  class_id    INTEGER NOT NULL REFERENCES rpg_class(class_id) ON DELETE CASCADE,
  ability_id  INTEGER NOT NULL REFERENCES ability(ability_id) ON DELETE RESTRICT,
  sort_order  INTEGER NOT NULL DEFAULT 0 CHECK (sort_order >= 0),
  PRIMARY KEY (class_id, ability_id)
);

CREATE TABLE character_ability (
  character_id  INTEGER NOT NULL REFERENCES character(character_id) ON DELETE CASCADE,
  ability_id    INTEGER NOT NULL REFERENCES ability(ability_id) ON DELETE RESTRICT,
  source        VARCHAR(60), -- e.g. 'CLASS', 'REWARD', 'ITEM', 'MANUAL'
  PRIMARY KEY (character_id, ability_id)
);

CREATE TABLE item (
  item_id      INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  system_id    INTEGER NOT NULL REFERENCES system(system_id) ON DELETE CASCADE,
  name         VARCHAR(160) NOT NULL,
  description  TEXT,
  CONSTRAINT uq_item__system_name UNIQUE (system_id, name)
);

CREATE TABLE character_item (
  character_id  INTEGER NOT NULL REFERENCES character(character_id) ON DELETE CASCADE,
  item_id       INTEGER NOT NULL REFERENCES item(item_id) ON DELETE RESTRICT,
  qty           INTEGER NOT NULL DEFAULT 1 CHECK (qty >= 0),
  PRIMARY KEY (character_id, item_id)
);
```

```sql
CREATE TABLE audit_log (
  audit_id   BIGINT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  table_name TEXT NOT NULL,
  action     TEXT NOT NULL,
  changed_at TIMESTAMPTZ NOT NULL DEFAULT now(),
  row_pk     TEXT,
  old_data   JSONB,
  new_data   JSONB
);

CREATE INDEX idx_entity_universe_kind ON entity(universe_id, kind_code);
CREATE INDEX idx_character_campaign ON character(campaign_id);
CREATE INDEX idx_stat_def_system ON stat_def(system_id);
CREATE INDEX idx_resource_def_system ON resource_def(system_id);

COMMIT;
```

c.2.İlgili “02_seed.sql” dosyasının içerisindeki SQL ifadeleri:

```sql
BEGIN;

INSERT INTO entity_kind(kind_code, description) VALUES
  ('CHARACTER', 'Player Character (PC)'),
  ('NPC',       'Non-player character'),
  ('LOCATION',  'Location / place'),
  ('FACTION',   'Faction / organization')
ON CONFLICT (kind_code) DO NOTHING;

-- Systems
INSERT INTO system(name) VALUES
  ('Umbra Caelis'),
  ('Wolfenstein')
ON CONFLICT (name) DO NOTHING;

-- Universe
INSERT INTO universe(system_id, name, description)
SELECT s.system_id, 'Umbra Caelis Core', 'Main setting'
FROM system s
WHERE s.name = 'Umbra Caelis'
ON CONFLICT (system_id, name) DO NOTHING;

-- Campaign (basit)
INSERT INTO campaign(universe_id, name, status)
SELECT u.universe_id, 'Campaign 1', 'ACTIVE'
FROM universe u
WHERE u.name = 'Umbra Caelis Core'
ON CONFLICT (universe_id, name) DO NOTHING;

```

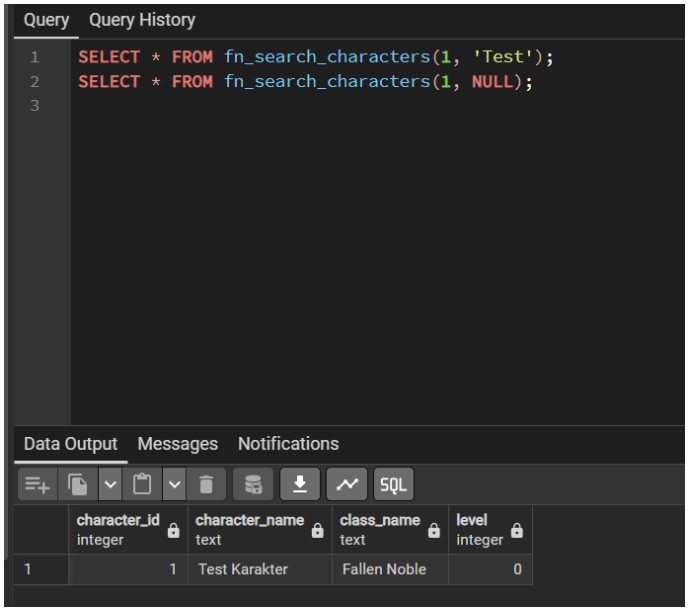
```sql
INSERT INTO stat_def(system_id, code, name, min_value, max_value)
SELECT s.system_id, v.code, v.name, 0, 6
FROM system s
JOIN (VALUES
 ('HAK','Hakimiyet'),
 ('SEB','Sebahat'),
 ('CAZ','Cazibe'),
 ('HIK','Hikmet'),
 ('GOR','Görü'),
 ('MAR','Marifet'),
 ('YAL','Yalman'),
 ('EFS','Efsun'),
 ('KAC','Kaçınç')
) v(code,name) ON true
WHERE s.name='Umbra Caelis'
ON CONFLICT (system_id, code) DO NOTHING;

