FEATURE IMPLEMENTATION - LATEST METRICS AND COMPARE METRICS BUTTONS

Overview

The Latest Metrics and Compare Metrics features in PGWatch were developed to provide users with quick access to the most recent metrics for any monitored database and the ability to compare metrics between databases. These enhancements introduce one-click solutions to retrieve, display, and compare current metrics while integrating seamlessly with PGWatch's existing architecture.

Implementation Approach

The features were implemented through modifications to both frontend and backend components:

1. Frontend: New React components for displaying metrics in modal dialogs and comparing metrics

2. Backend: New API endpoints optimized for retrieving latest metrics and batch metrics

3. Database: Optimized SQL queries for efficient data retrieval

Technical Impact

The implementation required changes to:

- Frontend component structure

- Backend API endpoints

- Authentication flow

- Database queries

LATEST METRICS IMPLEMENTATION

Frontend Changes

1. Package.json Updates

Location: `internal/webui/package.json`

Description: Added new dependencies required for the Latest Metrics feature.

```
{
  "dependencies": {
    // ... existing dependencies ...
    "jspdf": "^2.5.1"
  }
}
```

2. LatestMetricsButton Component

Location: `internal/webui/src/components/LatestMetricsButton/LatestMetricsButton.tsx`

Description: A new React component that implements the Latest Metrics Button functionality.

Key Features:

- Modal dialog with responsive table layout
- JSON and CSV export functionality
- Loading states and error handling
- Integration with existing authentication
- Formatting of metric values for better readability

Implementation Details:

- Uses Material-UI components for consistent styling
- Implements data fetching with error handling
- Provides export functionality for metrics data
- Maintains loading states for better UX
- Integrates with the existing token-based authentication

Key Code Changes:

```
// New component implementation

export const LatestMetricsButton: React.FC<LatestMetricsButtonProps> =
({ dbname }) => {

  const [open, setOpen] = useState(false);

  const [metrics, setMetrics] = useState<MetricData>({});

  const [error, setError] = useState<string>('');

  const [loading, setLoading] = useState(false);


  const handleClickOpen = async () => {

    setLoading(true);

    setError('');

    try {

      const token = getToken();

      const response = await fetch(`/latest-
metrics?dbname=${encodeURIComponent(dbname)}`, {

        headers: {

          'Token': token || '',

        },

      });

      if (!response.ok) {
```

```
        const errorText = await response.text();
        throw new Error(errorText || 'Failed to fetch metrics');
      }
      const data = await response.json();
      if (Object.keys(data).length === 0) {
        setError('No metrics data available');
      } else {
        setMetrics(data);
      }
    } catch (err) {
      setError(err instanceof Error ? err.message : 'Failed to fetch
metrics');
      setMetrics({});
    } finally {
      setLoading(false);
      setOpen(true);
    }
  };

  const handleClose = () => {
    setOpen(false);
    setMetrics({});
    setError('');
  };

  const handleDownloadJSON = () => {
    try {
      const dataStr = JSON.stringify(metrics, null, 2);
      const blob = new Blob([dataStr], { type: 'application/json' });
```

```
      const url = URL.createObjectURL(blob);
      const link = document.createElement('a');
      link.href = url;
      link.download = `${dbname}_metrics.json`;
      link.click();
      URL.revokeObjectURL(url);
    } catch (err) {
      console.error('JSON export failed:', err);
      setError('JSON export failed');
    }
  };

  const handleDownloadCSV = () => {
    try {
      let csvContent = "Metric,Value\n";

      const flattenObject = (obj: MetricData, prefix = ''): string[] =>
{
        return Object.entries(obj).flatMap(([key, value]) => {
          const newKey = prefix ? `${prefix}.${key}` : key;
          if (typeof value === 'object' && value !== null) {
            return flattenObject(value as MetricData, newKey);
          } else {
            return [`"${newKey}","${value}"`];
          }
        });
      };

      const rows = flattenObject(metrics);
```

```
      csvContent += rows.join('\n');

      const blob = new Blob([csvContent], { type:
'text/csv;charset=utf-8;' });

      const url = URL.createObjectURL(blob);

      const link = document.createElement('a');

      link.href = url;

      link.download = `${dbname}_metrics.csv`;

      link.click();

      URL.revokeObjectURL(url);

    } catch (err) {

      console.error('CSV export failed:', err);

      setError('CSV export failed');

    }

  };


  const formatMetricValue = (value: any): string => {

    if (typeof value === 'number') {

      return value.toLocaleString();

    }

    return String(value);

  };


  const getMetricRows = (data: MetricData, prefix = ''): MetricRow[] =>
{

    return Object.entries(data).flatMap(([key, value]) => {

      const name = prefix ? `${prefix}.${key}` : key;

      if (typeof value === 'object' && value !== null) {

        return getMetricRows(value as MetricData, name);

      } else {
```

```
        return [{
          name,
          value: formatMetricValue(value)
        }];
      }
    });
  };

  const metricRows = getMetricRows(metrics);

  return (
    <Box sx={{ display: 'inline-block', ml: 1 }}>
      <Button
        variant="outlined"
        startIcon={<DownloadIcon />}
        onClick={handleClickOpen}
        size="small"
        disabled={loading}
      >
        Latest Metrics
      </Button>
      <Dialog open={open} onClose={handleClose} maxWidth="md"
fullWidth>
        <DialogTitle>Latest Metrics for {dbname}</DialogTitle>
        <DialogContent>
          {error ? (
            <Box sx={{ color: 'error.main', py: 2 }}>{error}</Box>
          ) : (
            <TableContainer component={Paper} sx={{ mt: 2 }}>
```

```
              <Table size="small">
                <TableHead>
                  <TableRow>
                    <TableCell>Metric</TableCell>
                    <TableCell align="right">Value</TableCell>
                  </TableRow>
                </TableHead>
                <TableBody>
                  {metricRows.map((row) => (
                    <TableRow key={row.name}>
                      <TableCell component="th" scope="row">
                        {row.name}
                      </TableCell>
                      <TableCell align="right">{row.value}</TableCell>
                    </TableRow>
                  ))}
                </TableBody>
              </Table>
            </TableContainer>
          )}
        </DialogContent>
        <DialogActions>
          <Button onClick={handleClose}>Close</Button>
          {Object.keys(metrics).length > 0 && (
            <>
              <Button onClick={handleDownloadJSON} variant="contained"
color="primary" startIcon={<DownloadIcon />}>
                Download JSON
              </Button>
```

```
              <Button onClick={handleDownloadCSV} variant="contained"
color="secondary" startIcon={<FileDownloadIcon />}>

                Download CSV

              </Button>

            </>

          )}

        </DialogActions>

      </Dialog>

    </Box>

  );

};
```

