Database Documentation: Dead by Daylight

Table of Contents

1. Database Overview

The database is designed to track various entities related to the game "Dead by Daylight", a popular asymmetric survival horror game, acquired from the following website: https://dennisreep.nl/dbd/. The aim of the website is to store an inventory of items and features of the game (dim tables), along with the ratings of the player base on: (i) perk strength for different characters, (ii) addon strength for different characters (iii) map bias for survivors vs. killers (iv).

The purpose of the database is to store, manage, and analyse data related to the: (i) characters, (ii) game elements (maps, addons, perks), and (iii) match details. It utilises a combination of staging tables, dimension tables, and fact tables to efficiently organise and analyse the game's data.

This database and the analysis that arises from it, will support data-driven decision making for game balancing, player behaviour analysis, and player performance tracking.

2. Conceptual Model

The Dead by Daylight database conceptually revolves around key game elements such as characters (killers and survivors), perks, maps, and match outcomes. It aims to capture the relationships between these elements and store relevant attributes for analysis and game balance.

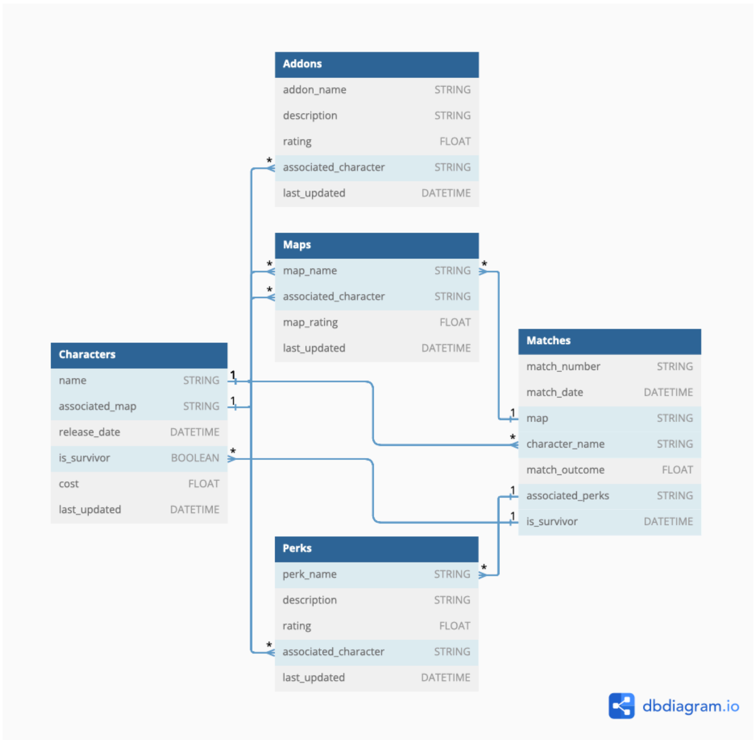
2.1. Overview

The database captures information about the following main entity groups:

1. Characters (Killers and Survivors)
2. Game Elements (Maps, Addons, Perks)
3. Match Data

2.2. High-Level Entity-Relationship Diagram

[Link]



2.3. Entities and Relationships

Key entities include Characters (Killers and Survivors), Perks, Maps, Add-ons, and Matches. Characters have perks and can participate in matches on various maps. Killers have specific add-ons and perks. Matches record outcomes and statistics for both killers and survivors.

1. Characters
- Includes both Killers and Survivors
- Attributes: name, tier, rating, survivor status, release date, licence status, associated map, DLC title, costs
- One character can participate in many matches
- One character can have many perks and addons

2. Maps
- Attributes: name, average score, bias, last updated date
- One map can be used in many matches
- Each map has ratings for different killers (1:M relationship)

3. Addons
- Specific to Killers in the current dataset, as the website being scraped doesn't yet have the rating feature for Survivor addons
- Attributes: name, description, tier, rating
- Many addons can belong to one killer (M:1 relationship)

4. Perks
- Can be for Killers, Survivors, or both
- Attributes: name, description, source, tier, rating, survivor status, category
- Many perks can be used by many characters in a match (M:M relationship)

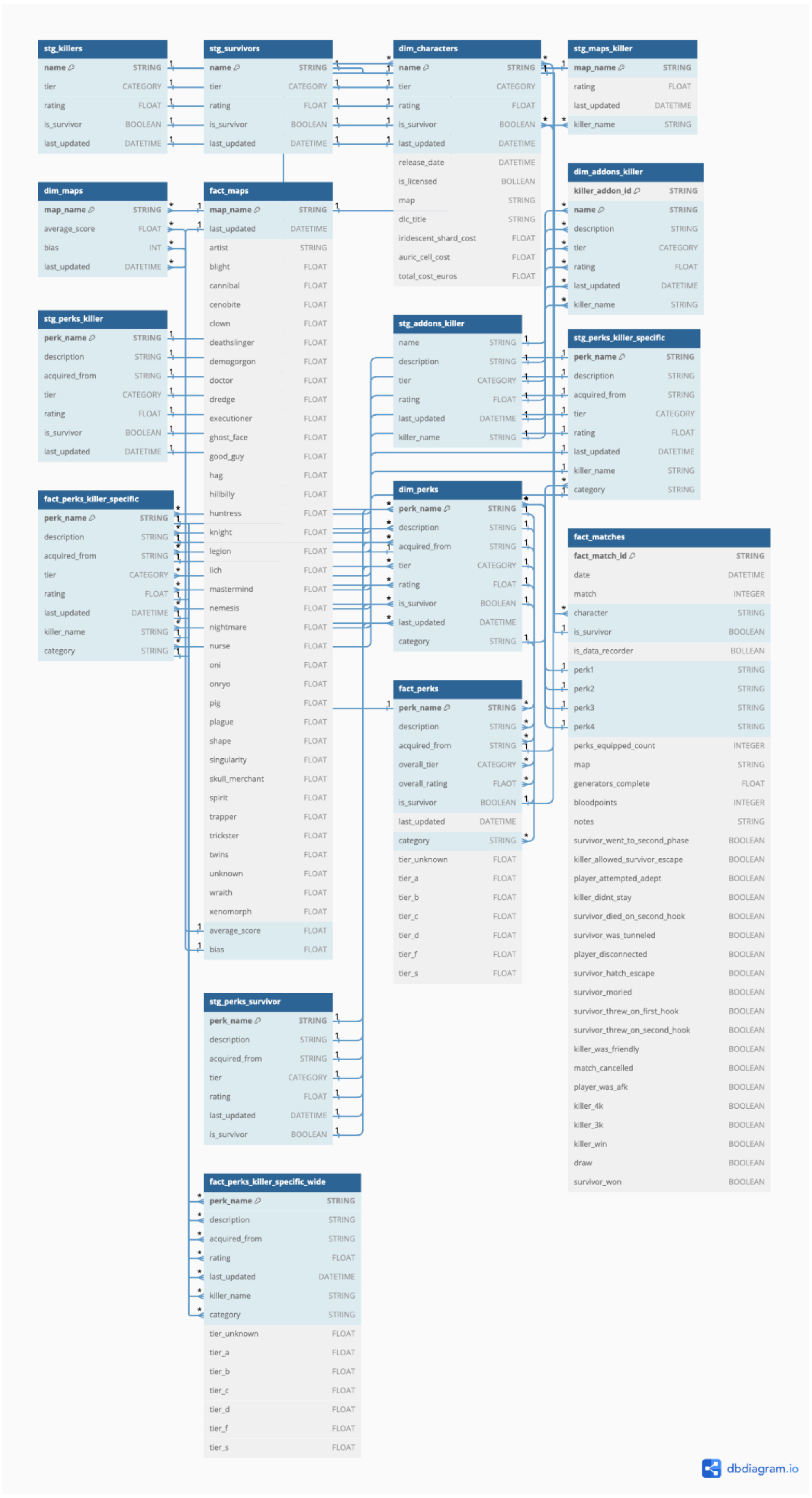
5. Matches
- Attributes: match ID, date, character, map, perks used, performance metrics
- Each match involves five characters (1 killer & 4 survivors) and one map
- Many perks can be used in many matches (M:M relationship)

3. Logical Model

Here, the database structure will be explained in more detail, including entity-relationship diagrams, table structures, and how the data is organised and related.

3.1. Entity-Relationship Diagram

[Link]



3.2. Tables and Attributes

The database consists of staging tables (prefixed with 'stg_'), dimension tables (prefixed with 'dim_'), and fact tables. Key tables include: dim_characters, dim_maps, dim_perks, fact_maps, fact_perks, and fact_matches.

1. Staging Tables

a. stg_killers

- name (PK, STRING)
- tier (STRING)
- rating (FLOAT)
- is_survivor (BOOLEAN)
- last_updated (DATETIME)

b. stg_survivors

- name (PK, STRING)
- tier (STRING)
- rating (FLOAT)
- is_survivor (BOOLEAN)
- last_updated (DATETIME)

c. stg_maps_killer

- map_name (PK, STRING)
- rating (FLOAT)
- last_updated (DATETIME)
- killer_name (STRING)

d. stg_addons_killer

- name (STRING)
- description (STRING)
- tier (STRING)
- rating (FLOAT)
- last_updated (DATETIME)
- killer_name (STRING)

e. stg_perks_killer

- perk_name (PK, STRING)
- description (STRING)
- acquired_from (STRING)
- tier (STRING)
- rating (FLOAT)
- is_survivor (BOOLEAN)
- last_updated (DATETIME)

f. stg_perks_survivor

- perk_name (PK, STRING)
- description (STRING)
- acquired_from (STRING)
- tier (STRING)
- rating (FLOAT)
- is_survivor (BOOLEAN)
- last_updated (DATETIME)

g. stg_perks_killer_specific

- perk_name (PK, STRING)
- description (STRING)
- acquired_from (STRING)
- tier (STRING)
- rating (FLOAT)
- last_updated (DATETIME)
- killer_name (STRING)
- category (STRING)