-- Umbra Caelis - resources (example 0..6
INSERT INTO resource_def(system_id, code, name, min_value, max_value)
SELECT s.system_id, v.code, v.name, 0, 6
FROM system s
JOIN (VALUES
 ('HP','Sağlık'),
 ('ESIN','Esin Puanı'),
 ('LUM','Lumre')
) v(code,name) ON true
WHERE s.name='Umbra Caelis'
ON CONFLICT (system_id, code) DO NOTHING;

-- Sample class: Fallen Noble
INSERT INTO rpg_class(system_id, name, lore_text)
SELECT s.system_id, 'Fallen Noble', 'Preset example'
FROM system s
WHERE s.name='Umbra Caelis'
ON CONFLICT (system_id, name) DO NOTHING;
```

```sql
INSERT INTO rpg_class(system_id, name, lore_text)
SELECT s.system_id, v.name, v.lore
FROM system s
JOIN (VALUES
  ('Arcanist', 'Magic-focused caster.'),
  ('Warden',   'Defensive / tank archetype.'),
  ('Vagabond', 'Mobile / skill-oriented archetype.')
) v(name, lore) ON true
WHERE s.name='Umbra Caelis'
ON CONFLICT (system_id, name) DO NOTHING;


-- Arcanist class
INSERT INTO class_stat_modifier(class_id, stat_id, delta)
SELECT rc.class_id, sd.stat_id, v.delta
FROM rpg_class rc
JOIN system s ON s.system_id = rc.system_id
JOIN stat_def sd ON sd.system_id = s.system_id
JOIN (VALUES
  ('EFS',  2),
  ('HIK',  1),
  ('YAL', -1)
) v(code, delta) ON v.code = sd.code
WHERE s.name='Umbra Caelis' AND rc.name='Arcanist'
ON CONFLICT (class_id, stat_id) DO UPDATE SET delta = EXCLUDED.delta;

-- Warden class
INSERT INTO class_stat_modifier(class_id, stat_id, delta)
SELECT rc.class_id, sd.stat_id, v.delta
FROM rpg_class rc
JOIN system s ON s.system_id = rc.system_id
JOIN stat_def sd ON sd.system_id = s.system_id
JOIN (VALUES
  ('HAK',  2),
  ('MAR',  1),
  ('CAZ', -1)
) v(code, delta) ON v.code = sd.code
WHERE s.name='Umbra Caelis' AND rc.name='Warden'
ON CONFLICT (class_id, stat_id) DO UPDATE SET delta = EXCLUDED.delta;
```

```sql
-- Vagabond
INSERT INTO class_stat_modifier(class_id, stat_id, delta)
SELECT rc.class_id, sd.stat_id, v.delta
FROM rpg_class rc
JOIN system s ON s.system_id = rc.system_id
JOIN stat_def sd ON sd.system_id = s.system_id
JOIN (VALUES
  ('SEB',  1),
  ('GOR',  1),
  ('HAK', -1)
) v(code, delta) ON v.code = sd.code
WHERE s.name='Umbra Caelis' AND rc.name='Vagabond'
ON CONFLICT (class_id, stat_id) DO UPDATE SET delta = EXCLUDED.delta;

-- CAZ
INSERT INTO class_stat_modifier(class_id, stat_id, delta)
SELECT rc.class_id, sd.stat_id, v.delta
FROM rpg_class rc
JOIN system s ON s.system_id = rc.system_id
JOIN stat_def sd ON sd.system_id = s.system_id
JOIN (VALUES
  ('CAZ',  2),
  ('HAK',  1),
  ('SEB',  1),
  ('KAC', -1)
) v(code, delta) ON v.code = sd.code
WHERE s.name='Umbra Caelis' AND rc.name='Fallen Noble'
ON CONFLICT (class_id, stat_id) DO UPDATE SET delta = EXCLUDED.delta;

-- Sample ability + mapping (optional, but useful)
INSERT INTO ability(system_id, name, rules_text, dice_type)
SELECT s.system_id, 'Noble Bearing', 'Sample ability granted by class.', 'd6'
FROM system s
WHERE s.name='Umbra Caelis'
ON CONFLICT (system_id, name) DO NOTHING;
```

```sql
INSERT INTO class_ability(class_id, ability_id, sort_order)
SELECT rc.class_id, a.ability_id, 1
FROM rpg_class rc
JOIN system s ON s.system_id = rc.system_id
JOIN ability a ON a.system_id = s.system_id
WHERE s.name='Umbra Caelis'
  AND rc.name='Fallen Noble'
  AND a.name='Noble Bearing'
ON CONFLICT (class_id, ability_id) DO NOTHING;

-- Sample item
INSERT INTO item(system_id, name, description)
SELECT s.system_id, 'Rope', 'Basic item'
FROM system s
WHERE s.name='Umbra Caelis'
ON CONFLICT (system_id, name) DO NOTHING;