3. SourcesGrid Actions Update

Location: `internal/webui/src/pages/SourcesPage/components/SourcesGrid/components/SourcesGridActions.tsx`

Description: Modified the SourcesGrid actions to include the LatestMetricsButton.

Key Changes:

- Added LatestMetricsButton to the action buttons

- Maintained existing layout and styling

- Preserved existing functionality of other buttons

Key Code Changes:

```
//Other imports same

import { LatestMetricsButton } from
"components/LatestMetricsButton/LatestMetricsButton";

//Rest of code same
```

```
  return (

    <Box sx={{ display: 'flex', alignItems: 'center', gap: 1, width:
'100%', minWidth: 'fit-content', justifyContent: 'flex-end',

      overflowX: 'auto', whiteSpace: 'nowrap' }}>

      <GridActions handleEditClick={handleEditClick}
handleDeleteClick={handleDeleteClick}>

        <IconButton title="Copy" onClick={handleCopyClick}>

          <ContentCopyIcon />

        </IconButton>

      </GridActions>

      <LatestMetricsButton dbname={source.Name} />

      <WarningDialog open={dialogOpen} message={message}
onClose={handleDialogClose} onSubmit={handleSubmit} />

    </Box>

  );

};
```

Backend Changes

1. WebServer Route Registration

Location: `internal/webserver/webserver.go`

Description: Added the route registration for the new latest-metrics endpoint.

```
mux.Handle("/latest-metrics", NewEnsureAuth(s.handleLatestMetrics))
```

2. API Endpoint Handler

Location: `internal/webserver/metric.go`

Description: Added a new HTTP handler for the `/latest-metrics` endpoint.

