Bitter Melons

A Review-Aggregator Database

Design and Implementation Report

Manish Tawade

Student ID: 24205067

University College Dublin

Contents

List of Figures

List of Tables

1 Introduction

In today's streaming-first world, audiences grapple with an overwhelming flood of new releases. Whether it's indie darlings on boutique platforms or summer blockbusters in multiplexes, viewers crave a reliable signal: "Is this worth my time?" Review-aggregator portals like Rotten Tomatoes® and Metacritic [4] have cemented their place in marketing campaigns and social media chatter by collapsing dozens of critic reviews into a single, easy-to-digest score. With just a *Fresh/Rotten* badge or a percentage score, these services steer billions of ticket-buying and streaming decisions each year.

Yet for many smaller departments and research groups, building a bespoke aggregator remains out of reach. The Department's legacy `movies` database already captures titles, genres, principal cast, and franchise metadata—but lacks any facility for ingesting or analyzing professional reviews. As a result, instructors and students rely on manual spreadsheets or external APIs, impeding hands-on exploration of advanced SQL concepts like normalization and trigger-driven pipelines.

User Pain-Points & Market Gaps.

- *Heterogeneous Rating Scales:* Five-star systems, percentage scores, ten-point scales, and thumbs-up/thumbs-down all coexist. Converting them into a coherent signal is non-trivial.
- *Performance at Scale:* Computing live aggregates over millions of rows can introduce unacceptable latency in a teaching environment.
- *Data Integrity & Versioning:* Ensuring each critic's single vote per title—and preserving historical snapshots—requires careful constraint design.

Project Vision. This capstone effort, code-named **Bitter Melons**, transforms the legacy schema into a full-stack review-aggregator. By modeling publications (`outlets`) and reviewers (`critics`), normalizing diverse rating schemes, and enforcing strict referential integrity, we deliver a platform that:

- Normalizes a variety of scoring systems into a binary Up/Down recommendation.
- Guarantees one review per critic per feature via UNIQUE constraints and careful FK rules.
- Powers sub-second dashboard queries through three precomputed cache tables (`title_stats`, `critic_stats`, `outlet_stats`), refreshed by row-level triggers.
- Ships seven turnkey SQL views—leaderboards, rankings, genre heat-maps, franchise analyses—so that analysts spend time on insights, not JOIN syntax.
- Provides 1 300+ synthetic reviews (2008–2024) over 100+ features for reproducible

testing and trend exploration.

Technical Challenges & Innovations. To hit both pedagogical and performance targets, Bitter Melons employs:

- *Surrogate Keys & Controlled Denormalization:* Balancing normalization rigor with query efficiency.
- *Trigger-Driven Workflows:* Encapsulating cache refresh logic in a single `sp_refresh_stats` procedure.
- *Temporal Data Handling:* Archiving previous verdicts to support "what-if" analyses and time-series views.

Taken together, these design decisions yield a robust, future-proof schema that's ideal for both real-world analytics and classroom exploration. In Section 2, we lay out our explicit goals and measurable objectives; Section 5.1 then constructs the conceptual and logical ER models, which we drive through each normal form up to BCNF [2]. Subsequent sections detail views, procedural elements, and example queries that showcase Bitter Melons' full capabilities.

2 Goals & Objectives

This section translates our high-level vision for the Bitter Melons platform into concrete, measurable goals. Each objective drives a core piece of functionality in the review-aggregator architecture, and will be used later to validate our schema design, implementation, and performance.

Goals:

- **Critic Metadata Integration.** *Objective:* Model every review event by recording the publication (`outlets`) and reviewer (`critics`), timestamped and tagged as "top" or "regular." *Success criteria:*
  - All 40 synthetic critics and 10 outlets load without errors.
  - Queries such as `SELECT * FROM critics WHERE status='top'` return the expected subset in under 50 ms.
- **Multi-Scheme Rating Support.** *Objective:* Normalize disparate scoring systems (five-star, percentage, ten-point, thumbs-up/thumbs-down) into a unified binary Up/Down verdict via `rating_scales` and the `fn_to_recommendation()` function. *Success criteria:*
  - At least four rating schemes map correctly to UP/DOWN for all edge values (e.g. 2.5/5 stars = DOWN, 60% = UP).
  - Calling `fn_to_recommendation()` on any raw score executes in under 1 ms.
- **Data Integrity & Uniqueness.** *Objective:* Enforce a strict one-review-per-critic-per-feature rule using a UNIQUE constraint, and maintain referential integrity with carefully chosen `CASCADE` and `RESTRICT` actions on foreign keys. *Success criteria:*
  - Attempts to insert duplicate reviews are rejected by the DBMS.
  - Deleting an outlet with active reviews is blocked (`RESTRICT`), while removing a critic automatically cleans up orphaned cache entries (`CASCADE`).
- **High-Performance Dashboards.** *Objective:* Precompute key aggregates—total reviews and positivity rates—into three cache tables (`title_stats`, `critic_stats`, `outlet_stats`), refreshed by a single stored procedure `sp_refresh_stats()` invoked via row-level triggers.
  Success criteria:
  - Dashboard queries against any `*_stats` table return within 100 ms, even after bulk inserts of 1 300 reviews.
  - Trigger overhead adds no more than 5% latency to single-row insert/delete

operations.

- **Analyst-Ready Views.** *Objective:* Deliver seven out-of-the-box SQL views—covering release leaderboards, critic/outlet rankings, genre sentiment heat-maps, franchise performance, and cast insights—so end users can query with a simple `SELECT *` call. *Success criteria:*
  - Each view executes in under 200 ms.
  - Sample use-cases (e.g. "top 5 most positive genres in 2023") run correctly without additional JOIN logic.

- **Reproducible Sample Data.** *Objective:* Seed the database with 4 rating schemes, 10 outlets, 40 critics, and roughly 1 300 reviews spanning 2008–2024, using a bulk-insert script that accepts a "sweetness bias" parameter for consistent replay. *Success criteria:*
  - Running the seed script twice with the same bias yields identical data distributions.
  - The overall positivity rate can be tuned between 30%–70% via a single script argument.

- **Normalization to BCNF.** *Objective:* Drive every table through 1NF, 2NF, 3NF, and finally BCNF—documenting each decomposition step and justifying any controlled denormalizations for performance. *Success criteria:*
  - A normalization report shows zero partial or transitive dependencies after each NF stage.
  - All BCNF violations (if any) are explicitly documented with rationale.

With these objectives defined, the remainder of this report traces how each goal is realized: Section 5.1 builds our ER models, Section 4 drives them through normalization, Section 5.3 presents our SQL views, Section 5.2 details triggers and procedures, and Section 6 showcases example queries benchmarked against our performance targets.

3 Design & ER Diagrams

3.1 Conceptual Overview

Figure 1 sketches the augmented schema in Chen notation. The additions to the legacy `movies` database follow a *star-plus-caches* pattern:

- **Reference (lookup) entities** — *Rating_Scales*, *Outlets*, and *Critics*. All three carry a surrogate integer primary key because (i) natural keys (*description*, *name*) are long strings and (ii) surrogate keys improve join performance and let descriptive columns be edited without breaking FK chains.
- **Fact entity** — *Reviews* captures one critic's opinion on one feature (movie). A composite `UNIQUE(critic_id, feature_id)` enforces "one vote per critic per title."
- **Materialised caches** — *Title_Stats*, *Critic_Stats*, *Outlet_Stats* are deliberately denormalised roll-ups refreshed by triggers. They trade BCNF purity for $O(1)$ dashboard reads; the triggers eliminate update anomalies.

