Leaderboards

Namazu Elements Core Features · https://namazustudios.com/docs/namazu-elements-core/leaderboards/

Elements supports **Leaderboards** for tracking player scores and displaying ranked results. A leaderboard defines the rules for *how scores are tracked over time* and *how new scores are applied*.

There are three primary resources involved:

- **Leaderboard** – the configuration (name, reset cadence, score strategy, etc.)
- **Score** – a player's score entry on a leaderboard
- **Rank / RankRow** – a ranked view of scores (position + score/user info)

> Before working with Scores or Ranks, you must create a Leaderboard.

Leaderboard Properties

A **Leaderboard** has the following properties:

- **id** – Database ID assigned when created.
- **name** – Unique identifier for the leaderboard.
- **title** – Player-facing title shown in UI.
- **scoreUnits** – Units label for the score ("points", "coins", etc.).
- **timeStrategyType** – Controls whether the leaderboard resets:
  - **ALL_TIME** – does not reset
  - **EPOCHAL** – resets on a fixed interval
- **scoreStrategyType** – Controls how new scores update existing ones:
  - **OVERWRITE_IF_GREATER** – keeps `max(old, new)`
  - **ACCUMULATE** – keeps `old + new`
- **firstEpochTimestamp** (EPOCHAL only) – epoch start time (milliseconds since epoch).

- **epochInterval** (EPOCHAL only) – epoch duration in milliseconds.

If either `firstEpochTimestamp` or `epochInterval` is set during creation, the other must also be provided.

Creating and Managing Leaderboards (Server-Side)

On the server, you can manage leaderboards using the `LeaderboardDao`.

Create a Leaderboard

```
import dev.getelements.elements.sdk.dao.LeaderboardDao;
import dev.getelements.elements.sdk.model.leaderboard.Leaderboard;

public class LeaderboardsService {

    @Inject
    private LeaderboardDao leaderboardDao;

    public Leaderboard createAllTimeHighScoreLeaderboard() {
        Leaderboard lb = new Leaderboard();
        lb.setName("global_high_score");
        lb.setTitle("Global High Score");
        lb.setScoreUnits("points");

        lb.setTimeStrategyType(Leaderboard.TimeStrategyType.ALL_TIME);
        lb.setScoreStrategyType(Leaderboard.ScoreStrategyType.OVERWRITE_IF_GREATER);

        return leaderboardDao.createLeaderboard(lb);
    }
}

    public Leaderboard createAllTimeHighScoreLeaderboard() {
        Leaderboard lb = new Leaderboard();
        lb.setName("global_high_score");
        lb.setTitle("Global High Score");
        lb.setScoreUnits("points");

        lb.setTimeStrategyType(Leaderboard.TimeStrategyType.ALL_TIME);
        lb.setScoreStrategyType(Leaderboard.ScoreStrategyType.OVERWRITE_IF_GREATER);
```

```
        return leaderboardDao.createLeaderboard(lb);
    }
}
```

Create an Epochal (Resetting) Leaderboard

```
import java.time.Instant;

public Leaderboard createWeeklyLeaderboard() {
    Leaderboard lb = new Leaderboard();
    lb.setName("weekly_points");
    lb.setTitle("Weekly Points");
    lb.setScoreUnits("points");

    lb.setTimeStrategyType(Leaderboard.TimeStrategyType.EPOCHAL);
    lb.setScoreStrategyType(Leaderboard.ScoreStrategyType.ACCUMULATE);

    // Example: start epochs at a known boundary.
    long firstEpochMs = Instant.parse("2026-01-01T00:00:00Z").toEpochMilli();
    long oneWeekMs = 7L * 24 * 60 * 60 * 1000;

    lb.setFirstEpochTimestamp(firstEpochMs);
    lb.setEpochInterval(oneWeekMs);

    return leaderboardDao.createLeaderboard(lb);
}
```

List and Fetch Leaderboards

```
import dev.getelements.elements.sdk.model.Pagination;

public Pagination<Leaderboard> listLeaderboards(int offset, int count) {
    return leaderboardDao.getLeaderboards(offset, count);
}

public Leaderboard getLeaderboard(String nameOrId) {
    return leaderboardDao.getLeaderboard(nameOrId);
}
```

Update and Delete a Leaderboard

```
public Leaderboard renameTitle(String nameOrId, String newTitle) {
    Leaderboard lb = leaderboardDao.getLeaderboard(nameOrId);
    lb.setTitle(newTitle);
    return leaderboardDao.updateLeaderboard(nameOrId, lb);
}

public void deleteLeaderboard(String nameOrId) {
    leaderboardDao.deleteLeaderboard(nameOrId);
}
```

End-to-End Example (Server-Side)

Below is a consolidated example that shows a typical server-side flow:

1. Create (or fetch) a leaderboard
2. Post a score for a profile
3. Fetch ranked results (global + relative)
4. Optionally fetch a UI-friendly tabular view

```
import dev.getelements.elements.sdk.dao.LeaderboardDao;
import dev.getelements.elements.sdk.dao.RankDao;
import dev.getelements.elements.sdk.dao.ScoreDao;
import dev.getelements.elements.sdk.model.Pagination;
import dev.getelements.elements.sdk.model.Tabulation;
import dev.getelements.elements.sdk.model.leaderboard.Leaderboard;
import dev.getelements.elements.sdk.model.leaderboard.Rank;
import dev.getelements.elements.sdk.model.leaderboard.RankRow;
import dev.getelements.elements.sdk.model.leaderboard.Score;
import dev.getelements.elements.sdk.model.profile.Profile;

public class LeaderboardFlow {

    @Inject
    private LeaderboardDao leaderboardDao;
```

```
    @Inject
    private ScoreDao scoreDao;

    @Inject
    private RankDao rankDao;

    public void runExample(Profile profile, double points) {

        // 1) Create (or fetch) the leaderboard configuration.
        // If your deployment provisions leaderboards up-front, you can skip creation and just
call getLeaderboard(...).
        Leaderboard lb = new Leaderboard();
        lb.setName("global_high_score");
        lb.setTitle("Global High Score");
        lb.setScoreUnits("points");
        lb.setTimeStrategyType(Leaderboard.TimeStrategyType.ALL_TIME);
        lb.setScoreStrategyType(Leaderboard.ScoreStrategyType.OVERWRITE_IF_GREATER);

        try {
            leaderboardDao.createLeaderboard(lb);
        } catch (Exception ignored) {
            // Leaderboard likely already exists.
        }

        Leaderboard resolved = leaderboardDao.getLeaderboard("global_high_score");

        // 2) Post a score for this profile.
        Score score = new Score();
        score.setProfile(profile);
        score.setPointValue(points);

        scoreDao.createOrUpdateScore(resolved.getId(), score);

        long epoch = 0L;

        // 3a) Fetch global ranks (top N).
        Pagination<Rank> global = rankDao.getRanksForGlobal(resolved.getId(), 0, 25, epoch);

        // 3b) Fetch ranks relative to the current profile.
        Pagination<Rank> relative = rankDao.getRanksForGlobalRelative(
                resolved.getId(),
```

```
                profile.getId(),
                0,
                25,
                epoch
        );

        // 4) Optional: fetch a UI-friendly table of rows.
        Tabulation<RankRow> tabular = rankDao.getRanksForGlobalTabular(resolved.getId(), epoch);
    }
}
```

Posting Scores

A **Score** records a player's entry on a leaderboard.

Typical usage:

- Post a score when a match ends or an achievement occurs.
- Let the leaderboard's **scoreStrategyType** decide whether to overwrite or accumulate.

Example (Server-Side) – Create or Update a Score

On the server, use `ScoreDao#createOrUpdateScore(...)` to write a player's score for a leaderboard. If a score already exists for the same **leaderboard + profile**, the existing row is updated rather than creating a duplicate.

```
import dev.getelements.elements.sdk.dao.ScoreDao;
import dev.getelements.elements.sdk.model.leaderboard.Score;
import dev.getelements.elements.sdk.model.profile.Profile;

public class ScoresService {

    @Inject
    private ScoreDao scoreDao;

    public Score postScore(String leaderboardNameOrId, Profile profile, double points) {
        Score score = new Score();
        score.setProfile(profile);
```

```
        score.setPointValue(points);

        // Server assigns creationTimestamp + leaderboardEpoch.
        return scoreDao.createOrUpdateScore(leaderboardNameOrId, score);
    }
}

    public Score postScore(String leaderboardNameOrId, Profile profile, double points) {
        Score score = new Score();
        score.setProfile(profile);
        score.setPointValue(points);

        // Server assigns creationTimestamp + leaderboardEpoch.
        return scoreDao.createOrUpdateScore(leaderboardNameOrId, score);
    }
}
```

Reading Ranks

A **Rank** pairs a `position` (1st, 2nd, 3rd, …) with a `Score` object. In many UIs you'll display either:

- a **global** leaderboard page (top N)
- a **relative** page (N entries centered around the current user)
- a **friends/followers** page (filtered to a social graph)

On the server, use `RankDao` to retrieve ranked views of scores.

