Issue #6: Add leaderboard viewing command

Labels: `enhancement`, `feature`
Milestone: Feature Complete
Estimated Effort: 6-8 hours

Description

Enhance the existing `/leaderboard` command with additional viewing options, filtering capabilities, pagination, and improved formatting. Add a new command `/score` for individual user lookups and statistics.

Current Implementation

Currently, the bot has:
- ✅ `/leaderboard` - Shows top 10 users with medal emojis (🥇🥈🥉)
- ✅ Basic score display
- ❌ No pagination for large leaderboards
- ❌ No filtering options
- ❌ No user statistics
- ❌ No historical data

Proposed Enhancements

Enhanced /leaderboard Command Features

1. **Pagination** - Navigate through all users, not just top 10
2. **Time Filters** - Show leaderboard for specific time periods
3. **Channel Filters** - Show leaderboard for specific channels
4. **Direction Filters** - Show bottom performers (++ only users, – only users)
5. **Interactive Buttons** - Use Slack blocks for better UX
6. **Export Options** - Generate CSV/JSON exports

New Commands

- `/score @user` - Show detailed score for specific user
- `/score me` - Show your own detailed score
- `/stats` - Show overall system statistics
- `/history @user` - Show vote history for a user (if vote tracking enabled)

Tasks

Phase 1: Enhanced Leaderboard Display (2-3 hours)

- [ ] Add pagination support (Next/Previous buttons)
- [ ] Improve formatting with Slack Block Kit
- [ ] Add emoji indicators for rank changes
- [ ] Show percentage vs top scorer

- [ ] Add timestamp of last update

Phase 2: Filtering Options (2-3 hours)

- [ ] Add time period filters (today, week, month, all-time)
- [ ] Add channel-specific leaderboards
- [ ] Add top/bottom toggle
- [ ] Add minimum score threshold filter

Phase 3: User Statistics (2-3 hours)

- [ ] Implement `/score` command for individual lookups
- [ ] Show user rank, percentile, and trend
- [ ] Add vote history summary
- [ ] Show comparison with other users

Phase 4: Interactive Components (2-3 hours)

- [ ] Add Slack buttons for navigation
- [ ] Add dropdown menus for filters
- [ ] Implement button click handlers
- [ ] Add loading states and confirmations

Implementation Details

Enhanced Leaderboard with Blocks

```
// Enhanced leaderboard command
app.command('/leaderboard', async ({ command, ack, say, client }) => {
  await ack();

  try {
    // Get parameters from command text
    const params = parseLeaderboardParams(command.text);
    const { page = 1, limit = 10, filter = 'all-time', channel = null } = params;

    // Get leaderboard data from database
    const { users, total, hasMore } = await getLeaderboardData({
      page,
      limit,
      filter,
      channel,
    });

    // Build Slack blocks
    const blocks = buildLeaderboardBlocks(users, { page, total, hasMore, filter });

    await say({
      blocks,
      text: 'Leaderboard', // Fallback text
    });
  } catch (error) {
    logger.error('Leaderboard command error:', error);
    await say('❌ Failed to fetch leaderboard. Please try again.');
  }
});

// Parse leaderboard parameters
function parseLeaderboardParams(text) {
  const params = {
    page: 1,
    limit: 10,
    filter: 'all-time',
    channel: null,
  };

  // Parse page number: /leaderboard page:2
  const pageMatch = text.match(/page:(\d+)/i);
  if (pageMatch) params.page = parseInt(pageMatch[1]);

  // Parse filter: /leaderboard filter:week
  const filterMatch = text.match(/filter:(today|week|month|all-time)/i);
  if (filterMatch) params.filter = filterMatch[1].toLowerCase();

  // Parse channel: /leaderboard channel:#general
  const channelMatch = text.match(/channel:<#([A-Z0-9]+)\|.*?>/);
  if (channelMatch) params.channel = channelMatch[1];

  return params;
}
```