2. Dimension Tables

a. dim_characters

- name (PK, STRING)
- tier (STRING)
- rating (FLOAT)
- is_survivor (BOOLEAN)
- last_updated (DATETIME)
- release_date (DATETIME)
- is_licensed (BOOLEAN)
- map (STRING)
- dlc_title (STRING)
- iridescent_shard_cost (FLOAT)
- auric_cell_cost (FLOAT)
- total_cost_euros (FLOAT)

b. dim_maps

- map_name (PK, STRING)
- average_score (FLOAT)
- bias (INT)
- last_updated (DATETIME)

c. dim_addons_killer

- killer_addon_id (PK, STRING)
- name (PK, STRING)
- description (STRING)
- tier (STRING)
- rating (FLOAT)
- last_updated (DATETIME)
- killer_name (STRING)

d. dim_perks

- perk_name (PK, STRING)
- description (STRING)
- acquired_from (STRING)
- tier (STRING)
- rating (FLOAT)
- is_survivor (BOOLEAN)
- last_updated (DATETIME)
- category (STRING)

3. Fact Tables

a. fact_maps

- map_name (PK, STRING)
- last_updated (DATETIME)
- [1 column for each killer's rating per killer: 37 columns total](FLOAT)
- average_score (FLOAT)
- bias (FLOAT)

b. fact_perks_killer_specific

- perk_name (PK, STRING)
- description (STRING)
- acquired_from (STRING)
- tier (STRING)
- rating (FLOAT)
- last_updated (DATETIME)
- killer_name (STRING)
- category (STRING)

c. fact_perks_killer_specific_wide

- perk_name (PK, STRING)
- description (STRING)
- acquired_from (STRING)
- rating (FLOAT)
- last_updated (DATETIME)
- killer_name (STRING)
- category (STRING)
- tier_unknown (FLOAT)
- tier_a (FLOAT)
- tier_b (FLOAT)
- tier_c (FLOAT)
- tier_d (FLOAT)
- tier_f (FLOAT)
- tier_s (FLOAT)

d. fact_perks

- perk_name (PK, STRING)
- description (STRING)
- acquired_from (STRING)
- overall_tier (STRING)
- overall_rating (FLOAT)
- is_survivor (BOOLEAN)
- last_updated (DATETIME)
- category (STRING)
- tier_unknown (FLOAT)
- tier_a (FLOAT)
- tier_b (FLOAT)
- tier_c (FLOAT)
- tier_d (FLOAT)
- tier_f (FLOAT)
- tier_s (FLOAT)

e. fact_matches

- fact_match_id (PK, STRING)
- date (DATETIME)
- match (INTEGER)
- character (STRING)
- is_survivor (BOOLEAN)
- is_data_recorder (BOOLEAN)
- perk1 (STRING)
- perk2 (STRING)
- perk3 (STRING)
- perk4 (STRING)
- perks_equipped_count (INTEGER)
- map (STRING)
- generators_complete (FLOAT)
- bloodpoints (INTEGER)
- notes (STRING)
- survivor_went_to_second_phase BOOLEAN
- killer_allowed_survivor_escape BOOLEAN
- player_attempted_adept BOOLEAN
- killer_didnt_stay BOOLEAN
- survivor_died_on_second_hook BOOLEAN
- survivor_was_tunneled BOOLEAN
- player_disconnected BOOLEAN
- survivor_hatch_escape BOOLEAN
- survivor_moried BOOLEAN
- survivor_threw_on_first_hook BOOLEAN
- survivor_threw_on_second_hook BOOLEAN
- killer_was_friendly BOOLEAN
- match_cancelled BOOLEAN
- player_was_afk BOOLEAN
- killer_4k BOOLEAN
- killer_3k BOOLEAN
- killer_win BOOLEAN
- draw BOOLEAN
- survivor_won BOOLEAN

3.3. Relationships

Relationships are primarily enforced through foreign key constraints.

stg_killers

- `name` - dim_characters.name (one-to-one: stg_killers.name - dim_characters.name)
  - Explanation: Each killer in the staging table corresponds to exactly one character in the dimension table, and vice versa.
- `tier` - dim_characters.tier (one-to-one: stg_killers.tier - dim_characters.tier)
  - Explanation: The tier of a killer in the staging table directly corresponds to the tier of the same character in the dimension table.
- `rating` - dim_characters.rating (one-to-one: stg_killers.rating - dim_characters.rating)
  - Explanation: The rating of a killer in the staging table directly corresponds to the rating of the same character in the dimension table.
- `is_survivor` - dim_characters.is_survivor (one-to-one: stg_killers.is_survivor - dim_characters.is_survivor)
  - Explanation: The is_survivor flag for a killer in the staging table directly corresponds to the same flag for the character in the dimension table.

stg_survivors

- `name` - dim_characters.name (one-to-one: stg_killers.name - dim_characters.name)
  - Explanation: Each killer in the staging table corresponds to exactly one character in the dimension table, and vice versa.
- `tier` - dim_characters.tier (one-to-one: stg_killers.tier - dim_characters.tier)
  - Explanation: The tier of a killer in the staging table directly corresponds to the tier of the same character in the dimension table.
- `rating` - dim_characters.rating (one-to-one: stg_killers.rating - dim_characters.rating)
  - Explanation: The rating of a killer in the staging table directly corresponds to the rating of the same character in the dimension table.
- `is_survivor` - dim_characters.is_survivor (one-to-one: stg_killers.is_survivor - dim_characters.is_survivor)
  - Explanation: The is_survivor flag for a killer in the staging table directly corresponds to the same flag for the character in the dimension table.

stg_maps_killer

- `map_name` - fact_maps.map_name (many-to-one: stg_maps_killer.map_name > fact_maps.map_name)
  - Explanation: Multiple entries in stg_maps_killer can refer to the same map in fact_maps, as different killers have ratings for the same map.
- `killer_name` - dim_characters.name (many-to-one: stg_maps_killer.killer_name > dim_characters.name)
  - Explanation: Each killer name in stg_maps_killer refers to one character in dim_characters, but a character can have multiple map entries.

dim_perks

- `perk_name` - fact_perks.perk_name (one-to-one: dim_perks.perk_name - fact_perks.perk_name)
  - Explanation: A single perk in dim_perks appears once in fact_perks.
- `perk_name` - fact_perks_killer_specific.perk_name (one-to-many: dim_perks.perk_name < fact_perks_killer_specific.perk_name)
  - Explanation: A single perk in dim_perks can have multiple entries in fact_perks_killer_specific, as it can be associated with different killers.
- `perk_name` - fact_perks_killer_specific_wide.perk_name (one-to-one: dim_perks.perk_name - fact_perks_killer_specific_wide.perk_name)
  - Explanation: A single perk in dim_perks appears once in the wide table.
- `is_survivor` - fact_perks.is_survivor (one-to-one: dim_perks.is_survivor - fact_perks.is_survivor)
  - Explanation: The is_survivor flag for a perk in the dimension table directly corresponds to the same flag in the fact table.

stg_perks_killer_specific

- `perk_name` - fact_perks_killer_specific.perk_name (many-to-many: stg_perks_killer_specific.perk_name <> fact_perks_killer_specific.perk_name)
  - Explanation: A perk name is duplicated once per killer. The perks between the staging and fact tables can then be many to many.
- `description` - fact_perks_killer_specific.description (many-to-many: stg_perks_killer_specific.description <> fact_perks_killer_specific.description)
  - Explanation: The perk description is duplicated once per killer. The descriptions between the staging and fact tables can then be many to many.
- `acquired_from` - fact_perks_killer_specific.acquired_from (many-to-many: stg_perks_killer_specific.acquired_from <> fact_perks_killer_specific.acquired_from)
  - Explanation: The source of a perk in the staging table can be associated with multiple entries in the fact table, as it can be used by different killers, and vice versa.
- `killer_name` - fact_perks_killer_specific.killer_name (many-to-many: stg_perks_killer_specific.killer_name <> fact_perks_killer_specific.killer_name)
  - Explanation: A killer name in the staging table can be associated with multiple perk entries in the fact table, and vice versa.
- `category` - fact_perks_killer_specific.category (many-to-many: stg_perks_killer_specific.category < fact_perks_killer_specific.category)
  - Explanation: The category of a perk in the staging table can be associated with multiple entries in the fact table, and vice versa.

fact_perks_killer_specific

- `perk_name` - dim_perks.perk_name (many-to-one: fact_perks_killer_specific.perk_name > dim_perks.perk_name)
  - Explanation: Multiple entries in fact_perks_killer_specific can refer to the same perk in dim_perks, as a perk can be used by multiple killers.
- `perk_name` - fact_perks_killer_specific_wide.perk_name (many-to-one: fact_perks_killer_specific.perk_name <> fact_perks_killer_specific_wide.perk_name)
  - Explanation: Each perk in fact_perks_killer_specific_wide can have multiple entries in the non-wide table and vice versa.
- `description` - fact_perks_killer_specific_wide.description (many-to-many: fact_perks_killer_specific.description <> fact_perks_killer_specific_wide.description)

    - Explanation: Each description in fact_perks_killer_specific_wide can have multiple entries in the non-wide table, and vice versa.
- `acquired_from` - fact_perks_killer_specific_wide.acquired_from (many-to-many: fact_perks_killer_specific.acquired_from <> fact_perks_killer_specific_wide.acquired_from)
    - Explanation: Each acquired_from in fact_perks_killer_specific_wide can have multiple entries in the non-wide table, and vice versa.
- `killer_name` - fact_perks_killer_specific_wide.killer_name (many-to-many: fact_perks_killer_specific.killer_name <> fact_perks_killer_specific_wide.killer_name)
    - Explanation: A killer name in the wide table can be associated with multiple killer name entries in the fact table, and vice versa.
- `category` - fact_perks_killer_specific_wide.category (many-to-many: fact_perks_killer_specific.category <> fact_perks_killer_specific_wide.category)
    - Explanation: The category of a perk in the fact table can be associated with multiple entries in the wide table, possibly representing different categorization schemes or contexts.

fact_perks

- `acquired_from` - dim_characters.name (many-to-one: fact_perks.acquired_from > dim_characters.name)
    - Explanation: Multiple perks can be acquired from the same character, but each perk is acquired from only one character.
- `is_survivor` - dim_characters.is_survivor (many-to-one: fact_perks.is_survivor > dim_characters.is_survivor)
    - Explanation: Multiple perks can have the same is_survivor flag, corresponding to whether they are survivor or killer perks, but each perk has only one such flag.
- `perk_name` - fact_perks_killer_specific.perk_name (one-to-many: fact_perks.perk_name < fact_perks_killer_specific.perk_name)
    - Explanation: A single perk in fact_perks can have multiple entries in fact_perks_killer_specific, as it can be associated with different killers or contexts.