COMMIT;
```

d. 4 Adet Saklı yordam ve 4 adet tetikleyici madde örneği ve işlevleri



d.1.“fn_search_characters” Fonksiyonu (Arama İşlevi)

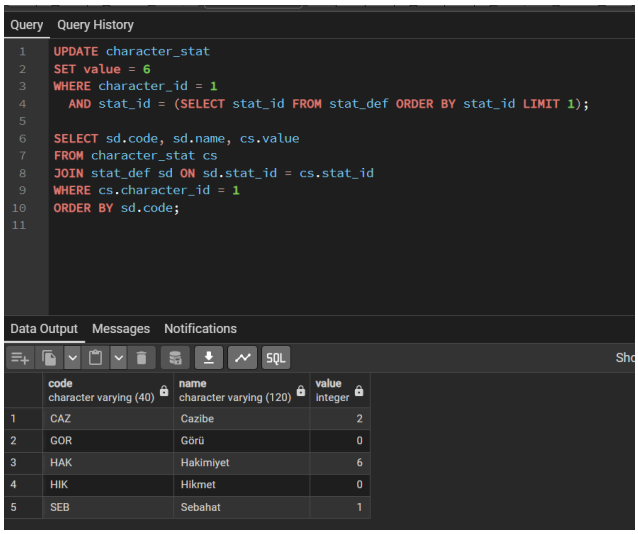
Amaç: Kampanya içerisindeki karakterleri ada göre aramak ve listelemek.

Yapılan İşlem: fn_search_characters(p_campaign_id, p_name_like) fonksiyonu çağrılmıştır.

Fonksiyon, verilen kampanya ID’si içinde karakter adını ILIKE operatörü ile aramaktadır.

Arama parametresi NULL gönderildiğinde kampanyadaki tüm karakterler listelenmektedir.

Sonuç: Fonksiyon başarıyla çalışmış ve 'Test' parametresi ile yalnızca ilgili karakter, NULL parametresi ile kampanyadaki tüm karakterler doğru şekilde döndürülmüştür.



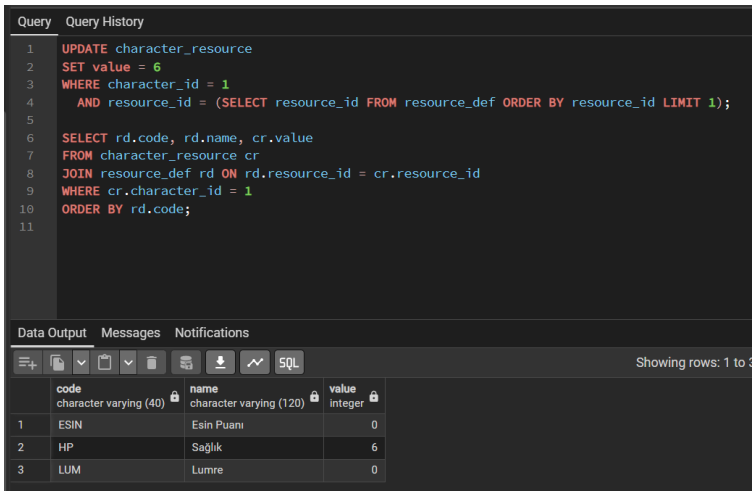
d.2.Stat Min/Max Kontrol Trigger’ı (t_check_stat_bounds)

Amaç: Karakter stat değerlerinin, sistemde tanımlanan minimum ve maksimum sınırlar dışında güncellenmesini engellemek.

Yapılan İşlem: Bir karakterin stat değeri üst sınır olan 6 değerine ayarlanmıştır. Trigger, bu güncellemeyi kontrol ederek kurala uygunluğunu doğrulamıştır.

Sonuç: Güncelleme işlemi başarıyla gerçekleşmiş ve stat değeri sınırlar içinde kalmıştır.

Trigger’ın doğru şekilde çalıştığı doğrulanmıştır.