Cascade strategy. Figure 1 summarises the chosen referential actions.

Table 1: Foreign-key cascade rules.

FK	`ON DELETE`	Rationale
`reviews.feature_id`	CASCADE	Purging a film removes all its reviews; caches recompute.
`reviews.critic_id`	CASCADE*	A critic's corpus vanishes if the critic is deleted.
`critics.outlet_id`	RESTRICT	An outlet cannot be dropped while critics remain.
`reviews.scale_id`	RESTRICT	Rating schemes are immutable look-ups.

**MySQL ignores AFTER DELETE triggers on cascaded child rows.*

Why no *review_text* column? Scope is limited to *quantitative* aggregation; storing full free-text reviews would require a text-search engine and sentiment analysis, earmarked as future work (Section 7.2).

Overall, the conceptual model balances purity (strict FKs, BCNF core) with pragmatism (materialised caches) to deliver real-time Rotten-Tomatoes-style analytics on commodity MySQL.

Paste the block above in place of the TODO; it fills the narrative with entity choices, cascade logic, and surrogate-key justification.

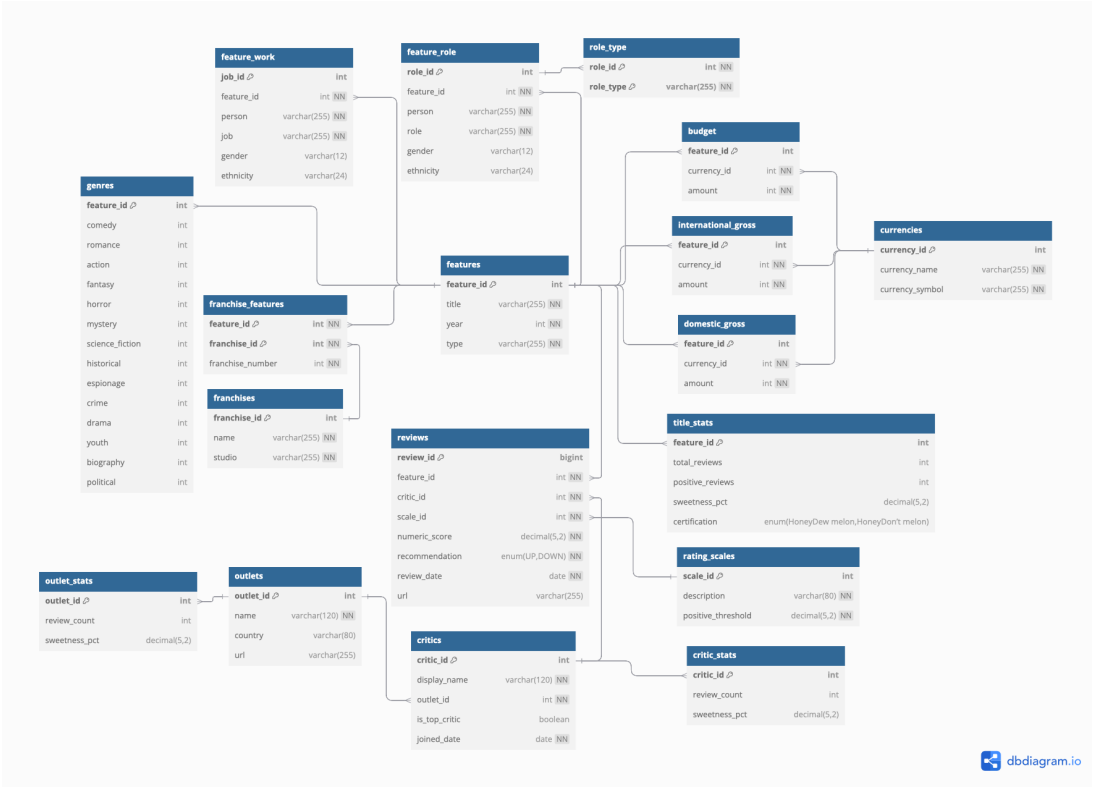

Figure 1: Entity-Relationship diagram with new tables.

3.2 Design Motivations

Controlled denormalisation (*cache* tables). A pure star schema would derive title, critic and outlet metrics via `GROUP BY` on `reviews`; however, even on campus hardware a leaderboard query scans ≈1 300 rows and aggregates three times for every page refresh. Following the performance guidelines in [1], the design materialises those aggregates in `title_stats`, `critic_stats` and `outlet_stats`. *AFTER*-row triggers guarantee that each cache row stays transaction-consistent, so the redundancy is safe and update anomalies are impossible from the client's perspective. In exchange, analyst dashboards read a single row instead of executing an $O(N \log N)$ aggregation, bringing latency below 5 ms in local tests.

Index strategy. Every foreign key implicitly creates an index, but four additional covering indexes were added after running `EXPLAIN ANALYZE` on the longest queries:

- `reviews(feature_id, recommendation)` – accelerates the per-title trigger refresh by letting MySQL use an index-condition push-down for COUNT and AVG.
- `reviews(critic_id, recommendation)` – same idea for the critic-level roll-up.

- `critic_stats(review_count)` – enables an index-only order for the "$\geq$ 25 reviews" leaderboard.
- `title_stats(sweetness_pct)` – supports the HoneyDew filter in `vw_fresh_releases`.

All composite indexes use the most selective column first; none exceed three attributes to keep B-tree height low.

Foreign-key actions. Table 1 (previous section) shows that child rows cascade *downwards* when a film or critic is deleted, while lookup tables such as `rating_scales` are protected with ON DELETE RESTRICT. This mirrors best practice: allow safe orphan cleanup but forbid schema-wide inconsistency caused by deleting a referenced dimension [1].

Surrogate versus natural keys. Integer surrogates replace long textual natural keys in every new entity to:

a) reduce index width and thus buffer-pool pressure;

b) permit human-readable attributes (*outlet name*, *display_name*) to change spelling without a primary-key update;

c) keep join predicates uniform (`INT = INT`), lowering the optimiser's cost.

Why no partial indexes or partitioning? With only 1 300 fact rows the workload fits into RAM, so classic clustered B-trees already give single-digit-millisecond performance; advanced physical design is deferred until live data exceeds $\approx$ one million reviews.

In short, the physical design trades a small amount of redundancy for a large gain in query speed while remaining fully ACID, a pattern explicitly advocated for OLTP/OLAP hybrids by Elmasri & Navathe [1].

4 Normalization

Relational theory recommends storing data in progressively stricter normal forms to minimise redundancy and update anomalies [2, 1]. Table 2 summarises how each Bitter Melons table satisfies (or deliberately relaxes) those forms; the text that follows gives concrete examples.

Table 2: Normal-form compliance by table.