stg_addons_killer

- `name` - dim_addons_killer.name (many-to-many: stg_addons_killer.name <> dim_addons_killer.name)
    - Explanation: Each addon in the staging table corresponds to potentially more than one addon in the dimension table. Mostly follows a one-to-one relationship, except for 2 addon_names, which have a many-to-many relationship.
- `description` - dim_addons_killer.description (one-to-one: stg_addons_killer.description - dim_addons_killer.description)
    - Explanation: The description of an addon in the staging table directly corresponds to the description in the dimension table. Each description is unique.
- `killer_name` - dim_addons_killer.killer_name (many-to-many: stg_addons_killer.killer_name <> dim_addons_killer.killer_name)
    - Explanation: There are many killer names associated with an addon in the staging table, and they correspond to many killer_names in the dimension_table.
- `killer_name` - dim_characters.name (many-to-one: stg_addons_killer.killer_name > dim_characters.name)
    - Explanation: Multiple addons can belong to the same killer, but each addon belongs to only one killer.

fact_matches

- `perk1` - dim_perks.perk_name (many-to-one: fact_matches.perk1 > dim_perks.perk_name)
    - Explanation: Many matches can use the same perk in the first perk slot, but each perk1 slot in a match refers to only one specific perk.
- `perk2` - dim_perks.perk_name (many-to-one: fact_matches.perk2 > dim_perks.perk_name)
    - Explanation: Many matches can use the same perk in the second perk slot, but each perk2 slot in a match refers to only one specific perk.
- `perk3` - dim_perks.perk_name (many-to-one: fact_matches.perk3 > dim_perks.perk_name)
    - Explanation: Many matches can use the same perk in the third perk slot, but each perk3 slot in a match refers to only one specific perk.
- `perk4` - dim_perks.perk_name (many-to-one: fact_matches.perk4 > dim_perks.perk_name)
    - Explanation: Many matches can use the same perk in the fourth perk slot, but each perk4 slot in a match refers to only one specific perk.
- `character` - dim_characters.name (many-to-one: fact_matches.character > dim_characters.name)
    - Explanation: Many matches can involve the same character, however, they all point back to the same character in the character table.
- `is_survivor` - dim_characters.is_survivor (many-to-many: fact_matches.is_survivor <> dim_characters.is_survivor)
    - Explanation: Many matches can involve the same character, and multiple players could choose to play the same character. Each character is associated with either is_survivor == 1 or is_survivor ==0. As such, the same relationship applies to is_survivor.

fact_maps

- `map_name` - dim_maps.map_name (one-to-one: fact_maps.map_name - dim_maps.map_name)
    - Explanation: One map name in the dimension table corresponds to a map in the fact_table

stg_perks_killer

- `perk_name` - dim_perks.perk_name (one-to-one: stg_perks_killer.perk_name - dim_perks.perk_name)
    - Explanation: One entry in the staging table can refer to the same perk in the dimension table.
- `description` - dim_perks.description (one-to-one: stg_perks_killer.description - dim_perks.description)
    - Explanation: One description in the staging table has the same description in the dimension table.
- `acquired_from` - dim_perks.acquired_from (many-to-many: stg_perks_killer.acquired_from <> dim_perks.acquired_from)
    - Explanation: Multiple perks in the staging table can be acquired from the same source, this is also true for the acquired from in the dimension table.
- `is_survivor` - dim_perks.is_survivor (many-to-many: stg_perks_killer.is_survivor <> dim_perks.is_survivor)
    - Explanation: Multiple perks in the staging table can be acquired from the same source, this is also true for the acquired from in the dimension table. Each acquired_from source has their equivalent is_survivor field. The same logic, thus, applies to this field too.

stg_perks_survivor

- `perk_name` - dim_perks.perk_name (one-to-one: stg_perks_killer.perk_name - dim_perks.perk_name)
    - Explanation: One entry in the staging table can refer to the same perk in the dimension table.
- `description` - dim_perks.description (one-to-one: stg_perks_killer.description - dim_perks.description)
    - Explanation: One description in the staging table has the same description in the dimension table.
- `acquired_from` - dim_perks.acquired_from (many-to-many: stg_perks_killer.acquired_from <> dim_perks.acquired_from)
    - Explanation: Multiple perks in the staging table can be acquired from the same source, this is also true for the acquired from in the dimension table.
- `is_survivor` - dim_perks.is_survivor (many-to-many: stg_perks_killer.is_survivor <> dim_perks.is_survivor)
    - Explanation: Multiple perks in the staging table can be acquired from the same source, this is also true for the acquired from in the dimension table. Each acquired_from source has their equivalent is_survivor field. The same logic, thus, applies to this field too.

3.4. Normalisation Level

The database schema primarily adheres to the *Third Normal Form (3NF)*, while incorporating elements of *Dimensional Modelling* (Kimball and Ross, 2013) to facilitate efficient data analysis and reporting.

3.4.1. Third Normal Form (3NF) Compliance

The schema largely follows 3NF principles:

- All tables have clearly defined primary keys.
- Non-key attributes are fully functionally dependent on the primary key.
- There are no transitive dependencies of non-prime attributes on the primary key.

This is evident in the separation of entities such as characters, maps, perks, and addons into distinct tables, with relationships established through foreign keys.

3.4.2. Dimensional Modelling Elements

While maintaining 3NF principles, the schema also incorporates *Dimensional Modelling* concepts:

- **Dimension tables:** dim_characters, dim_maps, dim_addons_killer, and dim_perks contain descriptive attributes for their respective entities.
- **Fact tables:** fact_maps, fact_perks_killer_specific, fact_perks_killer_specific_wide, fact_perks, and fact_matches contain measures and foreign keys linking to dimension tables.

3.4.3. Deviations from strict normalisation

Some aspects of the schema deviate from strict normalisation for practical purposes:

- **Staging Tables:** The stg_* tables are intentionally denormalized to facilitate data ingestion.
- **Wide Fact Tables:** fact_maps and fact_matches have numerous columns, which is typical in dimensional modelling but diverges from strict normalisation.
- **Denormalized Structures:** fact_perks_killer_specific_wide and fact_perks include denormalized tier columns (tier_unknown, tier_a, tier_b, etc.) to enhance reporting and analysis capabilities.

3.4.4. Rationale

This hybrid approach, combining 3NF principles with Dimensional Modelling elements, aims to balance data integrity with query performance and ease of reporting. It allows for efficient data storage and maintenance while also facilitating complex analytical queries typical in game analytics scenarios.