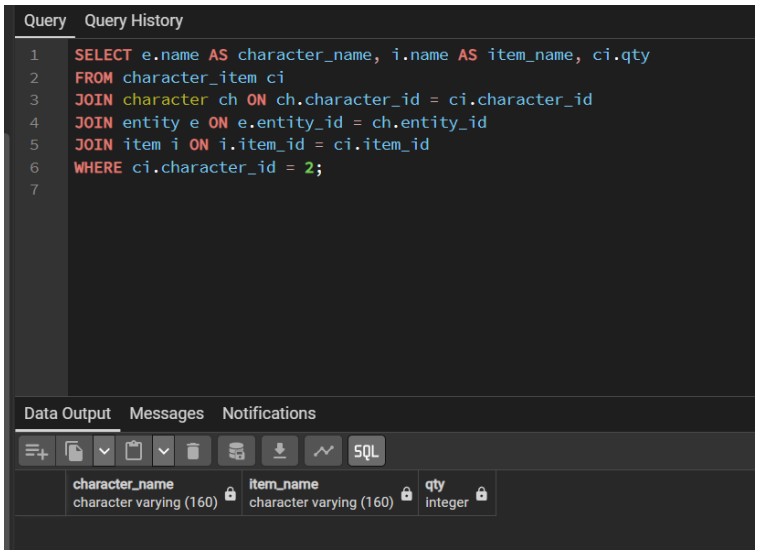
d.3.Resource Min/Max Kontrol Trigger'ı (t_check_resource_bounds)

Amaç: Karakter kaynak (resource) değerlerinin tanımlı aralık dışına çıkmasını engellemek.

Yapılan İşlem: Karaktere ait bir resource değeri maksimum sınır olan 6 olarak güncellenmiştir.

Sonuç: Güncelleme başarıyla tamamlanmış, kaynak değerleri sınırlar içinde tutulmuştur.

Trigger doğru şekilde çalışmaktadır.

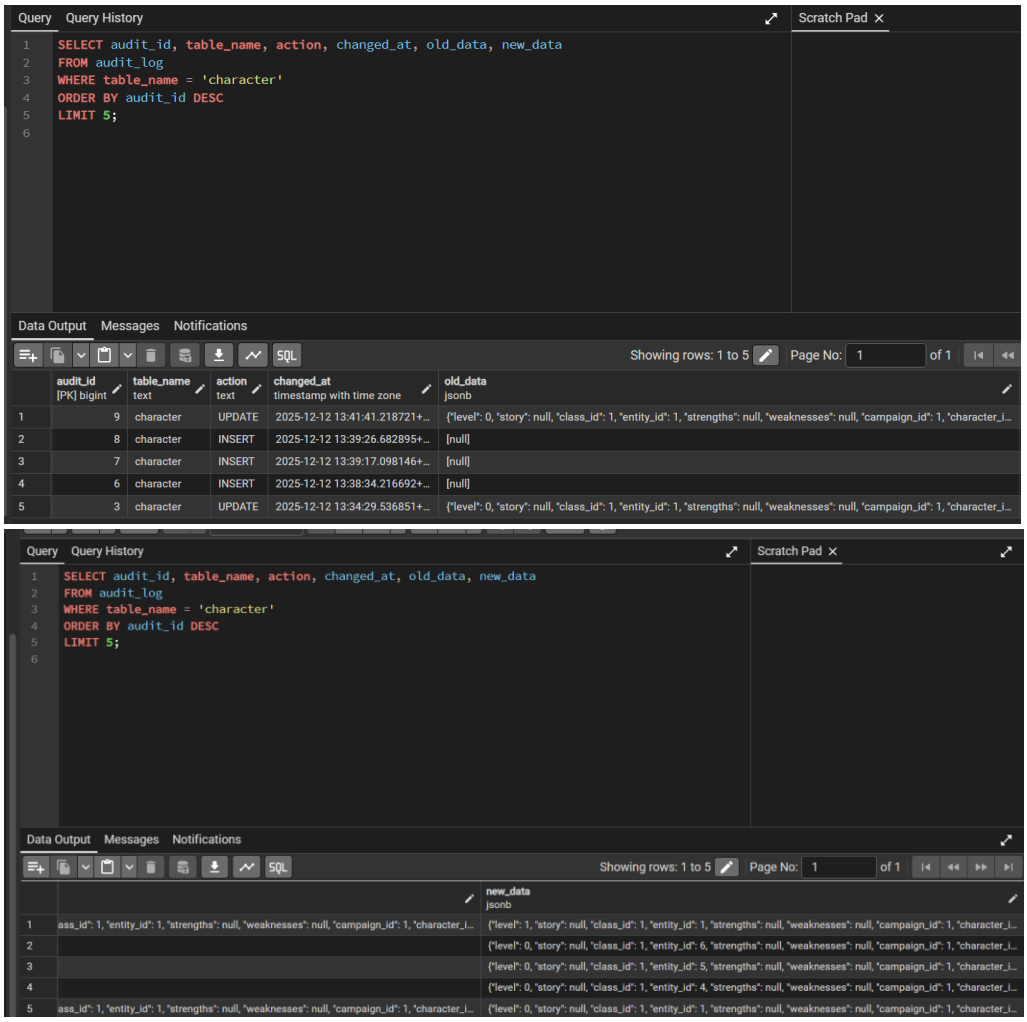


d.4.Envanter Temizleme Trigger'ı (t_item_qty_cleanup)

Amaç: Bir karakterin envanterindeki eşya miktarı 0 olduğunda, ilgili satırın otomatik olarak silinmesini sağlamak.

Yapılan İşlem: Bir karakterin sahip olduğu eşyanın miktarı 0 olacak şekilde güncellenmiştir.

Sonuç: Trigger devreye girerek ilgili eşya satırını otomatik olarak silmiştir. Envanterde qty = 0 olan kayıt kalmamıştır.



d.5.Audit Log Trigger'ı (t_audit_character)

Amaç: character tablosunda gerçekleşen INSERT ve UPDATE işlemlerinin otomatik olarak kayıt altına alınması.

Yapılan İşlem: Bir karakterin level alanı güncellenmiştir. Bu işlem sonrası audit_log tablosu sorgulanmıştır.

Sonuç:

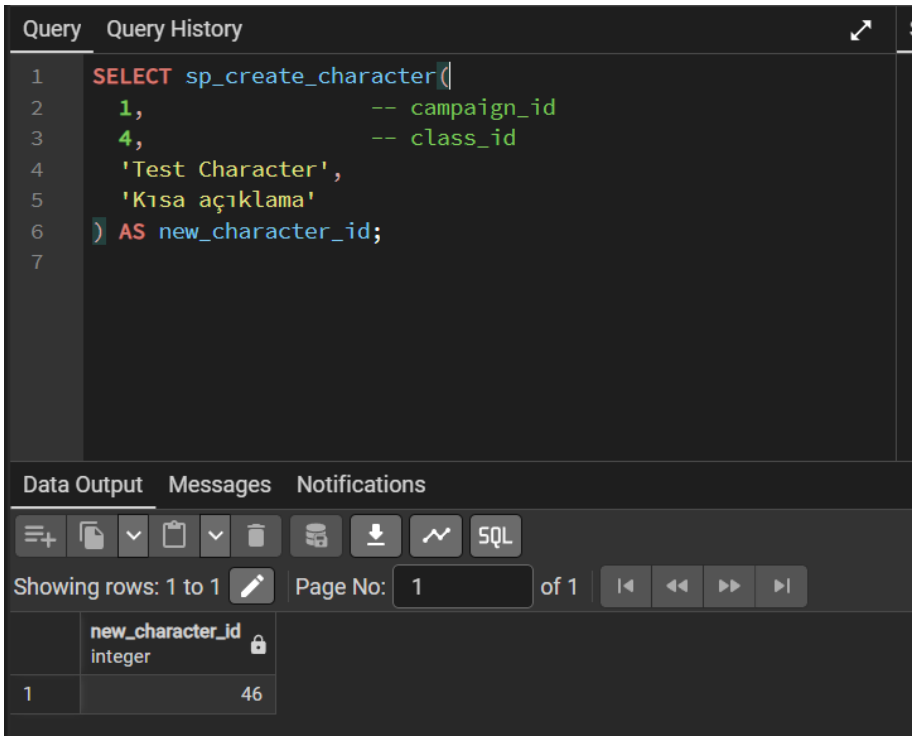
Audit log tablosunda:

İşlem türü (INSERT / UPDATE)