Leaderboard Blocks Builder

```
function buildLeaderboardBlocks(users, options) {
  const { page, total, hasMore, filter } = options;
  const startRank = (page - 1) * 10 + 1;

  const blocks = [
    // Header
    {
      type: 'header',
      text: {
        type: 'plain_text',
        text: `🏆 Leaderboard ${filter !== 'all-time' ? `(${filter})` : ''}`,
        emoji: true,
      },
    },
    {
      type: 'context',
      elements: [
        {
          type: 'mrkdwn',
          text: `Page ${page} • Total users: ${total} • Last updated: ${new Date().to-
LocaleString()}`,
        },
      ],
    },
    {
      type: 'divider',
    },
  ];

  // User list
  users.forEach((user, index) => {
    const rank = startRank + index;
    const medal = rank === 1 ? '🥇' : rank === 2 ? '🥈' : rank === 3 ? '🥉' : `$
{rank}.`;
    const trend = user.trend > 0 ? '📈' : user.trend < 0 ? '📉' : '➡️';
    const percentage = total > 0 ? Math.round((user.score / users[0].score) * 100) :
0;

    blocks.push({
      type: 'section',
      text: {
        type: 'mrkdwn',
        text: `${medal} <@${user.userId}> • *${user.score}* points ${trend}`,
      },
      accessory: {
        type: 'button',
        text: {
          type: 'plain_text',
          text: 'View Details',
          emoji: true,
        },
        value: user.userId,
        action_id: 'view_user_details',
      },
    });

    // Progress bar
    if (rank <= 10) {
      const progressBar = '▓'.repeat(Math.floor(percentage / 10)) +
                          '░'.repeat(10 - Math.floor(percentage / 10));
      blocks.push({
        type: 'context',
```

```
        elements: [
          {
            type: 'mrkdwn',
            text: `${progressBar} ${percentage}%`,
          },
        ],
      });
    }
  });

  blocks.push({ type: 'divider' });

  // Navigation buttons
  const navigationButtons = {
    type: 'actions',
    elements: [],
  };

  if (page > 1) {
    navigationButtons.elements.push({
      type: 'button',
      text: {
        type: 'plain_text',
        text: '⬅️ Previous',
        emoji: true,
      },
      value: `page:${page - 1}`,
      action_id: 'leaderboard_prev',
    });
  }

  if (hasMore) {
    navigationButtons.elements.push({
      type: 'button',
      text: {
        type: 'plain_text',
        text: 'Next ➡️',
        emoji: true,
      },
      value: `page:${page + 1}`,
      action_id: 'leaderboard_next',
    });
  }

  // Filter dropdown
  navigationButtons.elements.push({
    type: 'static_select',
    placeholder: {
      type: 'plain_text',
      text: 'Filter',
      emoji: true,
    },
    options: [
      {
        text: { type: 'plain_text', text: 'All Time', emoji: true },
        value: 'all-time',
      },
      {
        text: { type: 'plain_text', text: 'This Month', emoji: true },
        value: 'month',
      },
      {
        text: { type: 'plain_text', text: 'This Week', emoji: true },
```

```
        value: 'week',
      },
      {
        text: { type: 'plain_text', text: 'Today', emoji: true },
        value: 'today',
      },
    ],
    action_id: 'leaderboard_filter',
  });

  blocks.push(navigationButtons);

  return blocks;
}
```

Database Queries for Enhanced Features

```
// Get leaderboard with pagination and filtering
async function getLeaderboardData({ page = 1, limit = 10, filter = 'all-time',
channel = null }) {
  const offset = (page - 1) * limit;

  // Build WHERE clause for filtering
  let whereClause = 'WHERE score != 0';
  const params = [limit, offset];

  if (filter !== 'all-time') {
    const timeFilter = getTimeFilter(filter);
    whereClause += ` AND updated_at >= $${params.length + 1}`;
    params.push(timeFilter);
  }

  if (channel) {
    // If vote_history table exists and tracks channels
    whereClause += ` AND EXISTS (
      SELECT 1 FROM vote_history
      WHERE vote_history.voted_user_id = leaderboard.user_id
      AND vote_history.channel_id = $${params.length + 1}
    )`;
    params.push(channel);
  }

  // Get total count
  const countResult = await pool.query(
    `SELECT COUNT(*) as total FROM leaderboard ${whereClause}`,
    params.slice(2) // Exclude limit and offset
  );
  const total = parseInt(countResult.rows[0].total);

  // Get leaderboard data
  const result = await pool.query(
    `SELECT
      user_id,
      score,
      (score - LAG(score, 1, score) OVER (ORDER BY score DESC)) as trend
     FROM leaderboard
     ${whereClause}
     ORDER BY score DESC
     LIMIT $1 OFFSET $2`,
    params
  );

  return {
    users: result.rows,
    total,
    hasMore: (offset + limit) < total,
  };
}

// Get time filter timestamp
function getTimeFilter(filter) {
  const now = new Date();
  switch (filter) {
    case 'today':
      return new Date(now.setHours(0, 0, 0, 0));
    case 'week':
      const weekAgo = new Date(now);
      weekAgo.setDate(weekAgo.getDate() - 7);
      return weekAgo;
```

```
    case 'month':
      const monthAgo = new Date(now);
      monthAgo.setMonth(monthAgo.getMonth() - 1);
      return monthAgo;
    default:
      return new Date(0); // Beginning of time
  }
}
```