4. Physical Model

Here is an example of how the database structure could be replicated in a database management system using SQL:

```
-- Staging Tables
CREATE TABLE stg_killers (
      name VARCHAR PRIMARY KEY,
      tier VARCHAR,
      rating FLOAT CHECK (rating BETWEEN 0 AND 5),
      is_survivor BOOLEAN DEFAULT FALSE CHECK (is_survivor = FALSE),
      last_updated DATETIME CHECK (last_updated <= CURRENT_TIMESTAMP)
);

CREATE TABLE stg_survivors (
      name VARCHAR PRIMARY KEY,
      tier VARCHAR,
      rating FLOAT CHECK (rating BETWEEN 0 AND 5),
      is_survivor BOOLEAN DEFAULT TRUE CHECK (is_survivor = TRUE),
      last_updated DATETIME CHECK (last_updated <= CURRENT_TIMESTAMP)
);

CREATE TABLE stg_maps_killer (
      map_name VARCHAR PRIMARY KEY,
      rating FLOAT CHECK (rating BETWEEN 0 AND 5),
      last_updated DATETIME CHECK (last_updated <= CURRENT_TIMESTAMP),
      killer_name VARCHAR
);

CREATE TABLE stg_addons_killer (
      name VARCHAR PRIMARY KEY,
      description TEXT,
      tier VARCHAR,
      rating FLOAT CHECK (rating BETWEEN 0 AND 5),
      last_updated DATETIME CHECK (last_updated <= CURRENT_TIMESTAMP),
      killer_name VARCHAR
);

CREATE TABLE stg_perks_killer (
      perk_name VARCHAR PRIMARY KEY,
      description TEXT,
      acquired_from VARCHAR,
      tier VARCHAR,
      rating FLOAT CHECK (rating BETWEEN 0 AND 5),
      is_survivor BOOLEAN DEFAULT FALSE CHECK (is_survivor = FALSE),
      last_updated DATETIME CHECK (last_updated <= CURRENT_TIMESTAMP)
```

```sql
);

CREATE TABLE stg_perks_survivor (
	perk_name VARCHAR PRIMARY KEY,
	description TEXT,
	acquired_from VARCHAR,
	tier VARCHAR,
	rating FLOAT CHECK (rating BETWEEN 0 AND 5),
	last_updated DATETIME CHECK (last_updated <= CURRENT_TIMESTAMP),
	is_survivor BOOLEAN DEFAULT TRUE CHECK (is_survivor = TRUE)
);

CREATE TABLE stg_perks_killer_specific (
	perk_name VARCHAR PRIMARY KEY,
	description TEXT,
	acquired_from VARCHAR,
	tier VARCHAR,
	rating FLOAT CHECK (rating BETWEEN 0 AND 5),
	last_updated DATETIME CHECK (last_updated <= CURRENT_TIMESTAMP),
	killer_name VARCHAR,
	category VARCHAR
);

-- Dimension Tables
CREATE TABLE dim_characters (
	name VARCHAR PRIMARY KEY,
	tier VARCHAR,
	rating FLOAT CHECK (rating BETWEEN 0 AND 5),
	is_survivor BOOLEAN,
	last_updated DATETIME CHECK (last_updated <= CURRENT_TIMESTAMP),
	release_date DATETIME CHECK (release_date <= CURRENT_TIMESTAMP),
	is_licensed BOOLEAN,
	map VARCHAR,
	dlc_title VARCHAR,
	iridescent_shard_cost FLOAT,
	auric_cell_cost FLOAT,
	total_cost_euros FLOAT
);

CREATE TABLE dim_maps (
	map_name VARCHAR PRIMARY KEY,
	average_score FLOAT CHECK (average_score BETWEEN 0 AND 10),
	bias INT CHECK (bias BETWEEN -1 AND 1),
	last_updated DATETIME CHECK (last_updated <= CURRENT_TIMESTAMP)
);

CREATE TABLE dim_addons_killer (
	killer_addon_id VARCHAR PRIMARY KEY,
	name VARCHAR,
	description TEXT,
	tier VARCHAR,
      rating FLOAT CHECK (rating BETWEEN 0 AND 10),
      last_updated DATETIME CHECK (last_updated <= CURRENT_TIMESTAMP),
      killer_name VARCHAR
);

CREATE TABLE dim_perks (
      perk_name VARCHAR PRIMARY KEY,
      description TEXT,
      acquired_from VARCHAR,
      tier VARCHAR,
      rating FLOAT CHECK (rating BETWEEN 0 AND 10),
      is_survivor BOOLEAN,
      last_updated DATETIME CHECK (last_updated <= CURRENT_TIMESTAMP),
      category VARCHAR
);

-- Fact Tables
CREATE TABLE fact_maps (
      map_name VARCHAR PRIMARY KEY,
      last_updated DATETIME CHECK (last_updated <= CURRENT_TIMESTAMP),
      artist FLOAT CHECK (artist BETWEEN -1 AND 1),
      blight FLOAT CHECK (blight BETWEEN -1 AND 1),
      cannibal FLOAT CHECK (cannibal BETWEEN -1 AND 1),
      cenobite FLOAT CHECK (cenobite BETWEEN -1 AND 1),
      clown FLOAT CHECK (clown BETWEEN -1 AND 1),
      deathslinger FLOAT CHECK (deathslinger BETWEEN -1 AND 1),
      demogorgon FLOAT CHECK (demogorgon BETWEEN -1 AND 1),
      doctor FLOAT CHECK (doctor BETWEEN -1 AND 1),
      dredge FLOAT CHECK (dredge BETWEEN -1 AND 1),
      executioner FLOAT CHECK (executioner BETWEEN -1 AND 1),
      ghost_face FLOAT CHECK (ghost_face BETWEEN -1 AND 1),
      good_guy FLOAT CHECK (good_guy BETWEEN -1 AND 1),
      hag FLOAT CHECK (hag BETWEEN -1 AND 1),
      hillbilly FLOAT CHECK (hillbilly BETWEEN -1 AND 1),
      huntress FLOAT CHECK (huntress BETWEEN -1 AND 1),
      knight FLOAT CHECK (knight BETWEEN -1 AND 1),
      legion FLOAT CHECK (legion BETWEEN -1 AND 1),
      lich FLOAT CHECK (lich BETWEEN -1 AND 1),
      mastermind FLOAT CHECK (mastermind BETWEEN -1 AND 1),
      nemesis FLOAT CHECK (nemesis BETWEEN -1 AND 1),
      nightmare FLOAT CHECK (nightmare BETWEEN -1 AND 1),
      nurse FLOAT CHECK (nurse BETWEEN -1 AND 1),
      oni FLOAT CHECK (oni BETWEEN -1 AND 1),
      onryo FLOAT CHECK (onryo BETWEEN -1 AND 1),
      pig FLOAT CHECK (pig BETWEEN -1 AND 1),
      plague FLOAT CHECK (plague BETWEEN -1 AND 1),
      shape FLOAT CHECK (shape BETWEEN -1 AND 1),
      singularity FLOAT CHECK (singularity BETWEEN -1 AND 1),
      skull_merchant FLOAT CHECK (skull_merchant BETWEEN -1 AND 1),
      spirit FLOAT CHECK (spirit BETWEEN -1 AND 1),
      trapper FLOAT CHECK (trapper BETWEEN -1 AND 1),
      trickster FLOAT CHECK (trickster BETWEEN -1 AND 1),
      twins FLOAT CHECK (twins BETWEEN -1 AND 1),
      unknown FLOAT CHECK (unknown BETWEEN -1 AND 1),
      wraith FLOAT CHECK (wraith BETWEEN -1 AND 1),
      xenomorph FLOAT CHECK (xenomorph BETWEEN -1 AND 1),
      average_score FLOAT CHECK (average_score BETWEEN -1 AND 1),
      bias FLOAT CHECK (bias BETWEEN -1 AND 1),
      FOREIGN KEY (map_name) REFERENCES dim_maps(map_name)
);

CREATE TABLE fact_perks_killer_specific (
      perk_name VARCHAR PRIMARY KEY,
      description TEXT,
      acquired_from VARCHAR,
      tier VARCHAR,
      rating FLOAT CHECK (rating BETWEEN 0 AND 5),
      last_updated DATETIME CHECK (last_updated <= CURRENT_TIMESTAMP),
      killer_name VARCHAR,
      category VARCHAR,
      FOREIGN KEY (perk_name) REFERENCES dim_perks(perk_name),
      FOREIGN KEY (killer_name) REFERENCES dim_characters(name)
);

CREATE TABLE fact_perks_killer_specific_wide (
      perk_name VARCHAR PRIMARY KEY,
      description TEXT,
      acquired_from VARCHAR,
      rating FLOAT CHECK (rating BETWEEN 0 AND 10),
      last_updated DATETIME CHECK (last_updated <= CURRENT_TIMESTAMP),
      killer_name VARCHAR,
      category VARCHAR,
      tier_unknown FLOAT CHECK (tier_unknown BETWEEN 0 AND 1),
      tier_a FLOAT CHECK (tier_a BETWEEN 0 AND 1),
      tier_b FLOAT CHECK (tier_b BETWEEN 0 AND 1),
      tier_c FLOAT CHECK (tier_c BETWEEN 0 AND 1),
      tier_d FLOAT CHECK (tier_d BETWEEN 0 AND 1),
      tier_f FLOAT CHECK (tier_f BETWEEN 0 AND 1),
      tier_s FLOAT CHECK (tier_s BETWEEN 0 AND 1),
      FOREIGN KEY (perk_name) REFERENCES dim_perks(perk_name),
      FOREIGN KEY (killer_name) REFERENCES dim_characters(name)
);

CREATE TABLE fact_perks (
      perk_name VARCHAR PRIMARY KEY,
      description TEXT,
      acquired_from VARCHAR,
      overall_tier VARCHAR,
      overall_rating FLOAT CHECK (overall_rating BETWEEN 0 AND 5),
      is_survivor BOOLEAN,
      last_updated DATETIME CHECK (last_updated <= CURRENT_TIMESTAMP),
      category VARCHAR,
```

```sql
      tier_unknown FLOAT,
      tier_a FLOAT,
      tier_b FLOAT,
      tier_c FLOAT,
      tier_d FLOAT,
      tier_f FLOAT,
      tier_s FLOAT,
      FOREIGN KEY (perk_name) REFERENCES dim_perks(perk_name),
      FOREIGN KEY (acquired_from) REFERENCES dim_characters(name)
);

CREATE TABLE fact_matches (
      fact_match_id VARCHAR PRIMARY KEY,
      date DATETIME CHECK (date <= CURRENT_TIMESTAMP),
      match INTEGER,
      character VARCHAR,
      is_survivor BOOLEAN,
      is_data_recorder BOOLEAN,
      perk1 VARCHAR,
      perk2 VARCHAR,
      perk3 VARCHAR,
      perk4 VARCHAR,
      perks_equipped_count INTEGER CHECK (perks_equipped_count BETWEEN 0 AND 4),
      map VARCHAR,
      generators_complete FLOAT CHECK (generators_complete BETWEEN 0 AND 5),
      bloodpoints INTEGER CHECK (bloodpoints >= 0),
      notes VARCHAR(500),
      survivor_went_to_second_phase BOOLEAN,
      killer_allowed_survivor_escape BOOLEAN,
      player_attempted_adept BOOLEAN,
      killer_didnt_stay BOOLEAN,
      survivor_died_on_second_hook BOOLEAN,
      survivor_was_tunneled BOOLEAN,
      player_disconnected BOOLEAN,
      survivor_hatch_escape BOOLEAN,
      survivor_moried BOOLEAN,
      survivor_threw_on_first_hook BOOLEAN,
      survivor_threw_on_second_hook BOOLEAN,
      killer_was_friendly BOOLEAN,
      match_cancelled BOOLEAN,
      player_was_afk BOOLEAN,
      killer_4k BOOLEAN,
      killer_3k BOOLEAN,
      killer_win BOOLEAN,
      draw BOOLEAN,
      survivor_won BOOLEAN
);
```

5. Data Dictionary

General Terms

- **Character**: Refers to the entities in the game, either killers or survivors, each with unique abilities and attributes.
- **Killer**: One of the playable characters whose goal is to capture and sacrifice the survivors.
- **Survivor**: One of the playable characters whose goal is to escape from the killer by repairing generators and opening exit gates.
- **Match**: A single instance of gameplay involving one killer and up to four survivors. Survivors can be replaced by AI players, called Bots.
- **Bot:** An AI player that replaces a real-life player when they quit a match that hasn't yet finished, to allow the other players to continue the match in their absence.
- **Generator**: A mechanical device that survivors need to repair to power the exit gates and escape. It takes roughly 90 seconds to complete one generator. This time can be altered by perks, addons and other factors.
- **Hook**: An object used by killers to hang captured survivors. Survivors need to be rescued by other players or escape before reaching the second phase, which is entered into after a certain amount of time has passed.
- **Phase**: Refers to the stages a survivor goes through when hooked. The second phase is more critical and harder to escape from. There are 2 phases: 1st and 2nd.
- **Hatch**: An alternate escape route that appears when only one survivor remains in the game.
- **Mori**: A killer's ability to kill a survivor directly without needing to hook them first, often as a special offering.
- **Gate:** Once 5 generators have been repaired. Survivors are able to open one or both of two gates that appear on the map. To open the gate, they have to power the switch that opens it, which takes time.
- **Perk**: Special abilities that characters (both killers and survivors) can equip to gain advantages during a match.
- **Tier**: The ranking or level of a perk, which determines its effectiveness. Common tiers include S (highest), A, B, C, D, F (lowest). The tier-ratings are given to the perks by the player community.
- **Adept Challenge**: A challenge where players must complete a match using only the three unique perks of a specific character. There are game achievements or trophies tied to completing these.
- **Map**: The environment where the match takes place, featuring various structures, obstacles, and objectives.
- **Bias**: A measure of favorability or disadvantage inherent to a particular map for either killers or survivors.
- **Bloodpoints**: In-game currency earned by playing matches, used to unlock perks, items, and add-ons.
- **Iridescent Shards**: A currency earned by levelling up the player's account, used to purchase characters and cosmetics.
- **Auric Cells**: A premium currency bought with real money, used for purchasing characters, cosmetics, and other premium content.
- **Addon**: Items that can be equipped by killers or survivors to modify their abilities or offer additional effects during a match.
- **Throwing**: A verb for when a survivor or killer tries to end the game early by exploiting the in-game mechanics. It is usually done when the player believes the game to be no longer winnable.
- **AFK**: Indicates if the player was **away from the keyboard** during the match.
- **Killer 4K**: Indicates if the killer killed all four survivors.
- **Killer 3K**: Indicates if the killer killed three survivors.
- **Killer Win**: Indicates if the killer won the match: 3 or more survivors killed.
- **Draw**: Indicates if the match ended in a draw: 2 survivors killed.
- **Survivor Won**: Indicates if the survivor(s) won the match: 1 or fewer survivors killed.