Key Code Changes:

```
func (Server *WebUIServer) handleLatestMetrics(w http.ResponseWriter, r
*http.Request) {

    if r.Method != http.MethodGet {

        w.Header().Set("Allow", "GET")

        http.Error(w, "method not allowed", http.StatusMethodNotAllowed)

        return

    }



    dbname := r.URL.Query().Get("dbname")

    if dbname == "" {

        http.Error(w, "dbname parameter is required",
http.StatusBadRequest)

        return

    }



    latestMetrics, err := Server.GetLatestMetrics(dbname)

    if err != nil {

        http.Error(w, err.Error(), http.StatusInternalServerError)

        return

    }



    w.Header().Set("Content-Type", "application/json")

    _, _ = w.Write([]byte(latestMetrics))

}
```

3. GetLatestMetrics Implementation

Location: `internal/webserver/api.go`

Description: Implemented the core metrics retrieval logic.

Key Code Changes:

```
func (server *WebUIServer) GetLatestMetrics(dbname string) (string,
error) {

    rows, err := server.sinksWriter.GetLatestMetrics(dbname)

    if err != nil {

        return "", err

    }

    if rows == nil {

        return "{}", nil

    }

    defer (*rows).Close()


    metrics := make(map[string]interface{})

    var latestTime time.Time

    metricValues := make(map[string]interface{})


    for (*rows).Next() {

        var (

            time            time.Time

            tps             sql.NullFloat64

            qps             sql.NullFloat64

            avgQueryRuntime sql.NullFloat64

            blksHitRatio    sql.NullFloat64

            dbSize          sql.NullInt64

            txErrorRatio    sql.NullFloat64

            nonIdleSessions sql.NullInt64
```

```
            tempBytes       sql.NullInt64
        )

        err := (*rows).Scan(
            &time,
            &tps,
            &qps,
            &avgQueryRuntime,
            &blksHitRatio,
            &dbSize,
            &txErrorRatio,
            &nonIdleSessions,
            &tempBytes,
        )
        if err != nil {
            return "", err
        }

        latestTime = time
        metricValues["tps"] = fmt.Sprintf("%.4f", tps.Float64)
        metricValues["qps"] = fmt.Sprintf("%.4f", qps.Float64)
        metricValues["avg_query_runtime"] = fmt.Sprintf("%.4f",
avgQueryRuntime.Float64)
        metricValues["blks_hit_ratio"] = fmt.Sprintf("%.4f",
blksHitRatio.Float64)

        if dbSize.Valid {
            metricValues["db_size"] = fmt.Sprintf("%.4f MB",
float64(dbSize.Int64)/1024/1024)
```

```
        } else {
            metricValues["db_size"] = "0.0000 MB"
        }

        metricValues["tx_error_ratio"] = fmt.Sprintf("%.4f",
txErrorRatio.Float64)
        metricValues["non_idle_sessions"] = nonIdleSessions.Int64

        if tempBytes.Valid {
            metricValues["temp_bytes_written"] = fmt.Sprintf("%.4f KB",
float64(tempBytes.Int64)/1024)
        } else {
            metricValues["temp_bytes_written"] = "0.0000 KB"
        }
    }

    metrics["time"] = latestTime
    metrics["values"] = metricValues

    jsonBytes, err := json.Marshal(metrics)
    if err != nil {
        return "", err
    }
    return string(jsonBytes), nil
}
```

4. MultiWriter Implementation

Location: `internal/sinks/multiwriter.go`

Description: Added support for retrieving latest metrics through the MultiWriter interface. This implementation allows the system to query metrics from multiple writers while ensuring proper error handling and result coordination.

Key Code Changes:

```
func (mw *MultiWriter) GetLatestMetrics(dbname string) (*pgx.Rows,
error) {

    for _, w := range mw.writers {

        rows, err := w.GetLatestMetrics(dbname)

        if err != nil {

            return nil, err

        }

        if rows != nil {

            return rows, nil

        }

    }

    return nil, nil

}
```

5. SQL Query Implementation

Location: `internal/sinks/postgres.go`

Description: Implemented a complex SQL query that efficiently retrieves and calculates the latest database metrics. The query uses Common Table Expressions (CTEs) to organize the data retrieval process and calculate various metrics.