Individual User Score Command

```
app.command('/score', async ({ command, ack, say, client }) => {
  await ack();

  try {
    // Parse user ID from command text
    let userId = command.text.match(/<@([A-Z0-9]+)>/)?.[1];

    // If no user specified or "me", use command invoker
    if (!userId || command.text.trim() === 'me') {
      userId = command.user_id;
    }

    // Get user stats
    const stats = await getUserStats(userId);
    const userInfo = await client.users.info({ user: userId });

    // Build response blocks
    const blocks = [
      {
        type: 'header',
        text: {
          type: 'plain_text',
          text: `📊 Stats for ${userInfo.user.real_name || userInfo.user.name}`,
          emoji: true,
        },
      },
      {
        type: 'section',
        fields: [
          {
            type: 'mrkdwn',
            text: `*Current Score:*\n${stats.score}`,
          },
          {
            type: 'mrkdwn',
            text: `*Rank:*\n#${stats.rank} of ${stats.totalUsers}`,
          },
          {
            type: 'mrkdwn',
            text: `*Percentile:*\n${stats.percentile}%`,
          },
          {
            type: 'mrkdwn',
            text: `*Trend (7 days):*\n${stats.weekTrend > 0 ? '📈' : stats.weekTrend
< 0 ? '📉' : '➡️'} ${stats.weekTrend > 0 ? '+' : ''}${stats.weekTrend}`,
          },
        ],
      },
      {
        type: 'section',
        text: {
          type: 'mrkdwn',
          text: `*Vote Summary:*\n` +
                `↑ Received: ${stats.upvotesReceived} • ↓ Received: ${stats.downvotesR
eceived}\n` +
                `↑ Given: ${stats.upvotesGiven} • ↓ Given: ${stats.downvotesGiven}`,
        },
      },
    ];

    await say({ blocks, text: `Stats for ${userInfo.user.name}` });
  } catch (error) {
```

```
    logger.error('Score command error:', error);
    await say('❌ Failed to fetch user stats. Please try again.');
  }
});

// Get comprehensive user statistics
async function getUserStats(userId) {
  // Get current score and rank
  const rankResult = await pool.query(
    `SELECT
      user_id,
      score,
      (SELECT COUNT(*) + 1 FROM leaderboard l2 WHERE l2.score > l1.score) as rank,
      (SELECT COUNT(*) FROM leaderboard WHERE score != 0) as total_users
     FROM leaderboard l1
     WHERE user_id = $1`,
    [userId]
  );

  const { score = 0, rank = 0, total_users = 0 } = rankResult.rows[0] || {};
  const percentile = total_users > 0 ? Math.round((1 - (rank - 1) / total_users) *
100) : 0;

  // Get vote history (if table exists)
  const voteHistory = await pool.query(
    `SELECT
      COUNT(*) FILTER (WHERE vote_type = '++' AND voted_user_id = $1) as up-
votes_received,
      COUNT(*) FILTER (WHERE vote_type = '--' AND voted_user_id = $1) as down-
votes_received,
      COUNT(*) FILTER (WHERE vote_type = '++' AND voter_id = $1) as upvotes_given,
      COUNT(*) FILTER (WHERE vote_type = '--' AND voter_id = $1) as downvotes_given
     FROM vote_history
     WHERE voted_user_id = $1 OR voter_id = $1`,
    [userId]
  );

  // Get 7-day trend
  const trendResult = await pool.query(
    `SELECT
      COALESCE(SUM(CASE WHEN vote_type = '++' THEN 1 WHEN vote_type = '--' THEN -1
END), 0) as week_trend
     FROM vote_history
     WHERE voted_user_id = $1
     AND created_at >= NOW() - INTERVAL '7 days'`,
    [userId]
  );

  return {
    score,
    rank,
    totalUsers: total_users,
    percentile,
    weekTrend: trendResult.rows[0]?.week_trend || 0,
    upvotesReceived: voteHistory.rows[0]?.upvotes_received || 0,
    downvotesReceived: voteHistory.rows[0]?.downvotes_received || 0,
    upvotesGiven: voteHistory.rows[0]?.upvotes_given || 0,
    downvotesGiven: voteHistory.rows[0]?.downvotes_given || 0,
  };
}
```