Table: `stg_killers`

- **name** (STRING, pk): The name of the killer character.
- **tier** (CATEGORY): The tier ranking of the killer: S & A-F (with S being the highest and F the lowest).
- **rating** (FLOAT): The rating score of the killer. Given by the player community.
- **is_survivor** (BOOLEAN): Indicates if the character is a survivor (should always be false for this table).
- **last_updated** (DATETIME): The date and time when the data was last updated.

Table: `stg_survivors`

- **name** (STRING, pk): The name of the survivor character.
- **tier** (CATEGORY): The tier ranking of the survivor: S & A-F (with S being the highest and F the lowest).
- **rating** (FLOAT): The rating score of the survivor. Given by the player community.
- **is_survivor** (BOOLEAN): Indicates if the character is a survivor (should always be true for this table).
- **last_updated** (DATETIME): The date and time when the data was last updated.

Table: `dim_characters`

- **name** (STRING, pk): The name of the character (either killer or survivor).
- **tier** (CATEGORY): The tier ranking of the character: S & A-F (with S being the highest and F the lowest).
- **rating** (FLOAT): The rating score of the character. Given by the player community.
- **is_survivor** (BOOLEAN): Indicates if the character is a survivor.
- **last_updated** (DATETIME): The date and time when the data was last updated.
- **release_date** (DATETIME): The release date of the character, when they were first available to use by the player base.
- **is_licensed** (BOOLEAN): Indicates if the character is licensed from another franchise. Dead by Daylight works with many franchises to bring famous characters into their game.
- **map** (STRING): The associated map for the character. If a character is released as part of a downloadable content (DLC) package, there is often a map that comes with 2-3 characters.
- **dlc_title** (STRING): The downloadable content (DLC) title where the character is available.
- **iridescent_shard_cost** (FLOAT): The cost of the character in Iridescent Shards, an in-game currency.
- **auric_cell_cost** (FLOAT): The cost of the character in Auric Cells, an in-game currency.
- **total_cost_euros** (FLOAT): The total cost of the character in Euros, when converting the auric cell value.

Table: `stg_maps_killer`

- **map_name** (STRING, pk): The name of the map.
- **rating** (FLOAT): The rating score of the map: S & A-F (with S being the highest and F the lowest).
- **last_updated** (DATETIME): The date and time when the data was last updated.
- **killer_name** (STRING): The name of the killer associated with the map.

Table: `dim_maps`

- **map_name** (STRING, pk): The name of the map.
- **average_score** (FLOAT): The average score of the map. Given by the player base. -1 is 100% survivor friendly, 0 is 100% neutral and 1 is 100% killer friendly.
- **bias** (INT): The bias score associated with the map. -1 for survivor friendly, 0 for neutral and 1 for killer friendly.
- **last_updated** (DATETIME): The date and time when the data was last updated.

Table: `fact_maps`

- **map_name** (STRING, pk): The name of the map.
- **last_updated** (DATETIME): The date and time when the data was last updated.
- **artist** to **xenomorph** (FLOAT): Rating scores for each killer on the map. 1 column per killer. Given by the player community.
- **average_score** (FLOAT): The overall average score of the map. Averages the ratings from each killer column into 1 score.
- **bias** (FLOAT): The bias score associated with the map. -1 for survivor friendly, 0 for neutral and 1 for killer friendly.

Table: `stg_addons_killer`

- **name** (STRING): The name of the killer addon.
- **description** (STRING): The description of the killer addon.
- **tier** (CATEGORY): The tier ranking of the killer addon: S & A-F (with S being the highest and F the lowest).
- **rating** (FLOAT): The rating score of the killer addon. Given by the player base.
- **last_updated** (DATETIME): The date and time when the data was last updated.
- **killer_name** (STRING): The name of the killer associated with the addon.

Table: `dim_addons_killer`

- **killer_addon_id** (STRING, pk): The unique identifier for the killer addon.
- **name** (STRING, pk): The name of the killer addon.
- **description** (STRING): The description of the killer addon.
- **tier** (CATEGORY): The tier ranking of the killer addon: S & A-F (with S being the highest and F the lowest).
- **rating** (FLOAT): The rating score of the killer addon. Given by the player base.
- **last_updated** (DATETIME): The date and time when the data was last updated.
- **killer_name** (STRING): The name of the killer associated with the addon.

Table: `stg_perks_killer`

- **perk_name** (STRING, pk): The name of the killer perk.
- **description** (STRING): The description of the killer perk. That is, what it does.
- **acquired_from** (STRING): The source from which the perk can be acquired.
- **tier** (CATEGORY): The tier ranking of the killer perk: S & A-F (with S being the highest and F the lowest).
- **rating** (FLOAT): The rating score of the killer perk. Given by the player base.
- **is_survivor** (BOOLEAN): Indicates if the perk is for a survivor (should always be false for this table).
- **last_updated** (DATETIME): The date and time when the data was last updated.

Table: `stg_perks_survivor`

- **perk_name** (STRING, pk): The name of the survivor perk.
- **description** (STRING): The description of the survivor perk.
- **acquired_from** (STRING): The source from which the perk can be acquired.
- **tier** (CATEGORY): The tier ranking of the survivor perk: S & A-F (with S being the highest and F the lowest).
- **rating** (FLOAT): The rating score of the survivor perk. Given by the player base.
- **last_updated** (DATETIME): The date and time when the data was last updated.
- **is_survivor** (BOOLEAN): Indicates if the perk is for a survivor (should always be true for this table).

Table: `dim_perks`

- **perk_name** (STRING, pk): The name of the perk.
- **description** (STRING): The description of the perk.
- **acquired_from** (STRING): The source from which the perk can be acquired.
- **tier** (CATEGORY): The tier ranking of the perk: S & A-F (with S being the highest and F the lowest)
- **rating** (FLOAT): The rating score of the perk. Given by the player base.
- **is_survivor** (BOOLEAN): Indicates if the perk is for a survivor.
- **last_updated** (DATETIME): The date and time when the data was last updated.
- **category** (STRING): The category of the perk (e.g., killer, survivor).

Table: `stg_perks_killer_specific`

- **perk_name** (STRING, pk): The name of the killer-specific perk.
- **description** (STRING): The description of the killer-specific perk.
- **acquired_from** (STRING): The source from which the perk can be acquired.
- **tier** (CATEGORY): The tier ranking of the killer-specific perk: S & A-F (with S being the highest and F the lowest)
- **rating** (FLOAT): The rating score of the killer-specific perk. Given by the player base.
- **last_updated** (DATETIME): The date and time when the data was last updated.
- **killer_name** (STRING): The name of the killer associated with the perk.
- **category** (STRING): The category of the perk. Given by the player base.

Table: `fact_perks_killer_specific`

- **perk_name** (STRING, pk): The name of the killer-specific perk.
- **description** (STRING): The description of the killer-specific perk.
- **acquired_from** (STRING): The source from which the perk can be acquired.
- **tier** (CATEGORY): The tier ranking of the killer-specific perk: S & A-F (with S being the highest and F the lowest)
- **rating** (FLOAT): The rating score of the killer-specific perk. Given by the player base.
- **last_updated** (DATETIME): The date and time when the data was last updated.
- **killer_name** (STRING): The name of the killer associated with the perk.
- **category** (STRING): The category of the perk. Given by the player base.

Table: `fact_perks_killer_specific_wide`

- **perk_name** (STRING, pk): The name of the killer-specific perk.
- **description** (STRING): The description of the killer-specific perk.
- **acquired_from** (STRING): The source from which the perk can be acquired.
- **rating** (FLOAT): The rating score of the killer-specific perk. Given by the player base.
- **last_updated** (DATETIME): The date and time when the data was last updated.
- **killer_name** (STRING): The name of the killer associated with the perk.
- **category** (STRING): The category of the perk. Given by the player base.
- **tier_unknown** to **tier_s** (FLOAT): The rating scores for each tier category of the killer-specific perk. S is the highest, followed by A. Then it is alphabetical until F, the lowest. Unknown is given if there are no ratings provided by the player base.

Table: `fact_perks`

- **perk_name** (STRING, pk): The name of the perk.
- **description** (STRING): The description of the perk.
- **acquired_from** (STRING): The source from which the perk can be acquired.
- **overall_tier** (CATEGORY): The overall tier ranking of the perk. S is the highest, followed by A. Then it is alphabetical until F, the lowest. Unknown is given if there are no ratings provided by the player base.
- **overall_rating** (FLOAT): The overall rating score of the perk. Provided by the player base.
- **is_survivor** (BOOLEAN): Indicates if the perk is for a survivor.
- **last_updated** (DATETIME): The date and time when the data was last updated.
- **category** (STRING): The category of the perk.
- **tier_unknown** to **tier_s** (FLOAT): The rating scores for each tier category of the perk: S & A-F (with S being the highest and F the lowest)