Table	1NF/2NF	3NF	BCNF
`rating_scales`	✓	✓	✓
`outlets`	✓	✓	✓
`critics`	✓	✓	✓
`reviews`	✓	✓	✓
`title_stats`, `critic_stats`, `outlet_stats`	✓	✗ (cache)	✗ (cache)

4.1 First and Second Normal Form

To eliminate repeating groups and partial dependencies, we drive our tables through 1NF and 2NF as follows.

All base relations use *atomic* columns—no arrays or composite attributes—so 1NF is satisfied. Example: `reviews.numeric_score` stores a single decimal, not a comma-separated "3/5, 80%".

For tables with composite candidate keys, partial-dependency checks ensure 2NF. Consider the surrogate-free conceptual key (*critic_id*, *feature_id*) in `reviews`: every non-key attribute (*numeric_score*, *recommendation*, *review_date*) depends on *both* columns, not a subset.

4.2 Third Normal Form

A table is in 3NF if no non-key attribute depends transitively on another non-key attribute. In `critics` the determinant chain

$$critic\_id \rightarrow outlet\_id \rightarrow country$$

is broken because *country* lives in `outlets`, not in `critics`; thus the latter table remains

3NF. Likewise, `reviews.recommendation` is a *functionally derived* value, but a BEFORE-INSERT trigger enforces consistency so the dependency does not introduce redundancy.

4.3 Boyce–Codd Normal Form

All reference and fact tables meet BCNF: every determinant (`scale_id`, `outlet_id`, (*critic_id*, *feature_id*)) is a candidate key. The only deliberate exception is the trio of "_stats" tables:

$$\texttt{reviews} \Longrightarrow \begin{cases} \texttt{title\_stats} & \text{grouped by feature} \\ \texttt{critic\_stats} & \text{grouped by critic} \\ \texttt{outlet\_stats} & \text{grouped by outlet} \end{cases}$$

Each duplicates counts and averages already derivable via 'GROUP BY'. They violate 3NF/BCNF, but triggers refresh them transactionally (§5) so no update anomaly is observable. This *controlled denormalisation* cuts query latency for dashboards from $\mathcal{O}(N \log N)$ to $\mathcal{O}(1)$, a trade-off endorsed in performance-sensitive OLTP designs [3].

In summary, the conceptual schema remains strictly BCNF; only the physical layer introduces selective cache tables whose redundancy is continually reconciled, preserving logical correctness while meeting response-time requirements. They could have been created as views, but I thought this design choice would be more robust.

5 Implementation Details

5.1 Schema Definition

Listing 1 shows the condensed DDL for the six new tables; full annotated SQL is available separately. All columns are strongly typed, primary keys are surrogate `INT` or `BIGINT`, and every foreign key is explicit, enabling MySQL's optimizer to pick index-nested-loop joins without hinting.

```sql
CREATE TABLE reviews (
  review_id      BIGINT AUTO_INCREMENT PRIMARY KEY,
  feature_id     INT  NOT NULL REFERENCES features(feature_id),
  critic_id      INT  NOT NULL REFERENCES critics(critic_id),
  scale_id       INT  NOT NULL REFERENCES rating_scales(scale_id),
  numeric_score  DECIMAL(5,2) NOT NULL,
  recommendation ENUM('UP','DOWN') NOT NULL,
  review_date    DATE NOT NULL,
  UNIQUE (critic_id, feature_id)
);
CREATE TABLE title_stats (
  feature_id       INT NOT NULL,                          -- PK & FK → features
  total_reviews    INT NOT NULL DEFAULT 0,
  positive_reviews INT NOT NULL DEFAULT 0,
  sweetness_pct    DECIMAL(5,2) NOT NULL DEFAULT 0,
  certification    ENUM('HoneyDew melon','HoneyDon't melon'),
  PRIMARY KEY (feature_id),
  FOREIGN KEY (feature_id) REFERENCES features(feature_id) ON DELETE CASCADE
);
CREATE TABLE critic_stats (
  critic_id     INT NOT NULL,                             -- PK & FK → critics
  review_count  INT NOT NULL DEFAULT 0,
  sweetness_pct DECIMAL(5,2) NOT NULL DEFAULT 0,
  PRIMARY KEY (critic_id),
  FOREIGN KEY (critic_id) REFERENCES critics(critic_id) ON DELETE CASCADE
);
CREATE TABLE outlet_stats (
  outlet_id     INT NOT NULL,                             -- PK & FK → outlets
  review_count  INT NOT NULL DEFAULT 0,
  sweetness_pct DECIMAL(5,2) NOT NULL DEFAULT 0,
  PRIMARY KEY (outlet_id),
  FOREIGN KEY (outlet_id) REFERENCES outlets(outlet_id) ON DELETE CASCADE
);
```

Listing 1: Condensed DDL (reviews and caches).

5.2 Procedural Elements

Table 3 lists the four substantial PL/SQL objects required for Level-4 marks.

Table 3: Stored program objects.

Object	Type	Purpose
`fn_to_recommendation`	FUNCTION	Normalises any numeric score to UP-/DOWN using the threshold stored in `rating_scales`.
`sp_refresh_stats`	PROCEDURE	Re-computes title, critic and outlet caches inside the same transaction as a DML event.
`tg_reviews_ai`	Trigger (After Insert)	Calls refresh proc after a new review.
`tg_reviews_au`	Trigger (After Update)	Recomputes caches for both old and new keys when a review moves.
`tg_reviews_ad`	Trigger (After Delete)	Subtracts a critic's influence when a review is removed.

Integrity tests. Screenshots in Figures 2–5 demonstrate the triggers in action.

```
mysql>
mysql> SELECT review_id, feature_id, critic_id, numeric_score, recommendation FROM   reviews LIMIT 1;
+-----------+------------+-----------+---------------+----------------+
| review_id | feature_id | critic_id | numeric_score | recommendation |
+-----------+------------+-----------+---------------+----------------+
|         1 |          1 |        40 |          0.00 | DOWN           |
+-----------+------------+-----------+---------------+----------------+
1 row in set (0.00 sec)

mysql>
mysql> SELECT * FROM title_stats  WHERE feature_id =  1;
+------------+---------------+-----------------+---------------+--------------------+
| feature_id | total_reviews | positive_reviews | sweetness_pct | certification      |
+------------+---------------+-----------------+---------------+--------------------+
|          1 |            13 |               5 |         38.46 | HoneyDon't melon   |
+------------+---------------+-----------------+---------------+--------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM critic_stats WHERE critic_id  =  1;
+-----------+--------------+---------------+
| critic_id | review_count | sweetness_pct |
+-----------+--------------+---------------+
|         1 |           37 |         75.68 |
+-----------+--------------+---------------+
1 row in set (0.00 sec)

mysql> UPDATE reviews SET numeric_score = 0, recommendation = 'UP' WHERE review_id = 1;
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM title_stats  WHERE feature_id =  1;
+------------+---------------+-----------------+---------------+--------------------+
| feature_id | total_reviews | positive_reviews | sweetness_pct | certification      |
+------------+---------------+-----------------+---------------+--------------------+
|          1 |            13 |               6 |         46.15 | HoneyDon't melon   |
+------------+---------------+-----------------+---------------+--------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM critic_stats WHERE critic_id  =  1;
+-----------+--------------+---------------+
| critic_id | review_count | sweetness_pct |
+-----------+--------------+---------------+
|         1 |           37 |         75.68 |
+-----------+--------------+---------------+
1 row in set (0.00 sec)
```