Button Interaction Handlers

```
// Handle pagination button clicks
app.action('leaderboard_next', async ({ ack, action, say }) => {
  await ack();
  const page = parseInt(action.value.split(':')[1]);
  // Re-fetch and display leaderboard for new page
  // ... (similar to /leaderboard command)
});

app.action('leaderboard_prev', async ({ ack, action, say }) => {
  await ack();
  const page = parseInt(action.value.split(':')[1]);
  // Re-fetch and display leaderboard for new page
});

// Handle filter dropdown
app.action('leaderboard_filter', async ({ ack, action, say }) => {
  await ack();
  const filter = action.selected_option.value;
  // Re-fetch and display leaderboard with new filter
});

// Handle view user details button
app.action('view_user_details', async ({ ack, action, client, body }) => {
  await ack();
  const userId = action.value;

  // Get user stats and show modal
  const stats = await getUserStats(userId);
  const userInfo = await client.users.info({ user: userId });

  await client.views.open({
    trigger_id: body.trigger_id,
    view: {
      type: 'modal',
      title: {
        type: 'plain_text',
        text: 'User Details',
      },
      blocks: [
        // ... (detailed user stats blocks)
      ],
    },
  });
});
```

Testing Strategy

Manual Testing

- [ ] Test `/leaderboard` with no parameters
- [ ] Test `/leaderboard page:2`
- [ ] Test `/leaderboard filter:week`
- [ ] Test `/leaderboard channel:#general`
- [ ] Test `/score @user`
- [ ] Test `/score me`
- [ ] Test pagination buttons

- [ ] Test filter dropdown
- [ ] Test with empty leaderboard
- [ ] Test with large leaderboard (100+ users)

Unit Tests

```
describe('Leaderboard enhancements', () => {
  test('parseLeaderboardParams extracts page number', () => {
    const params = parseLeaderboardParams('page:3');
    expect(params.page).toBe(3);
  });

  test('parseLeaderboardParams extracts filter', () => {
    const params = parseLeaderboardParams('filter:week');
    expect(params.filter).toBe('week');
  });

  test('getTimeFilter returns correct date for week', () => {
    const filter = getTimeFilter('week');
    const weekAgo = new Date();
    weekAgo.setDate(weekAgo.getDate() - 7);
    expect(filter.getDate()).toBe(weekAgo.getDate());
  });
});
```

Acceptance Criteria

- [ ] `/leaderboard` shows top 10 users with improved formatting
- [ ] Pagination works correctly with Next/Previous buttons
- [ ] Time filters (today, week, month, all-time) work correctly
- [ ] `/score @user` shows comprehensive user statistics
- [ ] `/score me` shows stats for command invoker
- [ ] Interactive buttons and dropdowns work correctly
- [ ] All database queries are optimized and indexed
- [ ] Error handling works for edge cases
- [ ] Documentation updated with new command examples
- [ ] Unit tests cover all new functionality

Performance Considerations

- Add database indexes for:
- `leaderboard.score DESC`
- `leaderboard.updated_at`
- `vote_history.voted_user_id`
- `vote_history.created_at DESC`
- Cache leaderboard data for 1-5 minutes
- Limit pagination to reasonable bounds (max 100 pages)

Documentation Updates

- Update EXAMPLES.md with new command examples

- Update README.md with feature descriptions
- Add screenshots of new Slack blocks
- Document command parameters and options

Future Enhancements (Out of Scope)

- Export leaderboard to CSV/JSON
- Schedule automatic leaderboard posts
- Gamification (achievements, badges)
- Team/department leaderboards
- Comparative analytics

Resources

- Slack Block Kit Builder (https://app.slack.com/block-kit-builder)
- Slack API: Block Kit (https://api.slack.com/block-kit)
- Slack API: Interactivity (https://api.slack.com/interactivity)
- PostgreSQL Window Functions (https://www.postgresql.org/docs/current/tutorial-window.html)

Dependencies

Requires:
- Issue #2: PostgreSQL integration (for advanced queries)

Recommended:
- Issue #7: Health check endpoint (for monitoring)