Table: `fact_matches`

- **fact_match_id** (STRING, pk): The unique identifier for the match.
- **date** (DATETIME): The date and time of the match.
- **match** (INTEGER): The match number or identifier. Repeated 5 times per unique fact_match_id, one per individual player in a match (4 killers and 1 survivor).
- **character** (STRING): The name of the character used in the match.
- **is_survivor** (BOOLEAN): Indicates if the character is a survivor.
- **is_data_recorder** (BOOLEAN): Indicates if the player in question was played by the person who recorded the match data.
- **perk1** to **perk4** (STRING): The names of the perks equipped during the match.
- **perks_equipped_count** (INTEGER): The number of perks equipped during the match.
- **map** (STRING): The name of the map where the match took place.
- **generators_complete** (FLOAT): The number of generators completed in the match.
- **bloodpoints** (INTEGER): The number of bloodpoints earned in the match.
- **notes** (STRING): Any additional notes about the match.
- **survivor_went_to_second_phase** (BOOLEAN): Indicates if the survivor reached the second phase on the hook.
- **killer_allowed_survivor_escape** (BOOLEAN): Indicates if the killer allowed the survivor to escape.
- **player_attempted_adept** (BOOLEAN): Indicates if the player attempted an adept challenge.
- **killer_didnt_stay** (BOOLEAN): Indicates if the killer left the match early.
- **survivor_died_on_second_hook** (BOOLEAN): Indicates if the survivor died on their second hook.
- **survivor_was_tunneled** (BOOLEAN): Indicates if the survivor was tunneled. Tunnelling is the act of a killer trying to single out a specific survivor and kill them first, to remove them from the match, making it harder for the remaining survivors. Ideally, the killer is expected to hook each survivor equally (first, all once and once the survivors have been hooked once, then aim to hook them all a second time).
- **player_disconnected** (BOOLEAN): Indicates if the player disconnected during the match.
- **survivor_hatch_escape** (BOOLEAN): Indicates if the survivor escaped through the hatch.
- **survivor_moried** (BOOLEAN): Indicates if the survivor was killed using a Mori.
- **survivor_threw_on_first_hook** (BOOLEAN): Indicates if the survivor gave up on the first hook.
- **survivor_threw_on_second_hook** (BOOLEAN): Indicates if the survivor gave up on the second hook.
- **killer_was_friendly** (BOOLEAN): Indicates if the killer was friendly during the match.
- **match_cancelled** (BOOLEAN): Indicates if the match was cancelled.
- **player_was_afk** (BOOLEAN): Indicates if the player was away from the keyboard during the match.
- **killer_4k** (BOOLEAN): Indicates if the killer achieved a 4K (killed all four survivors).
- **killer_3k** (BOOLEAN): Indicates if the killer achieved a 3K (killed three survivors).
- **killer_win** (BOOLEAN): Indicates if the killer won the match (3 or more kills in a match).
- **draw** (BOOLEAN): Indicates if the match ended in a draw (2 survivors killed).
- **survivor_won** (BOOLEAN): Indicates if the survivor won the match (1 or fewer kills in the match).

6. Data Flow

6.1. Data integration

Currently, the data is scraped from the website: https://dennisreep.nl/dbd/.

This comes with the potential issue of instability: if anything were to change on the website, in regards to its structure, the data would potentially not pull. This could be overcome by using an API.

Additionally, the data on the website is refreshed monthly, and the previous month overwritten. As such, it would also be beneficial to:

- Re-run the script monthly
- Append the new data
- Update the pipeline accordingly to handle the above

As the current project wasn't intended to span such a long time period, this hasn't been taken into account. Rather, at present, the data tables will be overridden with the most current data when the pipeline is run.

6.2. Table types

The following table types are used:

- **Temporary Tables (temp):** Used to hold data temporarily during ETL processes for intermediate processing.
- **Staging Tables (stg):** Used to store raw data from source system before it is processed.
- **Dimension Tables (dim):** Used to store descriptive attributes that provide context and details for data analysis.
- **Fact Tables (fact):** Used to store quantitative data for analysis and reporting.

6.3. Flow

If there are more than two web pages needed per database, data moves into temporary tables first.

Data tables where there is one temporary table for each {killer}:

- temp_{killer}_maps
- temp_{killer}_addons

Where there are one or two web pages per database, the data is stored first in a staging table before being moved to a dimension or fact table.

Cleaning & transformation, along with feature engineering, are done directly on the dimension or fact table.This decision was made, so that we always have a record of the raw data (pre-cleaning), should we need to check it for any reason (data validation, etc.).

6.4. Cleaning

Data is semi-cleaned during the ingression stage, where the column headers are put into lowercase and spaces are removed.

All data tables are passed through a quality check (function: dataframe_report()) once they become a fact or dimension table.

Then, the type of cleaning the dataset undergoes depends on the outcome of the quality check.

- Duplicates are investigated and then removed where necessary.
- Blanks and nulls are either investigated, kept or handled.
- Data types are converted to the appropriate types.
    - Dates to datetime objects
    - Numerical values to float or int
    - Text values to string
    - True or false values to boolean

dim_characters

- Columns were renamed to lowercase and spaces were replaced with underscores.
- Data types were changed
    - There were 0 duplicates
    - The null or blank values are explainable
        - Not all characters are released with their own map
        - Not all characters have a dlc_title associated with them

dim_perks

- Columns were renamed to lowercase and spaces were replaced with underscores.
- Data types were changed
    - There were 3 duplicates
        - Three from the same killer
            - As a precaution, I will ensure that 'general' perks are assigned correctly
            - Then, I will removed the duplicates
    - There were 0 blanks
    - There were 0 nulls

fact_perks

- Columns were renamed to lowercase and spaces were replaced with underscores.
- Data types were changed
- Duplicate rows were dropped
    - There were 0 duplicates
    - There were 0 blanks
    - There were 0 nulls

fact_perks_killer_specific

- Columns were renamed to lowercase and spaces were replaced with underscores.
- Data types were changed
    - There were 0 duplicates
    - There were 0 blanks
    - There were 0 nulls

fact_perks_killer_specific_wide

- Columns were renamed to lowercase and spaces were replaced with underscores.
- Data types were changed
- Duplicate rows were dropped
    - There were 0 duplicates
    - There were 0 blanks
    - The nulls were explainable:

        - Not all character have a rating for their perks
        - The ratings are user generated. If no users have rated, then the result is null.

dim_addons_killer

- Columns were renamed to lowercase and spaces were replaced with underscores.
- Data types were changed
    - There were 0 duplicates
    - There were 0 blanks
    - There were 0 nulls

dim_maps

- Columns were renamed to lowercase and spaces were replaced with underscores.
- Data types were changed
    - There were 0 duplicates
    - There were 0 blanks
    - There were 0 nulls

fact_maps

- Columns were renamed to lowercase and spaces were replaced with underscores.
- Data types were changed
    - There were 88 duplicates
        - Which were removed.
        - The data table for 2 killers was duplicated in the ingression phase due to the website structure not matching the other killers'.
    - There were 0 blanks
    - There nulls were explainable:
        - The user scores are user-generated
        - Some maps are not rated for some killers
        - These are the nulls that are seen in the dataset.

fact_matches

The fact_match dataset was a special dataset, in that it wasn't scraped from the same website like the rest of the data but was rather a dataset created by a player of the game and published in a Google Sheets spreadsheet.

- Raw data
- Clean data

The raw data was very unusable at the beginning, as it was not in a normalised, tabular structure. The following cleaning operations were done:

In the spreadsheet: pre-load

- Restructured the data into tabular format
    - One row per player per match
    - Each column variable duplicated to reflect this where needed
        - Date values pulled down and duplicated once per survivor (previously, there was one date per match and then empty row values for the rest of the players)
        - Map values pulled down, once per survivor
        - Generator_don pulled down once per survivor
- Empty row separator between matches removed

- Initial date row separator before a new match was also removed
- Concatenated all of the monthly data that was stored in separate tabs (one per month) into one complete tab.
- Columns were renamed to lowercase and spaces were replaced with underscores.
- Perks were split out into separate columns.
    - Originally there was one column with an array of up to 4 perk names. This was separated out, with each perk having its own column (perk1 | perk2 | perk3 | perk4)

In python: post-load

- Data types were changed
- The nulls are explainable:
    - **perk2** = not all players equipped a 2nd perk
    - **perk3** = not all players equipped a 3rd perk
    - **perk4** = not all players equipped a 4th perk
    - **generators_complete** = 5 rows (1 match) missing due to the data recorder not entering this information (manual error).
        - No additional cleaning needed. We can leave blank rather than inputing.
    - **notes:** not all players for each match had something additional entered

6.5. Transformation

Depending on the type of analysis needed, new features needed to be added to datasets and existing features transformed.

Each dataset was looked at on an individual basis, based on the required information for the research questions.

dim_characters:

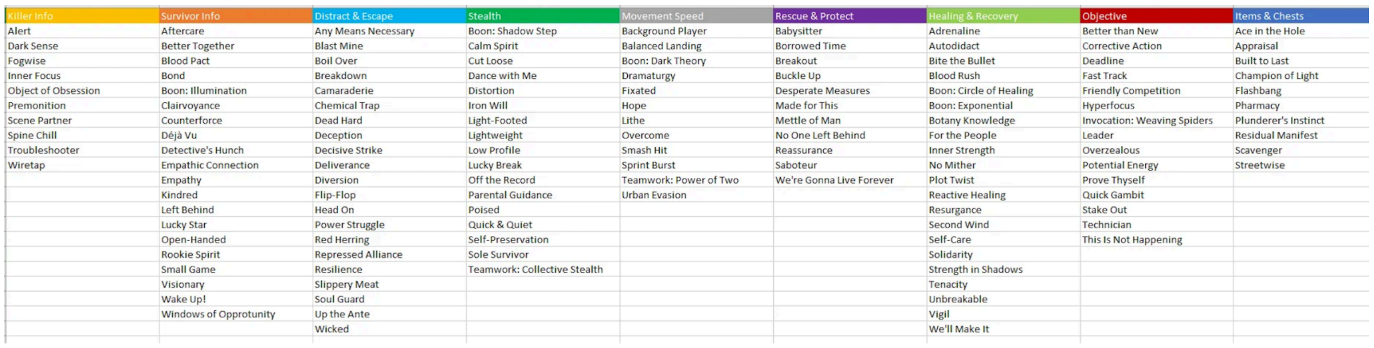
- Downloadable content data is added (dlc_data)
    - **Note:** This data came from the Wikipedia page for the game [link]. If more time was available, this could also be scraped and automated.
        - **release_date:** release date of the content
        - **title:** Title of the content
        - **map:** associated map name
        - **name:** character name
        - **is_survivor:** whether the character is a killer or a survivor
        - **is_licensed:** whether a character is licensed or not
- Cost information is added
    - **auric_cell_cost:** total cost of a character in auric cells
    - **iridescent_shard_cost:** total cost of a character in iridescent shards
    - **total_cost_euros:** total cost of a character in euros
        - Cost was difficult to automate.
        - There were several rules that determined the cost of a character.
        - Some characters were included with the base game.
        - Some characters have a permanent reduction applied to their cost.
        - Some characters are 'licensed' and others are not.
        - From there,
            - If licensed
                - Iridescent shard cost = 0
                - Auric cell cost = 500
            - If not licensed & before the discounted date
                - Iridescent shard cost = 4500
                - Auric cell cost = 250