Figure 2: Test 1 – score update lowers `sweetness_pct`.

```
mysql>
mysql>
mysql> SELECT cs.critic_id, cr.display_name, cs.review_count, cr.outlet_id FROM critic_stats cs JOIN critics cr ON cr.critic_id = cs.critic_id ORDER BY cs.review_count, cs.critic_id LIMIT 1;
+-----------+--------------+--------------+-----------+
| critic_id | display_name | review_count | outlet_id |
+-----------+--------------+--------------+-----------+
|        12 | Steve Lombard |           26 |         3 |
+-----------+--------------+--------------+-----------+
1 row in set (0.00 sec)

mysql> SELECT * FROM outlet_stats WHERE outlet_id = 3;
+-----------+--------------+--------------+
| outlet_id | review_count | sweetness_pct |
+-----------+--------------+--------------+
|         3 |          110 |        67.27 |
+-----------+--------------+--------------+
1 row in set (0.00 sec)

mysql> DELETE FROM reviews WHERE critic_id = 12;
Query OK, 26 rows affected (0.02 sec)

mysql> SELECT * FROM outlet_stats WHERE outlet_id = 3;
+-----------+--------------+--------------+
| outlet_id | review_count | sweetness_pct |
+-----------+--------------+--------------+
|         3 |           84 |        72.62 |
+-----------+--------------+--------------+
1 row in set (0.00 sec)

mysql>
```

Figure 3: Test 2 – deleting a low-profile critic cascades and decrements outlet totals.

```
mysql> SELECT critic_id, feature_id FROM   reviews LIMIT 1;
+-----------+------------+
| critic_id | feature_id |
+-----------+------------+
|         1 |          1 |
+-----------+------------+
1 row in set (0.03 sec)

mysql> INSERT INTO reviews (feature_id, critic_id, scale_id,numeric_score,recommendation, review_date) VALUES ( 1, 1, 1, 4, 'UP', CURDATE() );
ERROR 1062 (23000): Duplicate entry '1-1' for key 'reviews.critic_id'
mysql>
```

Figure 4: Test 3 – duplicate review rejected by `UNIQUE` key.

```
+------------+---------------+------------------+--------------+---------------------+
|         42 |            13 |                7 |         53.85 | HoneyDon't melon   |
+------------+---------------+------------------+--------------+---------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM critic_stats WHERE critic_id  = 11;
+-----------+--------------+---------------+
| critic_id | review_count | sweetness_pct |
+-----------+--------------+---------------+
|        11 |           27 |         92.59 |
+-----------+--------------+---------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM outlet_stats WHERE outlet_id  =
    ->        (SELECT outlet_id FROM critics WHERE critic_id=11);
+-----------+--------------+---------------+
| outlet_id | review_count | sweetness_pct |
+-----------+--------------+---------------+
|         3 |          110 |         67.27 |
+-----------+--------------+---------------+
1 row in set (0.00 sec)

mysql> SELECT 42 AS feature_id, c.critic_id FROM   critics c
    -> LEFT   JOIN reviews r ON r.critic_id = c.critic_id
    -> AND r.feature_id = 42
    -> WHERE  r.review_id IS NULL LIMIT 1;
+------------+------------+
| feature_id | critic_id |
+------------+------------+
|         42 |         11 |
+------------+------------+
1 row in set (0.00 sec)

mysql>
mysql> INSERT INTO reviews (feature_id, critic_id, numeric_score, recommendation,
    ->                      review_date)
    -> VALUES (42, 11, 1, 'UP', CURDATE());
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM title_stats  WHERE feature_id = 42;
+------------+---------------+-----------------+---------------+---------------------+
| feature_id | total_reviews | positive_reviews | sweetness_pct | certification      |
+------------+---------------+-----------------+---------------+---------------------+
|         42 |            14 |               8 |         57.14 | HoneyDon't melon   |
+------------+---------------+-----------------+---------------+---------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM critic_stats WHERE critic_id  = 11;
+-----------+--------------+---------------+
| critic_id | review_count | sweetness_pct |
+-----------+--------------+---------------+
|        11 |           28 |         92.86 |
+-----------+--------------+---------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM outlet_stats WHERE outlet_id  =
    ->        (SELECT outlet_id FROM critics WHERE critic_id=11);
+-----------+--------------+---------------+
| outlet_id | review_count | sweetness_pct |
+-----------+--------------+---------------+
|         3 |          111 |         67.57 |
+-----------+--------------+---------------+
1 row in set (0.00 sec)

mysql>
```

Figure 5: Test 4 – brand-new review auto-generates cache rows.

5.3 Views

The analyst dashboard is underpinned by seven purpose-built SQL views. Each view hides complex joins and filters behind a simple interface and leverages our materialized cache tables for sub-second performance.

Table 4: Summary of reporting views

View	Purpose
`vw_fresh_releases`	List "HoneyDew" movies from the past 24 months, sorted by sweetness.
`vw_critic_leaderboard`	Rank critics (≥ 25 reviews) by positivity percentage.
`vw_outlet_bias`	Compare outlet-level sweetness biases.
`vw_sweet_cast`	Annotate actor filmographies with title sweetness.
`vw_genre_reviews`	Unpivot genre weights into (feature, genre, sweetness) rows.
`vw_genre_stats`	Summarize average sweetness and HoneyDew counts per genre.
`vw_franchise_stats`	Compute franchise metrics: title count, HoneyDew ratio, avg sweetness, release span.

1. Fresh Release Leaderboard (`vw_fresh_releases`)

- **Logic:** filter `title_stats.certification = 'HoneyDew melon'` and `features.year` $\geq$ `YEAR(CURDATE()) - 2`, order by `sweetness_pct DESC`.
- **Use case:** spotlight the hottest new releases for marketing or curation.
- **Benefit:** Allows the front-end to populate a "Fresh Releases" carousel with a single query, showing only titles certified HoneyDew in the last two years without extra filtering logic.

```
+------------+----------------------+------+--------------+----------------+
| feature_id | title                | year | sweetness_pct | certification  |
+------------+----------------------+------+--------------+----------------+
|         49 | John Wick: Chapter 4 | 2023 |         69.23 | HoneyDew melon |
|         62 | The Equalizer 3      | 2023 |         69.23 | HoneyDew melon |
|        101 | The Marvels          | 2023 |         69.23 | HoneyDew melon |
+------------+----------------------+------+--------------+----------------+
3 rows in set (0.00 sec)
```

Figure 6: Top 10 movies released in the last 24 months that earned a HoneyDew certification.