İşlem zamanı

Eski ve yeni veri (JSON formatında)

bilgileri doğru şekilde kaydedilmiştir.

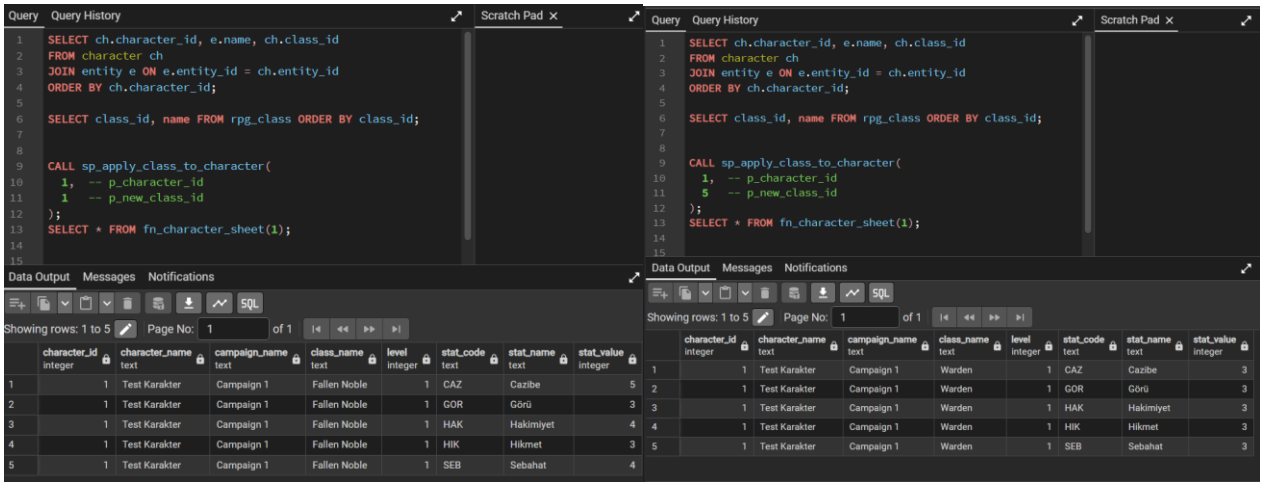


d.6.”sp_create_character” fonksiyonu ile karakter oluşturma

Amaç: Yeni bir karakterin, seçilen kampanya ve class bilgilerine göre tüm bağımlı verileriyle birlikte (entity, stat, resource, ability) tek bir işlem olarak oluşturulmasını sağlamak.

Yaptığı İşlem: Karakter için entity tablosunda kayıt oluşturur, character tablosuna yeni karakteri ekler. İlgili oyun sistemine ait tüm statları otomatik oluşturur ve Kaynak (resource) değerlerini minimum değerleriyle ekler.Class’a ait stat modifier’ları uygular ve son olarak Class’a ait yetenekleri karaktere otomatik tanımlar

Sonuç: Karakter oluşturma işlemi, uygulama katmanından bağımsız olarak tamamen veritabanı seviyesinde ve atomik biçimde gerçekleştirilmiştir.
İşlem sırasında veri bütünlüğü bozulmamaktadır.

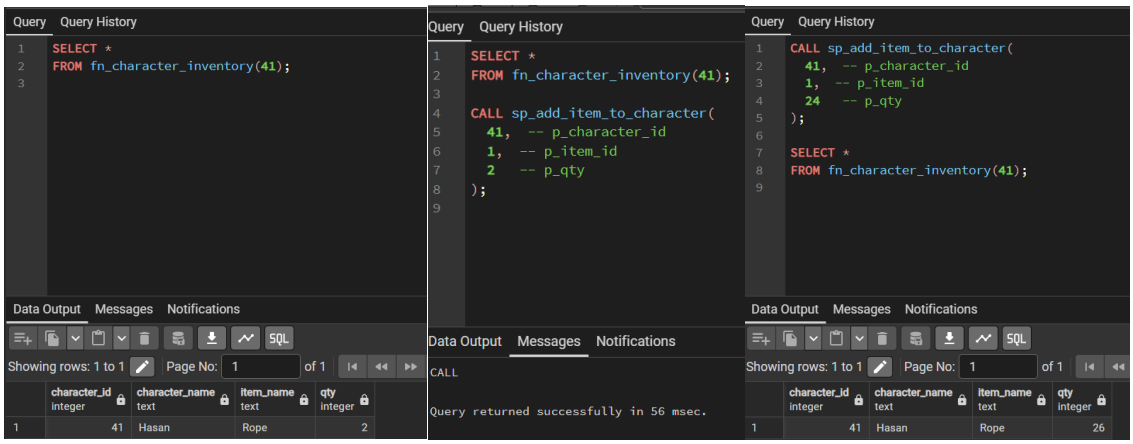


d.7.”sp_apply_class_to_character” fonksiyonu

Amaç: Mevcut bir karakterin class bilgisini değiştirirken, stat değerlerinin yeni class’a uygun şekilde yeniden hesaplanmasını sağlamak.