- If not licensed & after the discounted date
    - Iridescent shard cost = 9000
    - Auric cell cost = 500
- This is then converted into euro value
    - conversion_rate_auric_to_iridescent = (9000 / 500) * iridescent cost of a character
        - That is, find the value of 1 iridescent cost in euros and then multiply that by the iridescent cost of a character.
    - conversion_rate_iridescent_to_euros = (4.99 / 9000) * auric cell cost of a character
        - That is, find the value of 1 auric cell cost in euros and then multiply that by the auric cell cost of a character.

dim_perks:

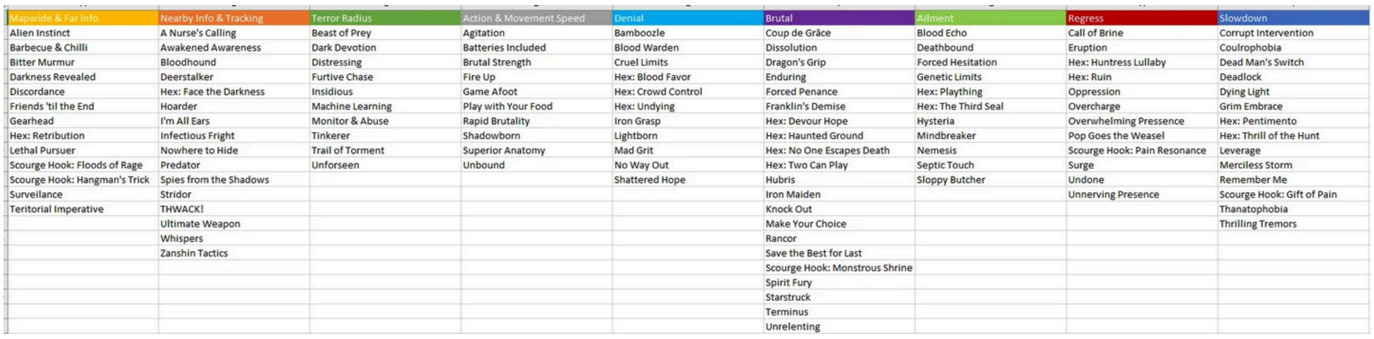
- Category data is added to classify perks.
    - Note: This data was manually added based on data collected from the player base [link].
        - category: The category a perk has. Helping to categorise its intended function.

Survivor perk categories



Killer Info	Survivor Info	Distract & Escape	Stealth	Movement Speed	Rescue & Protect	Healing & Recovery	Objective	Items & Chests
Alert	Aftercare	Any Means Necessary	Boon: Shadow Step	Background Player	Babysitter	Adrenaline	Better than New	Ace in the Hole
Dark Sense	Better Together	Blast Mine	Calm Spirit	Balanced Landing	Borrowed Time	Autodidact	Corrective Action	Appraisal
Fogwise	Blood Pact	Boil Over	Cut Loose	Boon: Dark Theory	Breakout	Bite the Bullet	Deadline	Built to Last
Inner Focus	Bond	Breakdown	Dance with Me	Dramaturgy	Buckle Up	Blood Rush	Fast Track	Champion of Light
Object of Obsession	Boon: Illumination	Camaraderie	Distortion	Fixated	Desperate Measures	Boon: Circle of Healing	Friendly Competition	Flashbang
Premonition	Clairvoyance	Chemical Trap	Iron Will	Hope	Made for This	Boon: Exponential	Hyperfocus	Pharmacy
Scene Partner	Counterforce	Dead Hard	Light-Footed	Lithe	Mettle of Man	Botany Knowledge	Invocation: Weaving Spiders	Plunderer's Instinct
Spine Chill	Déjà Vu	Deception	Lightweight	Overcome	No One Left Behind	For the People	Leader	Residual Manifest
Troubleshooter	Detective's Hunch	Decisive Strike	Low Profile	Smash Hit	Reassurance	Inner Strength	Overzealous	Scavenger
Wiretap	Empathic Connection	Deliverance	Lucky Break	Sprint Burst	Saboteur	No Mither	Potential Energy	Streetwise
	Empathy	Diversion	Off the Record	Teamwork: Power of Two	We're Gonna Live Forever	Plot Twist	Prove Thyself	
	Kindred	Flip-Flop	Parental Guidance	Urban Evasion		Reactive Healing	Quick Gambit	
	Left Behind	Head On	Poised			Resurgance	Stake Out	
	Lucky Star	Power Struggle	Quick & Quiet			Second Wind	Technician	
	Open-Handed	Red Herring	Self-Preservation			Self-Care	This Is Not Happening	
	Rookie Spirit	Repressed Alliance	Sole Survivor			Solidarity		
	Small Game	Resilience	Teamwork: Collective Stealth			Strength in Shadows		
	Visionary	Slippery Meat				Tenacity		
	Wake Up!	Soul Guard				Unbreakable		
	Windows of Opprotunity	Up the Ante				Vigil		
		Wicked				We'll Make It		

Killer perk categories



Mapwide & Far Info	Nearby Info & Tracking	Terror Radius	Action & Movement Speed	Denial	Brutal	Ailment	Regress	Slowdown
Alien Instinct	A Nurse's Calling	Beast of Prey	Agitation	Bamboozle	Coup de Grâce	Blood Echo	Call of Brine	Corrupt Intervention
Barbecue & Chilli	Awakened Awareness	Dark Devotion	Batteries Included	Blood Warden	Dissolution	Deathbound	Eruption	Coulrophobia
Bitter Murmur	Bloodhound	Distressing	Brutal Strength	Cruel Limits	Dragon's Grip	Forced Hesitation	Hex: Huntress Lullaby	Dead Man's Switch
Darkness Revealed	Deerstalker	Furtive Chase	Fire Up	Hex: Blood Favor	Enduring	Genetic Limits	Hex: Ruin	Deadlock
Discordance	Hex: Face the Darkness	Insidious	Game Afoot	Hex: Crowd Control	Forced Penance	Hex: Plaything	Oppression	Dying Light
Friends 'til the End	Hoarder	Machine Learning	Play with Your Food	Hex: Undying	Franklin's Demise	Hex: The Third Seal	Overcharge	Grim Embrace
Gearhead	I'm All Ears	Monitor & Abuse	Rapid Brutality	Iron Grasp	Hex: Devour Hope	Hysteria	Overwhelming Pressence	Hex: Pentimento
Hex: Retribution	Infectious Fright	Tinkerer	Shadowborn	Lightborn	Hex: Haunted Ground	Mindbreaker	Pop Goes the Weasel	Hex: Thrill of the Hunt
Lethal Pursuer	Nowhere to Hide	Trail of Torment	Superior Anatomy	Mad Grit	Hex: No One Escapes Death	Nemesis	Scourge Hook: Pain Resonance	Leverage
Scourge Hook: Floods of Rage	Predator	Unforseen	Unbound	No Way Out	Hex: Two Can Play	Septic Touch	Surge	Merciless Storm
Scourge Hook: Hangman's Trick	Spies from the Shadows			Shattered Hope	Hubris	Sloppy Butcher	Undone	Remember Me
Surveilance	Stridor				Iron Maiden		Unnerving Presence	Scourge Hook: Gift of Pain
Teritorial Imperative	THWACK!				Knock Out			Thanatophobia
	Ultimate Weapon				Make Your Choice			Thrilling Tremors
	Whispers				Rancor			
	Zanshin Tactics				Save the Best for Last			
					Scourge Hook: Monstrous Shrine			
					Spirit Fury			
					Starstruck			
					Terminus			
					Unrelenting			

fact_perks

- Tier ratings are converted into dummy columns for each killer, for aggregation purposes
- Then, the tier ratings are summed for each perk and each killer.
    - Example: Blight = (Tier) A would become
        - Tier_unknown = 0
        - Tier_S = 0
        - Tier_A = 1
        - Tier_B = 0
        - Tier_C = 0
        - Tier_D = 0

        - Tier_F = 0
- Tier ratings are summed per perk to give the total sum of each rating for the perk in question.

fact_perks_killer_specific_wide

- Tier ratings are converted into dummy columns for each killer, for aggregation purposes
- Then, the tier ratings are summed for each perk and each killer.
    - Example: Blight = (Tier) A would become
        - Tier_unknown = 0
        - Tier_S = 0
        - Tier_A = 1
        - Tier_B = 0
        - Tier_C = 0
        - Tier_D = 0
        - Tier_F = 0

dim_addons_killer

- Primary key is added, as this table is one of the few tables where the name column (addon_name) isn't entirely unique (2 rows use the same name)
    - DAK + INT + {killer_name}
        - Table acronym + 3 digit unique incremental int + {killer_name}
        - Example: DAK001blight
- Updated the acquired_from column to 'all_killers' for the default starting perks.
    - Perks that are accessible by all killers from the start of the game.

fact_maps

- One map is missing from the data pull (Léry's Memorial Institute), which is added to the table with NA values for the additional columns.
- Rating is added for each individual killer. One column per killer.
- An average score is given by averaging the individual rating columns of each killer per map.
- An overall bias column is created, to indicate whether a map is in the killer or the survivors favour.
    - >0 is killer-friendly
    - 0 is neutral
    - <0 is survivor-friendly

fact_matches

The fact_match dataset was a special dataset, in that it wasn't scraped from the same website like the rest of the data but was rather a dataset created by a player of the game and published in a Google Sheets spreadsheet.