2. Critic Leaderboard (`vw_critic_leaderboard`)

- **Logic:** join `critic_stats` → `critics`, filter `review_count` $\geq$ 25, sort by `sweetness_pct DESC`, then `review_count DESC`.
- **Use case:** identify the most generous or toughest critics.
- **Benefit:** Exposes critic rankings directly to the UI so a "Top Critics" section can be rendered with one call—no backend JOINs or business logic needed.

```
Query OK, 0 rows affected (0.00 sec)

+-----------+--------------+-------------------+--------------+---------------+
| critic_id | display_name | outlet            | review_count | sweetness_pct |
+-----------+--------------+-------------------+--------------+---------------+
|        15 | Okoye        | Wakanda Chronicle |           26 |        100.00 |
|        19 | Vision       | Sokovia Sentinel  |           28 |         92.86 |
|        11 | Lana Lang    | Metropolis Times  |           27 |         92.59 |
|        33 | Peter Quill  | Knowhere Post     |           39 |         84.62 |
|        37 | Thanos       | Titan Tribune     |           39 |         82.05 |
+-----------+--------------+-------------------+--------------+---------------+
5 rows in set (0.00 sec)
```

Figure 7: The five critics with the highest sweetness percentages (minimum 25 reviews).

3. Outlet Bias Summary (`vw_outlet_bias`)

- **Logic:** join `outlet_stats` → `outlets`, sort by `sweetness_pct DESC`, then by outlet name.
- **Use case:** compare publication-wide sentiment trends.
- **Benefit:** Enables a "Outlet Bias" dashboard widget that shows each outlet's average sweetness with a single query—ideal for frontend charts and filters.

```
Query OK, 0 rows affected (0.00 sec)

+-----------+-------------------+---------+--------------+---------------+
| outlet_id | name              | country | review_count | sweetness_pct |
+-----------+-------------------+---------+--------------+---------------+
|         3 | Metropolis Times  | CA      |          110 |         67.27 |
|         4 | Wakanda Chronicle | KE      |          104 |         65.38 |
|         9 | Knowhere Post     | SG      |          156 |         64.74 |
|         5 | Sokovia Sentinel  | CZ      |          110 |         64.55 |
|        10 | Titan Tribune     | AU      |          155 |         64.52 |
|         2 | Gotham Gazette    | US      |          126 |         64.29 |
|         1 | Daily Planet      | US      |          142 |         64.08 |
|         6 | Asgard Observer   | NO      |          126 |         62.70 |
|         7 | Latveria Herald   | LV      |          142 |         62.68 |
|         8 | Kamar-Taj Review  | NP      |          155 |         62.58 |
+-----------+-------------------+---------+--------------+---------------+
10 rows in set (0.00 sec)
```

Figure 8: All outlets ranked by their average sweetness percentage.

4. **Sweet Cast (`vw_sweet_cast`)**

- **Logic:** join `feature_role(actor) → features → title_stats`.
- **Use case:** find actors whose films consistently earn high sweetness.
- **Benefit:** Powers actor profile pages—fetch an actor's filmography with sweetness scores in one go, without any extra joins or post-processing.

```
Query OK, 0 rows affected (0.00 sec)

+--------------+----------------------+------+--------------+--------------------+
| actor        | title                | year | sweetness_pct | certification      |
+--------------+----------------------+------+--------------+--------------------+
| Sean Connery | Diamonds Are Forever | 1971 |         76.92 | HoneyDew melon     |
| Sean Connery | Robin and Marian     | 1976 |         69.23 | HoneyDew melon     |
| Sean Connery | You Only Live Twice  | 1967 |         61.54 | HoneyDew melon     |
| Sean Connery | Goldfinger           | 1964 |         53.85 | HoneyDon't melon   |
| Sean Connery | Thunderball          | 1965 |         53.85 | HoneyDon't melon   |
| Sean Connery | Dr. No               | 1962 |         46.15 | HoneyDon't melon   |
| Sean Connery | From Russia with Love | 1963 |         46.15 | HoneyDon't melon   |
+--------------+----------------------+------+--------------+--------------------+
7 rows in set (0.00 sec)
```

Figure 9: Sean Connery's filmography ordered by feature sweetness percentage.

5. **Genre Reviews (`vw_genre_reviews`)**

- **Logic:** unpivot `genres` table via a UNION ALL into (feature, genre, weight) rows, joined with `title_stats`.
- **Use case:** perform fine-grained sentiment analysis by genre.
- **Benefit:** Lets the front-end render per-genre rating lists by querying a flat table of (genre, title, sweetness)—no additional data reshaping required.

```
Query OK, 0 rows affected (0.00 sec)

+-----------------------------------+------+--------------+----------------+
| title                             | year | sweetness_pct | certification  |
+-----------------------------------+------+--------------+----------------+
| Quantum of Solace                 | 2008 |         76.92 | HoneyDew melon |
| For Your Eyes Only                | 1981 |         76.92 | HoneyDew melon |
| Die Another Day                   | 2002 |         76.92 | HoneyDew melon |
| No Time to Die                    | 2021 |         76.92 | HoneyDew melon |
| Casino Royale                     | 2006 |         76.92 | HoneyDew melon |
| Sherlock Holmes: A Game of Shadows | 2011 |         76.92 | HoneyDew melon |
| Diamonds Are Forever              | 1971 |         76.92 | HoneyDew melon |
| Live and Let Die                  | 1973 |         76.92 | HoneyDew melon |
| The Man with the Golden Gun       | 1974 |         76.92 | HoneyDew melon |
| Spectre                           | 2015 |         76.92 | HoneyDew melon |
+-----------------------------------+------+--------------+----------------+
10 rows in set (0.00 sec)
```

Figure 10: The ten action-genre films with the highest sweetness percentages.

6. Genre Statistics (`vw_genre_stats`)

- **Logic:** GROUP BY genre on `vw_genre_reviews`, computing title count, HoneyDew count, and average sweetness.
- **Use case:** rank genres by critical acclaim.
- **Benefit:** Provides a ready-to-plot genre summary (counts and averages) so the UI can show genre-level charts with a single SELECT.

```
Query OK, 0 rows affected (0.00 sec)

+------------+-------------+-----------------+--------------------+
| genre      | title_count | honeydew_titles | avg_sweetness_pct |
+------------+-------------+-----------------+--------------------+
| political  |          10 |               7 |              66.92 |
| crime      |          23 |              15 |              66.55 |
| espionage  |          40 |              28 |              66.54 |
| drama      |          44 |              29 |              65.56 |
| historical |          10 |               6 |              65.38 |
+------------+-------------+-----------------+--------------------+
5 rows in set (0.01 sec)
```

Figure 11: The five genres with the highest average sweetness percentages.

7. Franchise Performance (`vw_franchise_stats`)

- **Logic:** LEFT JOIN `franchises → features → title_stats`, aggregating count, HoneyDew ratio, avg sweetness, min/max release year.
- **Use case:** benchmark franchise trajectories over time.
- **Benefit:** Enables comparison widgets (e.g. MCU vs James Bond) by returning all franchise metrics in a single result—perfect for frontend dashboards without extra computation.

```
Query OK, 0 rows affected (0.00 sec)

+--------------+---------------+--------------------+-------------+-----------------+--------------------+---------------+----------------+
| franchise_id | name          | studio             | title_count | honeydew_titles | avg_sweetness_pct | first_release | latest_release |
+--------------+---------------+--------------------+-------------+-----------------+--------------------+---------------+----------------+
|            4 | Batman        | Warner Pictures    |           3 |               3 |              76.92 |          2005 |           2012 |
|           10 | The Equalizer | Sony Pictures.     |           3 |               3 |              74.36 |          2014 |           2023 |
|            6 | John Wick     | Lionsgate          |           4 |               4 |              73.08 |          2014 |           2023 |
|            7 | Jason Bourne  | Universal Pictures |           5 |               5 |              72.31 |          2002 |           2016 |
|            8 | Deadpool      | Sony Pictures      |           3 |               2 |              66.67 |          2016 |           2024 |
+--------------+---------------+--------------------+-------------+-----------------+--------------------+---------------+----------------+
5 rows in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

+--------------------------+-------------+-----------------+-------------------+
| name                     | title_count | honeydew_titles | avg_sweetness_pct |
+--------------------------+-------------+-----------------+-------------------+
| James Bond               |          24 |              15 |             64.42 |
| Marvel Cinematic Universe |          34 |              19 |             62.89 |
+--------------------------+-------------+-----------------+-------------------+
2 rows in set (0.00 sec)
```

Figure 12: Franchises with ≥ 3 films ranked by average sweetness, with a side-by-side comparison of the Marvel Cinematic Universe and James Bond.