Yaptığı İşlem: Karakterin class_id bilgisini günceller, Karakterin tüm statlarını baz değere (BASE 3) resetler. Yeni class’a ait stat modifier’ları yeniden uygular ve Yeni class’ın yeteneklerini karaktere ekler

Sonuç: Class değişikliği sonrası statlar hatalı veya çakışmalı hale gelmemekte, tüm hesaplamalar merkezi olarak saklı yordam aracılığıyla yapılmaktadır.



d.8.”sp_transfer_item” fonksiyonu

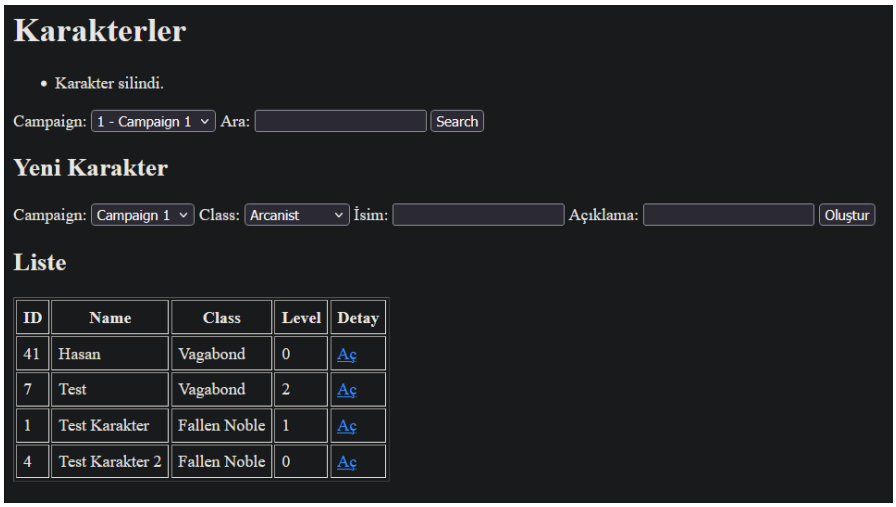
Amaç: Bir karakterin envanterindeki eşyaların, başka bir karaktere kontrollü ve güvenli şekilde transfer edilmesini sağlamak.

Yaptığı İşlem: Gönderen karakterin yeterli eşya adedine sahip olup olmadığını kontrol eder,eşya adedi yetersizse işlemi hata ile iptal eder.Gönderen karakterin envanterinden düşürür ve alıcı karakterin envanterine ekler. İşlem sırasında qty = 0 durumuna düşen kayıtlar trigger tarafından temizlenir.

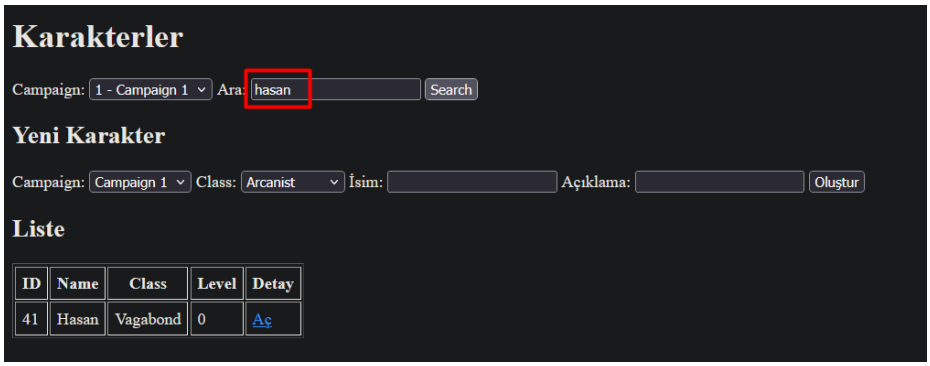
Sonuç: Envanter transfer işlemleri, uygulama seviyesinde kontrol gerektirmeden tamamen veritabanı seviyesinde güvenli şekilde yürütülmektedir.

e. Arama, Ekleme, Silme ve Güncelleme İşlemleri

Veri tabanının başlangıçtaki hali aşağıdaki şekildedir:

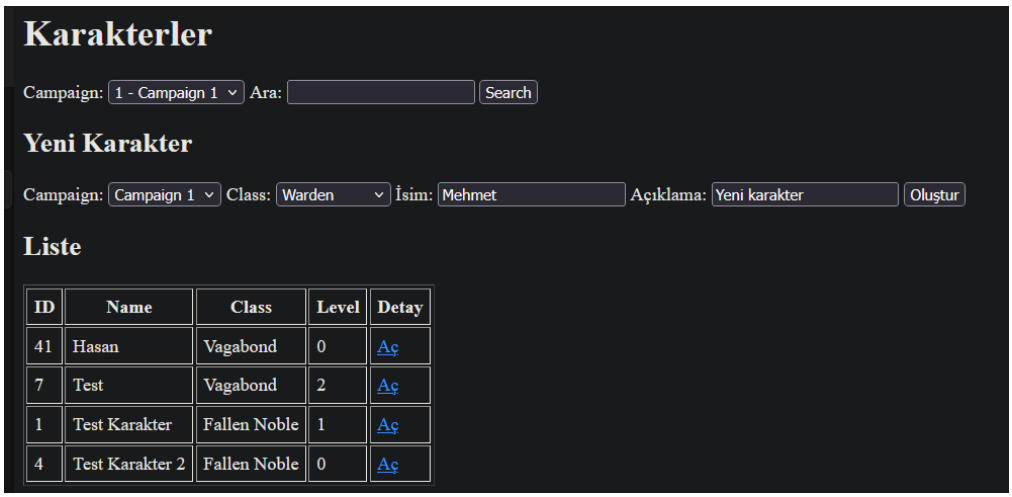


e.1. Arama İşlemi

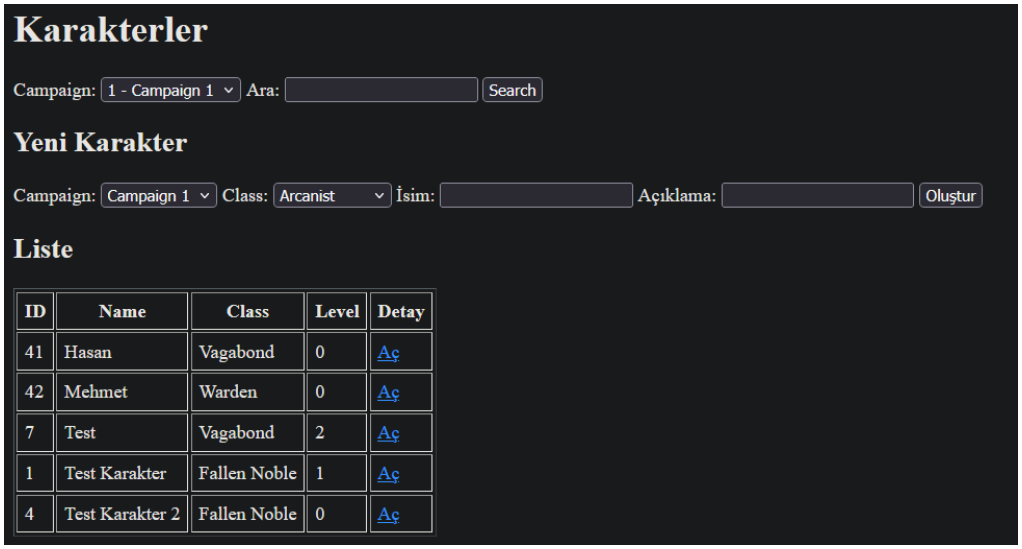