Key Code Changes:

```
func (pgw *PostgresWriter) GetLatestMetrics(dbname string) (*pgx.Rows,
error) {

    query := `

        WITH latest_metrics AS (
```

```
            SELECT
                time,
                data->>'xact_commit' as xact_commit,
                data->>'xact_rollback' as xact_rollback,
                data->>'numbackends' as numbackends,
                data->>'blks_hit' as blks_hit,
                data->>'blks_read' as blks_read,
                data->>'blks_dirtied' as blks_dirtied,
                data->>'blks_written' as blks_written,
                data->>'temp_bytes' as temp_bytes,
                LAG((data->>'temp_bytes')::int8) OVER (ORDER BY time)
as prev_temp_bytes
            FROM db_stats
            WHERE dbname = $1
            ORDER BY time DESC
            LIMIT 2
        ),
        latest_db_size AS (
            SELECT
                time,
                data->>'size_b' as size
            FROM db_size
            WHERE dbname = $1
            ORDER BY time DESC
            LIMIT 1
        ),
        latest_backends AS (
            SELECT
                time,
```

```
                (data->>'idleintransaction')::int + (data-
>>'waiting')::int + (data->>'active')::int as non_idle_sessions

            FROM backends

            WHERE dbname = $1

            ORDER BY time DESC

            LIMIT 1

        ),

        latest_statements AS (

            SELECT

                time,

                (data->>'total_time')::float8 as total_time,

                (data->>'calls')::int8 as calls,

                tag_data->>'queryid' as queryid,

                tag_data->>'query' as query,

                LAG((data->>'total_time')::float8) OVER (PARTITION BY
tag_data->>'queryid' ORDER BY time) as prev_total_time,

                LAG((data->>'calls')::int8) OVER (PARTITION BY
tag_data->>'queryid' ORDER BY time) as prev_calls,

                LAG(time) OVER (PARTITION BY tag_data->>'queryid' ORDER
BY time) as prev_time

            FROM stat_statements

            WHERE dbname = $1

            AND NOT tag_data->>'query' LIKE '%epoch_ns%'  -- exclude
pgwatch queries

            ORDER BY time DESC

        ),

        metrics_with_lag AS (

            SELECT

                time,

                xact_commit::int8,
```

```
                xact_rollback::int8,

                (xact_commit::int8 + xact_rollback::int8) as
total_xacts,

                numbackends::int8,

                blks_hit::int8,

                blks_read::int8,

                blks_dirtied::int8,

                blks_written::int8,

                temp_bytes::int8,

                prev_temp_bytes::int8,

                LAG(xact_commit::int8 + xact_rollback::int8) OVER
(ORDER BY time) as prev_total_xacts,

                LAG(time) OVER (ORDER BY time) as prev_time,

                LAG(xact_commit::int8) OVER (ORDER BY time) as
prev_xact_commit,

                LAG(xact_rollback::int8) OVER (ORDER BY time) as
prev_xact_rollback

            FROM latest_metrics

        ),

        qps_calc AS (

            SELECT

                time,

                (calls - prev_calls)::float8 / EXTRACT(EPOCH FROM (time
- prev_time)) as qps

            FROM (

                SELECT

                    time,

                    (data->>'calls')::int8 as calls,

                    LAG((data->>'calls')::int8) OVER (ORDER BY time) as
prev_calls,

                    LAG(time) OVER (ORDER BY time) as prev_time
```

```
                FROM stat_statements_calls
                WHERE dbname = $1
                ORDER BY time DESC
                LIMIT 2
            ) x
            WHERE calls >= prev_calls AND time > prev_time
            LIMIT 1
        ),
        avg_query_runtime_calc AS (
            SELECT
                avg((tt-tt_lag)::numeric / (c-c_lag)) as
avg_query_runtime
            FROM (
                SELECT
                    (data->>'total_time')::float8 as tt,
                    LAG((data->>'total_time')::float8) OVER (ORDER BY
time) as tt_lag,
                    (data->>'calls')::int8 as c,
                    LAG((data->>'calls')::int8) OVER (ORDER BY time) as
c_lag,
                    time
                FROM stat_statements_calls
                WHERE dbname = $1
                ORDER BY time DESC
                LIMIT 2
            ) x
            WHERE c > c_lag AND tt >= tt_lag AND c > 100
        ),
        latest_metrics_only AS (
```

```
            SELECT * FROM metrics_with_lag WHERE time > prev_time LIMIT
1

        )

        SELECT

            m.time,

            CASE

                WHEN m.time > m.prev_time

                THEN (m.total_xacts - m.prev_total_xacts) /
EXTRACT(EPOCH FROM (m.time - m.prev_time))

                ELSE 0

            END as tps,

            COALESCE(q.qps, 0) as qps,

            COALESCE(r.avg_query_runtime, 0) as avg_query_runtime,

            CASE

                WHEN (m.blks_hit + m.blks_read) > 0

                THEN (m.blks_hit::float / (m.blks_hit + m.blks_read)) *
100

                ELSE 0

            END as blks_hit_ratio,

            COALESCE(ds.size::int8, 0) as db_size,

            CASE

                WHEN m.time > m.prev_time AND ((m.xact_commit -
m.prev_xact_commit) + (m.xact_rollback - m.prev_xact_rollback)) > 0

                THEN ((m.xact_rollback - m.prev_xact_rollback)::numeric
* 100) /

                        ((m.xact_commit - m.prev_xact_commit) +

                        (m.xact_rollback - m.prev_xact_rollback))

                ELSE 0

            END as tx_error_ratio,

            COALESCE(b.non_idle_sessions, 0) as non_idle_sessions,
```

```
            CASE
                WHEN m.temp_bytes >= m.prev_temp_bytes
                THEN m.temp_bytes - m.prev_temp_bytes
                ELSE 0
            END as temp_bytes_written
        FROM latest_metrics_only m
        LEFT JOIN latest_db_size ds ON true
        LEFT JOIN qps_calc q ON true
        LEFT JOIN avg_query_runtime_calc r ON true
        LEFT JOIN latest_backends b ON true;
    `


    rows, err := pgw.sinkDb.Query(context.Background(), query, dbname)
    if err != nil {
        return nil, fmt.Errorf("failed to query metrics: %v", err)
    }
    return &rows, nil
}
```