Example – Get Global Ranks

```
import dev.getelements.elements.sdk.dao.RankDao;
import dev.getelements.elements.sdk.model.Pagination;
import dev.getelements.elements.sdk.model.leaderboard.Rank;

public class RanksService {

    @Inject
    private RankDao rankDao;
```

```
    public Pagination<Rank> getGlobalRanks(String leaderboardNameOrId, int offset, int count, long
epoch) {
        return rankDao.getRanksForGlobal(leaderboardNameOrId, offset, count, epoch);
    }
}

    public Pagination<Rank> getGlobalRanks(String leaderboardNameOrId, int offset, int count, long
epoch) {
        return rankDao.getRanksForGlobal(leaderboardNameOrId, offset, count, epoch);
    }
}
```

Example – Get Ranks Relative to a Profile

This pattern is useful for “show me around me” UI, where the player always sees themselves in the list.

```
import dev.getelements.elements.sdk.model.Pagination;
import dev.getelements.elements.sdk.model.leaderboard.Rank;

public Pagination<Rank> getRelativeGlobalRanks(String leaderboardNameOrId,
                                               String profileId,
                                               int offset, int count,
                                               long epoch) {
    return rankDao.getRanksForGlobalRelative(leaderboardNameOrId, profileId, offset, count,
epoch);
}
```

Example – Friends and Followers Filters

Depending on your social model, you can query ranks filtered to:

- **Friends** via `getRanksForFriends(...)` or `getRanksForFriendsRelative(...)`
- **Mutual followers** via `getRanksForMutualFollowers(...)` or `getRanksForMutualFollowersRelative(...)`

```
public Pagination<Rank> getFriendsRanks(String leaderboardNameOrId,
                                        String profileId,
                                        int offset, int count,
                                        long epoch) {
```

```
    return rankDao.getRanksForFriends(leaderboardNameOrId, profileId, offset, count, epoch);
}
```

Tabular (UI-Friendly) Output

If you want to render a leaderboard without additional profile lookups, you can request a tabular view of `RankRow`, which includes profile display fields (id, display name, image URL, last login) alongside the score and position.

```
import dev.getelements.elements.sdk.model.Tabulation;
import dev.getelements.elements.sdk.model.leaderboard.RankRow;

public Tabulation<RankRow> getGlobalRanksTabular(String leaderboardNameOrId, long epoch) {
    return rankDao.getRanksForGlobalTabular(leaderboardNameOrId, epoch);
}
```

> For **ALL_TIME** leaderboards, `leaderboardEpoch` is typically `0` by convention.

End-to-End Example

Below is a complete server-side flow showing how leaderboards are typically used together:

1. Create (or fetch) a leaderboard
2. Post a score for a player
3. Read back ranked results

This mirrors the most common production setup.

Example – Create Leaderboard, Post Score, Fetch Ranks

```
import jakarta.inject.Inject;
import dev.getelements.elements.sdk.dao.LeaderboardDao;
import dev.getelements.elements.sdk.dao.ScoreDao;
import dev.getelements.elements.sdk.dao.RankDao;
import dev.getelements.elements.sdk.model.Pagination;
import dev.getelements.elements.sdk.model.leaderboard.Leaderboard;
```

```
import dev.getelements.elements.sdk.model.leaderboard.Score;
import dev.getelements.elements.sdk.model.leaderboard.Rank;
import dev.getelements.elements.sdk.model.profile.Profile;

public class LeaderboardFlowService {

    private static final Logger logger = LoggerFactory.getLogger(LeaderboardFlowService.class);

    @Inject
    private final LeaderboardDao leaderboardDao;

    @Inject
    private final ScoreDao scoreDao;

    @Inject
    private final RankDao rankDao;

    public void runExample(Profile profile) {

        // 1. Create or fetch leaderboard
        Leaderboard leaderboard = leaderboardDao.getLeaderboard("weekly_points");

        if (leaderboard == null) {
            leaderboard = new Leaderboard();
            leaderboard.setName("weekly_points");
            leaderboard.setTitle("Weekly Points");
            leaderboard.setScoreUnits("points");
            leaderboard.setTimeStrategyType(Leaderboard.TimeStrategyType.EPOCHAL);
            leaderboard.setScoreStrategyType(Leaderboard.ScoreStrategyType.ACCUMULATE);

            leaderboard = leaderboardDao.createLeaderboard(leaderboard);
        }

        // 2. Post a score for the player
        Score score = new Score();
        score.setProfile(profile);
        score.setPointValue(250);

        scoreDao.createOrUpdateScore(leaderboard.getId(), score);

        // 3. Fetch top 10 ranks for the current epoch
```

```
        long currentEpoch = score.getLeaderboardEpoch();

        Pagination<Rank> ranks = rankDao.getRanksForGlobal(
                leaderboard.getId(),
                0,
                10,
                currentEpoch
        );

        for (Rank rank : ranks.getItems()) {
            logger.debug(rank.getPosition() + ": " +
                                rank.getScore().getProfile().getId() +
                                " : " + rank.getScore().getPointValue());
        }
    }
}
```

Notes and Best Practices

- **Use unique, stable leaderboard names**; treat them as durable identifiers.
- Pick **OVERWRITE_IF_GREATER** for “high score” style boards, and **ACCUMULATE** for “total points” style boards.
- Prefer **EPOCHAL** leaderboards for seasonal ladders and weekly/monthly competitions.
- When building UI, consider using **relative rank queries** so players always see themselves in the list.