- Raw data
- Clean data

The raw data was very unusable at the beginning, as it was not in a normalised, tabular structure. The following transformations were done:

<u>In the spreadsheet: pre-load</u>

- Created a unix column to store the datetime information
    - Done as a precaution, in case the date column didn't work once loaded, so that the date could be re-created, if needed.
- Added a 'match' column.
    - An integer that counts which match number a row belongs to

    - Repeated 5 times, once per player in a match.
- Created an: “is_survivor” column, that separates the survivors and killers.
    - In the spreadsheet, the pattern was recurring 1 0 0 0 0, with the killer always being reported first.
- Created a “died” column, that informs us whether a survivor died or not.
    - In the unclean data, this was recorded by text that was in red
- Created the “is_data_recorder” column, that informs us whether or not a particular character was played by the person recording the match data, or not.
    - In the unclean data, this was marked with the player name being prefixed with an asterisk (*).
- Perks were split out into separate columns.
    - Originally there was one column with an array of up to 4 perk names. This was separated out, with each perk having its own column (perk1 | perk2 | perk3 | perk4)
- Created the following boolean columns by analysis the comments left in the “Notes” column
    - **survivor_went_to_second_phase** (BOOLEAN): Indicates if the survivor reached the second phase on the hook.
        - If notes column contains “Second phase”, 1 else 0
    - **killer_allowed_survivor_escape** (BOOLEAN): Indicates if the killer allowed the survivor to escape.
        - If notes column contains “Allowed escape”, 1 else 0
    - **player_attempted_adept** (BOOLEAN): Indicates if the player attempted an adept challenge.
        - If notes column contains “adept”, 1 else 0
    - **killer_didnt_stay** (BOOLEAN): Indicates if the killer left the match early.
        - If notes column contains “Didn’t stay”, 1 else 0
    - **survivor_died_on_second_hook** (BOOLEAN): Indicates if the survivor died on their second hook.
        - If notes column contains “Died on second hook”, 1 else 0
    - **survivor_was_tunneled** (BOOLEAN): Indicates if the survivor was tunneled.
        - Tunnelling is the act of a killer trying to single out a specific survivor and kill them first, to remove them from the match, making it harder for the remaining survivors.
        - Ideally, the killer is expected to hook each survivor equally (first, all once and once the survivors have been hooked once, then aim to hook them all a second time).
        - If notes column contains “tunneled”, 1 else 0
    - **player_disconnected** (BOOLEAN): Indicates if the player disconnected during the match.
        - If notes column contains “dc”, 1 else 0
    - **survivor_hatch_escape** (BOOLEAN): Indicates if the survivor escaped through the hatch.
        - If notes column contains “hatch”, 1 else 0
    - **survivor_moried** (BOOLEAN): Indicates if the survivor was killed using a Mori.
        - If notes column contains “mori”, 1 else 0
    - **survivor_threw_on_first_hook** (BOOLEAN): Indicates if the survivor gave up on the first hook.
        - If notes column contains “Threw the match on first hook”, 1 else 0
    - **survivor_threw_on_second_hook** (BOOLEAN): Indicates if the survivor gave up on the second hook.
        - If notes column contains “Threw the match on second hook”, 1 else 0
    - **killer_was_friendly** (BOOLEAN): Indicates if the killer was friendly during the match.
        - If notes column contains “friendly”, 1 else 0
    - **match_cancelled** (BOOLEAN): Indicates if the match was cancelled.
        - If notes column contains “cancelled”, 1 else 0
    - **player_was_afk** (BOOLEAN): Indicates if the player was away from the keyboard during the match.
        - If notes column contains “afk”, 1 else 0

In python: post-load

- Dropped the unix column, as the date column worked as intended when loaded in.
- Data types were changed accordingly.
- Created the following boolean columns by combining data from other columns
    - **killer_4k (BOOLEAN):** Indicates if the killer achieved a 4K (killed all four survivors).
        - Used the “died” column sum per match
    - **killer_3k (BOOLEAN):** Indicates if the killer achieved a 3K (killed three survivors).
        - Used the “died” column sum per match
    - **killer_win (BOOLEAN):** Indicates if the killer won the match (3 or more kills in a match).
        - Used the “died” column sum per match, if 3 or more, then true, else false.
    - **draw (BOOLEAN):** Indicates if the match ended in a draw (2 survivors killed).
        - Used the “died” column sum per match, if 2 then true, else false.
    - **survivor_won (BOOLEAN):** Indicates if the survivor won the match (1 or fewer kills in the match).
        - Used the “died” column count per match, if 1 or less, then true, else false.

7. Data Integrity and Validation

7.1. Primary Keys and Uniqueness

- All primary keys (PK) must be unique.
- All character names (`stg_killers.name`, `stg_survivors.name`, `dim_characters.name`) must be unique.

7.2. Rating Validation

- Ratings (`stg_killers.rating`, `stg_survivors.rating`, `dim_characters.rating`, `stg_maps_killer.rating`, `dim_addons_killer.rating`, `stg_addons_killer.rating`, `stg_perks_killer.rating`, `stg_perks_survivor.rating`, `dim_perks.rating`, `fact_perks_killer_specific.rating`, `fact_perks_killer_specific_wide.rating`, `fact_perks.overall_rating`) must be between 0 and 5.

7.3. Date Validation

- Match dates (`fact_matches.date`) cannot be in the future.
- Last updated dates (`last_updated` fields in all tables) cannot be in the future.
- Release dates (`dim_characters.release_date`) cannot be in the future.

7.4. Bias Validation

- Bias values (`dim_maps.bias`, `fact_maps.bias`) must be between -1 and 1.

7.5. Boolean Fields Validation

- Boolean fields (`is_survivor`, `is_licensed`, `is_data_recorder`, etc.) must be either true or false.
- Ensure all records in `stg_killers` have `is_survivor` set to false.
- Ensure all records in `stg_survivors` have `is_survivor` set to true.

7.6. Cost Validation

- Iridescent Shard Cost (`dim_characters.iridescent_shard_cost`) must be a non-negative value.
- Auric Cell Cost (`dim_characters.auric_cell_cost`) must be a non-negative value.
- Total Cost in Euros (`dim_characters.total_cost_euros`) must be a non-negative value.

7.7. Map Name Consistency

- Map names (`map_name` fields in `stg_maps_killer`, `dim_maps`, `fact_maps`, and `fact_matches.map`) must be consistent and match across all tables.

7.8. Character Type Consistency

- Ensure that characters classified as killers or survivors in `stg_killers` and `stg_survivors` respectively are consistent in `dim_characters`.

7.9. Perk Category Consistency

- Perks classified as survivor perks (`is_survivor` set to true) in `stg_perks_survivor` must match corresponding records in `dim_perks`.
- Perks classified as killer perks (`is_survivor` set to false) in `stg_perks_killer` must match corresponding records in `dim_perks`.
- Specific killer perks (`killer_name` in `stg_perks_killer_specific`, `fact_perks_killer_specific`, and `fact_perks_killer_specific_wide`) must have a matching killer name in `stg_killers`.

7.10. Tier Validation

- Tier values (`tier` fields in all relevant tables) must be one of the predefined categories (e.g., `S`, `A`, `B`, `C`, `D`, `F`, `Unknown`).

7.11. Match Outcome Consistency

- Ensure logical consistency in match outcomes, such as `killer_win` should not be true if `survivor_won` is true, and vice versa.
- Ensure `killer_4k` and `killer_3k` are not both true for the same match.

7.12. Perks Equipped Count

- Perks equipped count (`fact_matches.perks_equipped_count`) should be between 0 and 4.

7.13. Notes Length

- Notes field (`fact_matches.notes`) should have a reasonable length limit (e.g., 500 characters) to ensure consistent data entry.

7.14. Non-Null Fields

- Ensure all primary key fields and any other critical fields are non-null.

8. Performance Considerations

At the current scope of the project, not many performance considerations really need to be taken into account, as the dataset is small.

However, should the project need to scale, here are some potential performance considerations that could be considered.

- **Partitioning**
    - If the project scope were to widen to incorporate rolling monthly data, a partitioning strategy for fact_matches table could be considered.
        - However, with the scope of the current project, this isn't needed as the data is manageable (5 months worth of data).
        - However, it could be partitioned on date, should more data be collected and on a more frequent timeframe.
- **Denormalization**
    - The current structure is fairly normalised. Depending on query patterns of the users, denormalizing some data for faster retrieval could be considered.
        - For instance, if users frequently need character information with match data, some character attributes could be added to the fact_matches table.
- **Batch Processing**
    - For the manual spreadsheet upload to fact_matches, batch processing could be used on a daily basis to handle large uploads efficiently.
        - Unless there is a stakeholder need for live data, which is unlikely.
- **ETL Process**
    - For the scraped data, incremental updates could be considered rather than full reloads.
    - This would actually work better with the data of the website being scraped, as it is refreshed on a monthly basis.

9. Security and Access Control

Not within the scope of the current project.

However, some considerations if the project were to scale would be:

- **Compliance**
    - Ensuring security measures comply with relevant data protection regulations (e.g., GDPR if dealing with EU user data).
- **Data Retention and Deletion**
    - Implement policies for secure data retention and deletion, especially for user-related data.
- **Data Masking and Anonymization:**
    - For non-production environments (like testing or development), use data masking to protect sensitive information.
- **Database Activity Monitoring:**
    - Implement real-time monitoring of database activities to detect and alert on suspicious behaviour.

10. Backup and Recovery

A full backup and recovery plan is not in scope of the current project.

In the scope of the project was saving the CSV files, as they will be overwritten by the website itself on a rolling monthly basis

11. References

CrypticGirl (2024). ‘*DBD Match Data Spreadsheet*.’ Behaviour Interactive Inc. Forum. https://forums.bhvr.com/dead-by-daylight/discussion/401797/dbd-match-data-spreadsheet. Accessed: 02 August, 2024.

Densmore, J. (2021). '*Data Pipelines Pocket Reference: Moving and Processing Data for Analytics.*' O'Reilly Media, Inc.

Dzzplayz. (2024). *‘I’ve categorized all of the perks, because I felt like it.’* Reddit. https://www.reddit.com/r/deadbydaylight/comments/1awkgnt/ive_categorized_all_of_the_perks_because_i_felt/. Accessed: 02 August, 2024.

GeeksforGeeks. (2021). ‘*Scrape Tables From any website using Python.’* https://www.geeksforgeeks.org/scrape-tables-from-any-website-using-python/. Accessed: 02 August, 2024.

Kimball, R. and Ross, M. (2013). '*The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling, Third Edition*.' John Wiley & Sons, Inc.

Peanits. (2024). ‘*Developer Update* | *Stats!*’ Behaviour Interactive Inc. https://forums.bhvr.com/dead-by-daylight/kb/articles/433-developer-update-stats. Accessed: 02 August, 2024.

Reep, D. (2024). ‘*Dead by Daylight Statistics Website.*’ https://dennisreep.nl/dbd/. Accessed: 02 August, 2024.

Reis, J. and Housley, M. (2022). '*Fundamentals of Data Engineering: Plan and Build Robust Data Systems.*' O'Reilly Media, Inc.

Wikipedia. (2024). ‘*Dead by Daylight.*’ https://en.wikipedia.org/wiki/Dead_by_Daylight. Accessed: 02 August, 2024.

12. Version History

- v1.0 (2024-08-02): Initial documentation