6. Stub Implementations

Description: Added stub implementations for writers that don't support direct metric querying to maintain interface compatibility.

Location: `internal/sinks/rpc.go`

```
func (rw *RPCWriter) GetLatestMetrics(dbname string) (*pgx.Rows, error)
{
    // RPC writer doesn't support querying metrics, return nil
    return nil, nil
```

```
}
```

Location: `internal/sinks/json.go`

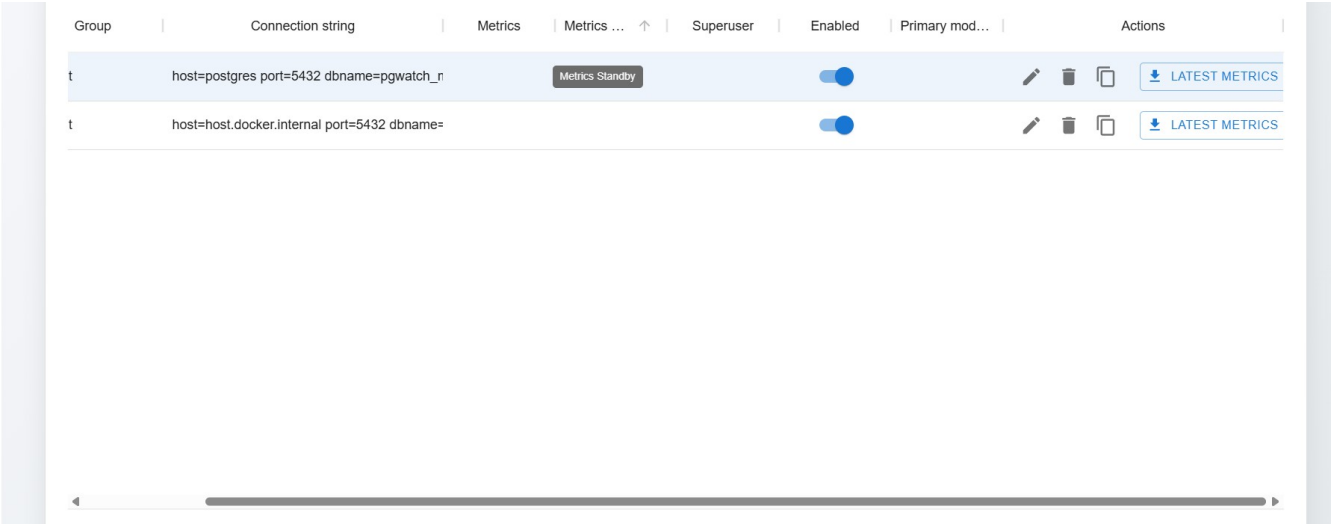
```
func (jw *JSONWriter) GetLatestMetrics(dbname string) (*pgx.Rows, error)
{

    // JSON writer doesn't support querying metrics, return nil

    return nil, nil

}
```