Kırmızı ile işaretlenmiş kutucuğa aranacak karakterin adı yazılır ve “Search” butonuna basılarak arama yapılır. Aratılan kritere uygun karakter(ler) listelenir.

e.2. Ekleme İşlemi



Yukarıda yer alan “Yeni Karakter” başlığının altındaki alan kullanılarak yeni oluşturulacak karaktere ait temel bilgiler girilir ve “Oluştur” tuşu ile yeni karakter oluşturulur.



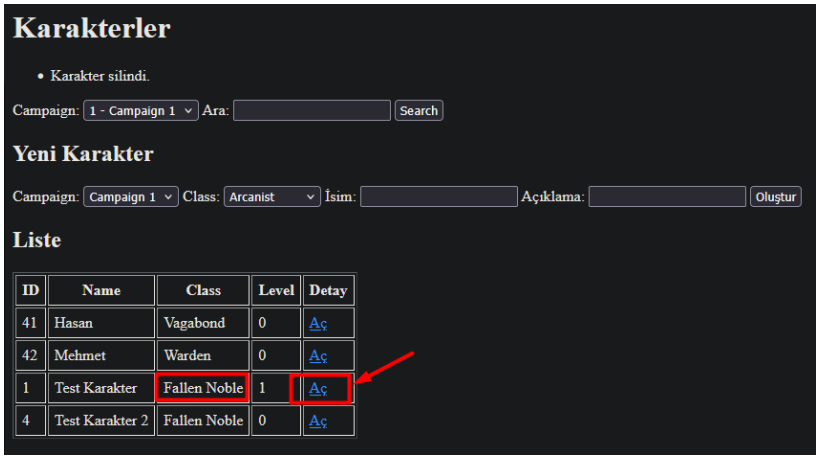
Yeni oluşturulan karakter Liste’nin altında listelenir.

e.3. Silme İşlemi

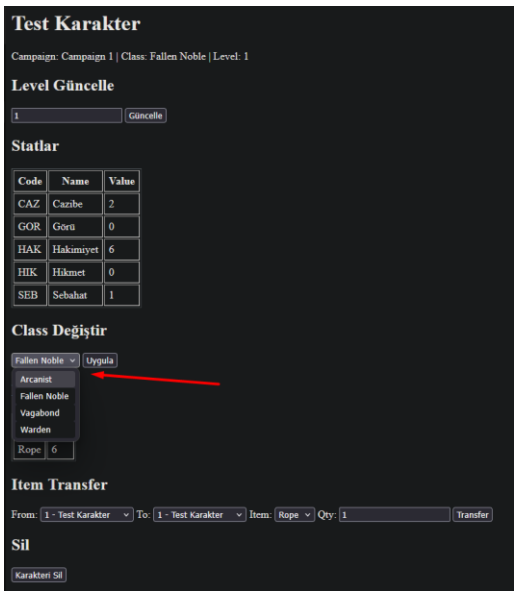
Karakterin detay penceresi “Detay” başlığı altındaki “Aç” tuşu ile açılır ve detay penceresinin altındaki “Karakter Sil” tuşu ile karakter silinir. Aşağıda buna örnek olarak ID numarası 7 olan “Test” karakteri silinmiştir:

e.4. Güncelleme İşlemi

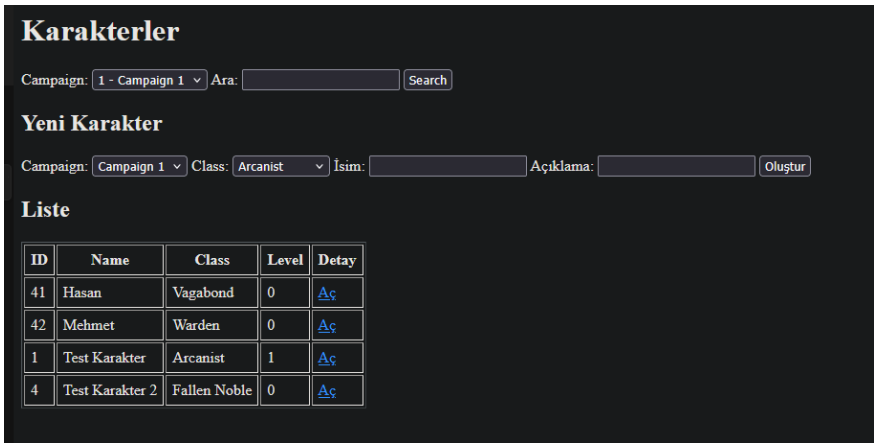
Güncelleme işlemine örnek olarak ID numarası 1 olan karakterin Class bilgisini güncelleyeceğiz. Öncelikle karakterin detay penceresini açıyoruz.



Detay penceresindeki "Class Değiştir" kısmındaki ComboBox ile ilgili karakterin Class bilgisi değiştirilir:



Class bilgisi değiştirilen 1 ID numaralı karakterin Class bilgisi başarılı bir şekilde değiştirilmiş olduğu karakter listesinden net bir şekilde görüntülenebilir:



f. Uygulamanın Kaynak Kodları

Uygulamanın kaynak kodlarına kapak sayfasında verilmiş olan GitHub reposunda (https://github.com/kuyash71/ttrpgDatabase) yer almaktadır. Aynı şekilde projenin SQL kodları ve ER diyagramı da GitHub reposunda yer almaktadır.

Sonuç:

Proje sonucunda, TTRPG oyunları için ölçeklenebilir ve yeniden kullanılabilir bir veritabanı mimarisi başarıyla geliştirilmiştir. Tasarlanan sistem; karakter oluşturma, sınıf bazlı istatistik uygulama, envanter yönetimi, arama işlemleri ve değişiklik kayıtlarının tutulması gibi temel oyun mekaniklerini desteklemektedir.

Stored procedure'ler kullanılarak karmaşık iş kuralları (örneğin sınıf seçimine bağlı otomatik stat atamaları ve eşya transferleri) merkezi bir yapıda toplanmış, trigger'lar ile istatistik ve kaynak sınırları denetlenmiş ve audit mekanizması sayesinde veri değişiklikleri kayıt altına alınmıştır. Bu sayede sistem hem güvenli hem de hataya dayanıklı hale getirilmiştir.

Geliştirilen veritabanı, ileride bir web veya masaüstü uygulaması ile entegre edilebilecek altyapıya sahiptir. Bu yönüyle proje, yalnızca bir akademik çalışma değil, aynı zamanda gerçek hayatta kullanılabilir bir TTRPG yönetim sistemi altyapısı sunmaktadır.