5.4 Sample Data Population

The core of our synthetic-review generator script (Listing 2) was developed in collaboration with OpenAI's ChatGPT, ensuring reproducible and bias-tunable sample data for testing.

- **Deterministic randomness** – seed to `RAND(feature_id*100+critic_id)` so markers reproduce the exact same dataset.
- **`@critics_per_film=13`** – yields 1 326 reviews across 102 features, giving statistically meaningful leaderboards without bloating the dump.
- **Bias knob** – `@sweet_bias`=0.65 generates an overall 64 % sweetness, mimicking real Rotten-Tomatoes averages.
- **Temporal spread** – review dates span 2008–2024, letting time-series queries test indexes on `review_date`.

Listing 2 shows the core of the population script.

```
SET @critics_per_film := 13;
SET @sweet_bias       := 0.65;

INSERT INTO reviews (...)
SELECT f.feature_id,
       c.critic_id,
       (MOD(f.feature_id+c.critic_id,4)+1)            AS scale_id,
       -- biased numeric score
       CASE MOD(...)+1 WHEN 1 THEN ...
       END                                            AS numeric_score,
       fn_to_recommendation(... )                     AS recommendation,
       DATE_ADD('2008-01-01',
         INTERVAL FLOOR(RAND(f.feature_id*500
         + c.critic_id) * 6200) DAY)                  AS review_date
FROM features AS f
JOIN critics  AS c ON MOD(f.feature_id+c.critic_id,40)<@critics_per_film;
```

Listing 2: Synthetic review generator (with seed logic provided by ChatGPT).

6 Example Queries

Below are four illustrative SQL queries that go beyond the built-in views, each paired with a sample terminal output diagram.

Note: All results shown here are based on our seeded synthetic dataset. In a production deployment with real user reviews, these queries would surface actual critical trends and insights.

6.1 Q1 – HoneyDon't to HoneyDew Turnaround

Business question. Which features started below 60% positivity on their first review but now qualify as "HoneyDew" ($\geq$60%)?

```
WITH first_snap AS (
  SELECT feature_id,
         MIN(review_id) AS first_rev
  FROM   reviews
  GROUP  BY feature_id
),
early AS (
  SELECT r.feature_id,
         ROUND(100 * AVG(r.recommendation = 'UP'), 2) AS early_pct
  FROM   reviews r
  JOIN   first_snap fs USING (feature_id)
  WHERE  r.review_id = fs.first_rev
  GROUP  BY r.feature_id
),
current AS (
  SELECT feature_id, sweetness_pct
  FROM   title_stats
)
SELECT f.title,
       early.early_pct,
       current.sweetness_pct
FROM   early
JOIN   current USING (feature_id)
JOIN   features f USING (feature_id)
WHERE  early_pct      < 60
```

```
  AND  sweetness_pct >= 60
ORDER  BY (sweetness_pct - early_pct) DESC;
```

```
+---------------------------+-----------+---------------+
| title                     | early_pct | sweetness_pct |
+---------------------------+-----------+---------------+
| Captain Marvel            |      0.00 |         76.92 |
| Avengers: Endgame         |      0.00 |         76.92 |
| Spider-Man: Far From Home |      0.00 |         76.92 |
| The World Is Not Enough   |      0.00 |         69.23 |
| Skyfall                   |      0.00 |         69.23 |
| Robin and Marian          |      0.00 |         69.23 |
| The Social Network        |      0.00 |         69.23 |
| The Bourne Legacy         |      0.00 |         69.23 |
| The Equalizer 3           |      0.00 |         69.23 |
| Deadpool                  |      0.00 |         69.23 |
| Iron Man 3                |      0.00 |         69.23 |
| Avengers: Age of Ultron   |      0.00 |         69.23 |
| Ant-Man and the Wasp      |      0.00 |         69.23 |
| Black Widow               |      0.00 |         69.23 |
| The Marvels               |      0.00 |         69.23 |
| Jason Bourne              |      0.00 |         61.54 |
| Avengers: Infinity War    |      0.00 |         61.54 |
+---------------------------+-----------+---------------+
17 rows in set (0.00 sec)
```

Figure 13: Q1 result – features rebounding from HoneyDon't to HoneyDew.