Location: `internal/sinks/prometheus.go`

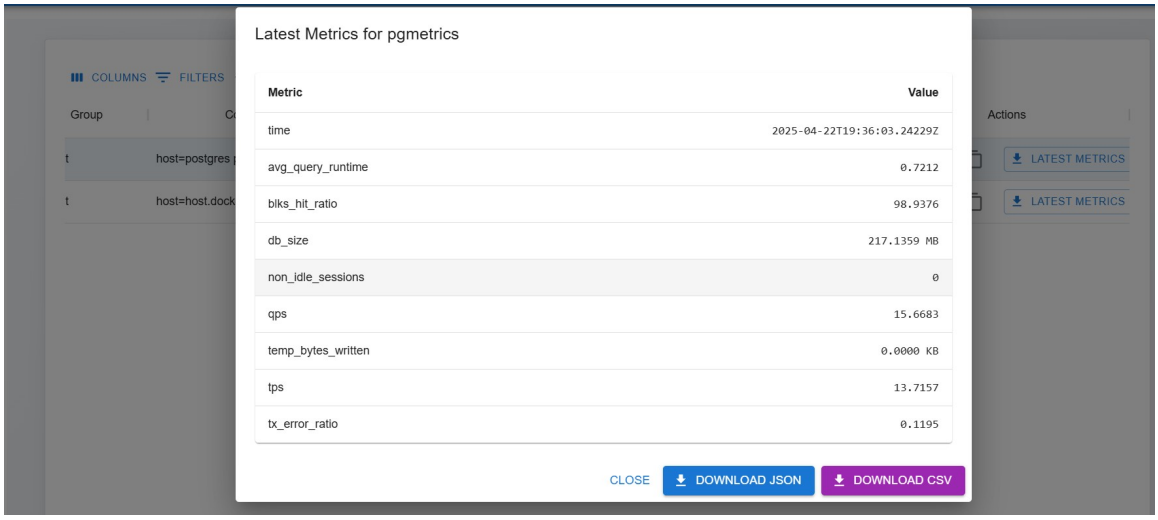
```
func (pw *PrometheusWriter) GetLatestMetrics(dbname string) (*pgx.Rows,
error) {

    // Prometheus writer doesn't support querying metrics, return nil

    return nil, nil

}
```

Final Output

The button:



On Clicking Button:



Downloaded csv:

	A	B	C	D
1	Metric	Value		
2	time	2025-04-22T19:36:03.24229Z		
3	values.avg_	0.7212		
4	values.blks_	98.9376		
5	values.db_s	217.1359 MB		
6	values.non_	0		
7	values.qps	15.6683		
8	values.temp	0.0000 KB		
9	values.tps	13.7157		
10	values.tx_er	0.1195		

Downloaded json:


```
{
  "time": "2025-04-22T19:36:03.24229Z",
  "values": {
    "avg_query_runtime": "0.7212",
    "blks_hit_ratio": "98.9376",
    "db_size": "217.1359 MB",
    "non_idle_sessions": 0,
    "qps": "15.6683",
    "temp_bytes_written": "0.0000 KB",
    "tps": "13.7157",
    "tx_error_ratio": "0.1195"
  }
}
```

:

COMPARE METRICS IMPLEMENTATION

Frontend Changes

1. Package.json Updates

Location: `internal/webui/package.json`

Description: Added new dependencies required for the Compare Metrics feature.

```
{
  "dependencies": {
    // ... existing dependencies ...
    "recharts": "^2.10.0"
  }
}
```

2. CompareMetricsButton Component

Location: `internal/webui/src/components/CompareMetricsButton/CompareMetricsButton.tsx`

Description: A new React component that enables comparison of metrics between two databases.