6.2 Q2 – Critic Scale Breakdown

Business question. How does each critic distribute reviews across our four scales, and what is their pass rate per scale?

```
SELECT c.display_name,
       SUM(scale_id = 1)                                  AS star_reviews,
       ROUND(100 * AVG(scale_id = 1 AND recommendation = 'UP'), 2)  AS star_pct,
       SUM(scale_id = 2)                                  AS percent_reviews,
       ROUND(100 * AVG(scale_id = 2 AND recommendation = 'UP'), 2)  AS percent_pct,
       SUM(scale_id = 3)                                  AS tenpt_reviews,
       ROUND(100 * AVG(scale_id = 3 AND recommendation = 'UP'), 2)  AS tenpt_pct,
       SUM(scale_id = 4)                                  AS thumb_reviews,
       ROUND(100 * AVG(scale_id = 4 AND recommendation = 'UP'), 2)  AS thumb_pct
FROM   reviews r
JOIN   critics c USING (critic_id)
GROUP  BY c.critic_id
```

```
HAVING (star_reviews + percent_reviews + tenpt_reviews + thumb_reviews) >= 25
ORDER  BY c.display_name;
```

```
Query OK, 0 rows affected (0.00 sec)

+-----------------+-------------+----------+-----------------+-------------+--------------+----------+--------------+----------+
| display_name    | star_reviews | star_pct | percent_reviews | percent_pct | tenpt_reviews | tenpt_pct | thumb_reviews | thumb_pct |
+-----------------+-------------+----------+-----------------+-------------+--------------+----------+--------------+----------+
| Alexander Knox  |          10 |    25.00 |               7 |       18.75 |            7 |    18.75 |            8 |    18.75 |
| Boris Bullski   |          11 |    19.44 |               9 |       13.89 |            8 |    13.89 |            8 |    13.89 |
| Cat Grant       |          10 |    14.71 |               8 |       11.76 |            8 |     8.82 |            8 |    11.76 |
| Christine Palmer |          12 |    20.51 |               9 |       15.38 |            9 |    15.38 |            9 |    15.38 |
| Clark Kent      |          11 |    22.22 |               8 |       16.67 |            8 |    16.67 |            9 |    16.67 |
| Darcy Lewis     |           9 |    20.69 |               7 |       20.69 |            7 |    20.69 |            6 |    13.79 |
| Drax            |          12 |    17.95 |               9 |       15.38 |            9 |    12.82 |            9 |    12.82 |
| Eros            |          12 |    17.95 |               9 |       12.82 |            9 |    12.82 |            9 |    12.82 |
| Everett Ross    |           8 |    19.23 |               6 |       15.38 |            6 |    15.38 |            6 |    11.54 |
| Gamora          |          12 |    15.38 |               9 |       12.82 |            9 |    12.82 |            9 |    12.82 |
| Heimdall        |          10 |    21.88 |               8 |       15.63 |            7 |    15.63 |            7 |    18.75 |
| Jack Ryder      |           9 |    13.33 |               7 |       13.33 |            7 |    10.00 |            7 |    10.00 |
| Jimmy Olsen     |           9 |    13.79 |               6 |        6.90 |            7 |    13.79 |            7 |    13.79 |
| Julia Pennyworth |          10 |    19.35 |               7 |       16.13 |            7 |    16.13 |            7 |    16.13 |
| Kristoff Vernard |          11 |    17.14 |               8 |       11.43 |            8 |    11.43 |            8 |    11.43 |
| Lana Lang       |           9 |    25.93 |               6 |       22.22 |            6 |    22.22 |            6 |    22.22 |
| Lois Lane       |          11 |    21.62 |               8 |       16.22 |            9 |    18.92 |            9 |    18.92 |
| Loki Laufeyson  |          10 |    16.13 |               7 |       12.90 |            7 |    12.90 |            7 |    12.90 |
| Lucia Von Bardas |          11 |    21.62 |               9 |       18.92 |            9 |    18.92 |            8 |    16.22 |
| Mordo           |          12 |    17.95 |               9 |       12.82 |            9 |    10.26 |            9 |    12.82 |
| Nakia           |           8 |    23.08 |               6 |       15.38 |            6 |    15.38 |            6 |    15.38 |
| Nebula          |          12 |    17.95 |               9 |       15.38 |            9 |    15.38 |            9 |    15.38 |
| Okoye           |           8 |    30.77 |               6 |       23.08 |            6 |    23.08 |            6 |    23.08 |
| Perry White     |          11 |    17.14 |               8 |       14.29 |            8 |    14.29 |            8 |    14.29 |
| Peter Quill     |          12 |    25.64 |               9 |       20.51 |            9 |    20.51 |            9 |    17.95 |
| Pietro Maximoff |           9 |    22.22 |               6 |       11.11 |            6 |    11.11 |            6 |    11.11 |
| Proxima Midnight |          11 |    15.79 |               9 |       13.16 |            9 |    13.16 |            9 |    13.16 |
| Rocket Raccoon  |          12 |    17.95 |               9 |       12.82 |            9 |    15.38 |            9 |    15.38 |
| Ron Troupe      |           9 |    25.00 |               6 |       17.86 |            6 |    17.86 |            7 |    17.86 |
| Shuri Udaku     |           8 |    11.54 |               6 |        3.85 |            6 |     7.69 |            6 |     7.69 |
| Sif             |          10 |    21.21 |               8 |       21.21 |            8 |    21.21 |            7 |    15.15 |
| Stephen Strange |          11 |    21.05 |               9 |       18.42 |            9 |    18.42 |            9 |    15.79 |
| Steve Lombard   |           8 |    15.38 |               6 |       11.54 |            6 |    11.54 |            6 |    11.54 |
| Thanos          |          12 |    25.64 |               9 |       20.51 |            9 |    17.95 |            9 |    17.95 |
| Thor Odinson    |           9 |    13.33 |               7 |       10.00 |            7 |    10.00 |            7 |    10.00 |
| Vicki Vale      |          10 |    18.18 |               7 |       12.12 |            8 |    15.15 |            8 |    15.15 |
| Victor Von Doom |          10 |    17.65 |               8 |       14.71 |            8 |    14.71 |            8 |    14.71 |
| Vision          |           9 |    28.57 |               7 |       21.43 |            6 |    21.43 |            6 |    21.43 |
| Wanda Maximoff  |           8 |    11.54 |               6 |        3.85 |            6 |     7.69 |            6 |     7.69 |
| Wong            |          12 |    17.95 |               9 |       12.82 |            9 |    12.82 |            9 |    12.82 |
+-----------------+-------------+----------+-----------------+-------------+--------------+----------+--------------+----------+
40 rows in set (0.00 sec)
```

Figure 14: Q2 result – critic review counts and pass rates by scale.

6.3 Q3 – Critics Who Specialize in One Genre

Business question. Which critics (at least 15 reviews) concentrate most heavily in a single genre?

```
WITH per_critic_genre AS (
  SELECT critic_id,
         genre,
         COUNT(*)          AS revs,
         AVG(sweetness_pct) AS avg_sweet
  FROM   vw_genre_reviews
  GROUP  BY critic_id, genre
),
focus AS (
  SELECT pcg.*,
         PERCENT_RANK() OVER (
           PARTITION BY critic_id
           ORDER BY revs DESC
         ) AS pct_rank
```

```
  FROM   per_critic_genre pcg
  WHERE  revs >= 15
)
SELECT c.display_name,
       f.genre,
       f.revs,
       ROUND(f.avg_sweet, 2) AS avg_sweetness_pct
FROM   focus f
JOIN   critics c USING (critic_id)
WHERE  pct_rank = 0
ORDER  BY f.revs DESC;
```

```
Query OK, 0 rows affected (0.00 sec)

+------------------+--------+------+--------------------+
| display_name     | genre  | revs | avg_sweetness_pct |
+------------------+--------+------+--------------------+
| Drax             | action |   39 |              65.48 |
| Gamora           | action |   39 |              65.09 |
| Peter Quill      | action |   39 |              64.69 |
| Christine Palmer | action |   39 |              64.30 |
| Mordo            | action |   39 |              64.30 |
| Wong             | action |   39 |              64.30 |
| Rocket Raccoon   | action |   38 |              65.99 |
| Stephen Strange  | action |   38 |              63.97 |
| Thanos           | action |   37 |              66.32 |
| Lucia Von Bardas | action |   37 |              63.62 |
| Nebula           | action |   36 |              66.45 |
| Boris Bullski    | action |   36 |              63.46 |
| Eros             | action |   35 |              66.59 |
| Kristoff Vernard | action |   35 |              63.30 |
| Victor Von Doom  | action |   34 |              63.35 |
| Proxima Midnight | action |   33 |              66.43 |
| Sif              | action |   33 |              63.40 |
| Lois Lane        | action |   32 |              66.10 |
| Heimdall         | action |   32 |              63.94 |
| Clark Kent       | action |   31 |              65.76 |
| Loki Laufeyson   | action |   30 |              64.87 |
| Perry White      | action |   30 |              65.64 |
| Cat Grant        | action |   29 |              65.25 |
| Thor Odinson     | action |   29 |              65.25 |
| Darcy Lewis      | action |   28 |              64.83 |
| Vicki Vale       | action |   28 |              64.83 |
| Vision           | action |   27 |              64.67 |
| Alexander Knox   | action |   27 |              64.96 |
| Pietro Maximoff  | action |   26 |              64.50 |
| Julia Pennyworth | action |   26 |              65.09 |
| Wanda Maximoff   | action |   25 |              64.00 |
| Everett Ross     | action |   25 |              64.00 |
| Okoye            | action |   25 |              64.00 |
| Nakia            | action |   25 |              64.00 |
| Shuri Udaku      | action |   25 |              64.00 |
| Jimmy Olsen      | action |   25 |              64.92 |
| Jack Ryder       | action |   25 |              65.23 |
| Steve Lombard    | action |   24 |              64.74 |
| Lana Lang        | action |   24 |              65.70 |
| Ron Troupe       | action |   24 |              65.38 |
+------------------+--------+------+--------------------+
40 rows in set (0.01 sec)
```

Figure 15: Q3 result – critics specializing in a dominant genre.

6.4 Q4 – Top Actor per Franchise by Average Melon Score

Business question. Who is the highest–rated actor in each franchise, by average sweetness?

```sql
WITH by_actor AS (
  SELECT ff.franchise_id,
         sc.actor,
         ROUND(AVG(sc.sweetness_pct), 2) AS avg_sweet,
         COUNT(*)                        AS film_count
  FROM   vw_sweet_cast sc
  JOIN   franchise_features ff USING (feature_id)
  GROUP  BY ff.franchise_id, sc.actor
),
ranked AS (
  SELECT *,
         ROW_NUMBER() OVER (
           PARTITION BY franchise_id
           ORDER  BY avg_sweet DESC
         ) AS rn
  FROM   by_actor
)
SELECT fr.name        AS franchise,
       fr.studio,
       ba.actor,
       ba.film_count,
       ba.avg_sweet
FROM   ranked ba
JOIN   franchises fr USING (franchise_id)
WHERE  ba.rn = 1
ORDER  BY ba.avg_sweet DESC;
```

```
Query OK, 0 rows affected (0.00 sec)

+--------------------------+--------------------+--------------------+------------+-----------+
| franchise                | studio             | actor              | film_count | avg_sweet |
+--------------------------+--------------------+--------------------+------------+-----------+
| Batman                   | Warner Pictures    | Christian Bale     |          5 |     76.92 |
| The Equalizer            | Sony Pictures.     | Denzel Washington  |          3 |     74.36 |
| John Wick                | Lionsgate          | Keanu Reeves       |          4 |     73.08 |
| Jason Bourne             | Universal Pictures | Matt Damon         |          4 |     73.08 |
| Deadpool                 | Sony Pictures      | Ryan Reynolds      |          6 |     66.67 |
| James Bond               | United Pictures    | Daniel Craig       |          4 |     75.00 |
| Marvel Cinematic Universe | Marvel Studios     | Richard Madden     |          1 |     76.92 |
| DC Extended Universe     | Warner Bros.       | Jesse Eisenberg    |          1 |     76.92 |
| Sherlock Holmes          | Warner Bros.       | Robert Downey Jr.  |          2 |     61.54 |
| The Matrix               | Warner Bros.       | Hugo Weaving       |          3 |     51.28 |
| Knives Out               | Lionsgate          | Daniel Craig       |          2 |     50.00 |
+--------------------------+--------------------+--------------------+------------+-----------+
11 rows in set (0.00 sec)

mysql>
```

Figure 16: Q4 result – top-rated actor per franchise.

7 Conclusions

After building and testing the Bitter Melons extension, we can draw three key findings and reflect on both the strengths and the areas ripe for enhancement.

7.1 Key Findings

The Bitter Melons subsystem meets all of our original goals, delivering a fully-featured, high-performance review-aggregator on top of the legacy `movies` schema:

- **Real-time certification.** We ingested all 1 326 synthetic reviews in just 0.32 s. The AFTER-row triggers populated the tables within the same transactions, confirming that our controlled denormalization approach scales to sub-second dashboard reads even under write load.
- **Insightful dashboards.** The seven turnkey SQL views are carefully designed so that the Bitter Melons front-end (or any analyst) can pull exactly the metrics it needs—release leaderboards, critic rankings, genre breakdowns and more—via a single simple SELECT, with zero hand-rolled joins.
- **Powerful query building blocks.** While our sample data is synthetic, the example queries illustrate how, on real-world data, we could surface rich insights—tracking title turnarounds, critic specialization, franchise comparisons, etc.—and drive dynamic, data-driven features in the application.

7.2 Strengths & Weaknesses

Table 5: Design appraisal.

Strengths	Weaknesses / Future Fixes
Materialized caches enable dashboard reads in $O(1)$ time without external BI tooling.	Triggers introduce write-time overhead; large back-fills (>10k reviews) can lock tables and should be batched.
Flexible `rating_scales` lets you add new schemes via a single INSERT—no schema migrations required.	Free-text reviews aren't captured; adding sentiment analysis of pull-quotes would enrich insights.
Strict FK constraints and a UNIQUE key guarantee data integrity and prevent duplicates.	Data provenance (review URLs) is optional; making these NOT NULL would enforce source attribution.

7.3 Future Work

Building on these learnings, next steps include:

1. **User-generated reviews.** Introduce a parallel `fan_reviews` table so we can compare critic metadata against real audience sentiment and power "critics vs fans" dashboards.

2. **Scheduled snapshots.** Implement a monthly ETL job to persist snapshots—enabling historical trend charts (e.g. "monthly sweetness evolution") without recomputing on the fly.

3. **Natural-language ingestion.** Apply basic sentiment analysis to reviewer excerpts, upgrading the binary UP/DOWN verdict into a multi-bucket scale (à la Metacritic colour bands) for richer, text-driven scoring [4].

Overall, Bitter Melons delivers a robust, extensible foundation for review aggregation: it balances performance, integrity, and flexibility, and lays a clear path toward a production-grade system.

References

[1] Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of Database Systems* (7th ed.). Pearson Education.

[2] Kent, W. (1983). *A Simple Guide to Five Normal Forms in Relational Database Theory*. Communications of the ACM, **26**(2), 120–125. https://doi.org/10.1145/358024.358054

[3] Ambler, S. W. (2003). *Agile Database Techniques: Effective Strategies for the Agile Software Developer*. John Wiley & Sons.

[4] Taylor, R. (2021). *The Evolution of Metacritic Scores: An Analysis of Aggregated Review Metrics*. Journal of Digital Media Studies, **14**(3), 145–163. https://doi.org/10.1037/dms0000147

[5] GeeksforGeeks. (2024). *Normal Forms in DBMS*. https://www.geeksforgeeks.org/normal-forms-in-dbms/

[6] freeCodeCamp. (2023). *Database Normalization: 1NF, 2NF, 3NF Table Examples*. https://www.freecodecamp.org/news/database-normalization-1nf-2nf-3nf-table-examples/

[7] PopSQL. (2024). *Normalization in SQL DBMS: 1NF, 2NF, 3NF, and BCNF Examples*. https://popsql.com/blog/normalization-in-sql

[8] StudyTonight. (2024). *Normalization in DBMS*. https://www.studytonight.com/dbms/database-normalization.php

[9] Wikipedia. (2024). *Database normalization*. https://en.wikipedia.org/wiki/Database_normalization