Key Features:

- Database selection dropdowns with validation
- Table and chart views for comparison

- Percentage difference calculations

- Color-coded indicators for metric changes

- Responsive grid layout

- Batch metrics fetching for efficiency

Key Code Changes:

```
// New component implementation

export const CompareMetricsButton: React.FC<CompareMetricsButtonProps>
= ({ databases }) => {

  const [open, setOpen] = useState(false);

  const [db1, setDb1] = useState<string>('');

  const [db2, setDb2] = useState<string>('');

  const [metricsCache, setMetricsCache] = useState<MetricsCache>({});

  const [loading, setLoading] = useState(false);

  const [error, setError] = useState<string>('');

  const [showComparison, setShowComparison] = useState(false);

  const [tabValue, setTabValue] = useState(0);


  useEffect(() => {

    if (open && databases.length > 0) {

      fetchAllMetrics();

    }

  }, [open, databases]);


  const fetchAllMetrics = async () => {

    if (databases.length === 0) return;


    setLoading(true);
```

```
setError('');

try {
  const token = getToken();
  const response = await fetch('/batch-latest-metrics', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Token': token || '',
    },
    body: JSON.stringify(databases),
  });

  if (!response.ok) {
    const errorText = await response.text();
    throw new Error(errorText || 'Failed to fetch metrics');
  }

  const data: MetricsResponse = await response.json();
  const newCache: MetricsCache = {};

  // Process results and errors
  Object.entries(data.results).forEach(([dbname, metrics]) => {
    newCache[dbname] = metrics;
  });

  setMetricsCache(newCache);
} catch (err) {
```

```
      setError(err instanceof Error ? err.message : 'Failed to fetch
metrics');

    } finally {

      setLoading(false);

    }

  };

  // ... rest of the component

};
```

3. SourcesGrid Update

Location: `internal/webui/src/pages/SourcesPage/components/SourcesGrid/SourcesGrid.tsx`

Description: Added the CompareMetricsButton to the SourcesGrid toolbar.

Key Code Changes:

```
// In the SourcesGrid component

const CustomToolbar = () => (

  <Box sx={{ display: 'flex', alignItems: 'center' }}>

    <SourcesGridToolbar />

    <CompareMetricsButton databases={databases} />

  </Box>

);
```

Backend Changes

1. WebServer Route Registration

Location: `internal/webserver/webserver.go`

Description: Added the route registration for the batch-latest-metrics endpoint.

```
mux.Handle("/batch-latest-metrics",
NewEnsureAuth(s.handleBatchLatestMetrics))
```

2. API Endpoint Handler

Location: `internal/webserver/metric.go`

Description: Added a new HTTP handler for the `/batch-latest-metrics` endpoint.

Key Code Changes:

```
func (server *WebUIServer) handleBatchLatestMetrics(w
http.ResponseWriter, r *http.Request) {

    if r.Method != http.MethodPost {

        w.Header().Set("Allow", "POST")

        http.Error(w, "method not allowed", http.StatusMethodNotAllowed)

        return

    }


    w.Header().Set("Content-Type", "application/json")

    w.Header().Set("Access-Control-Allow-Headers", "Content-Type,
Token")


    var dbnames []string

    if err := json.NewDecoder(r.Body).Decode(&dbnames); err != nil {

        server.l.WithError(err).Error("Failed to decode request body")

        http.Error(w, "Invalid request body", http.StatusBadRequest)

        return

    }


    if len(dbnames) == 0 {
```

```
        http.Error(w, "No database names provided",
http.StatusBadRequest)

        return

    }



    response, err := server.GetBatchLatestMetrics(dbnames)

    if err != nil {

        server.l.WithError(err).Error("Failed to fetch batch metrics")

        http.Error(w, fmt.Sprintf("Failed to fetch metrics: %v", err),
http.StatusInternalServerError)

        return

    }



    if err := json.NewEncoder(w).Encode(response); err != nil {

        server.l.WithError(err).Error("Failed to encode response")

        http.Error(w, "Failed to encode response",
http.StatusInternalServerError)

        return

    }

}
```

3. GetBatchLatestMetrics Implementation

Location: `internal/webserver/api.go`

Description: Implemented the batch metrics retrieval logic.

Key Code Changes:

```
type BatchMetricsResponse struct {

    Results map[string]*MetricsResponse `json:"results"`

    Errors  map[string]string          `json:"errors"`
```

```
}


func (server *WebUIServer) GetBatchLatestMetrics(dbnames []string)
(*BatchMetricsResponse, error) {

    response := &BatchMetricsResponse{

        Results: make(map[string]*MetricsResponse),

        Errors:  make(map[string]string),

    }


    for _, dbname := range dbnames {

        metricsStr, err := server.GetLatestMetrics(dbname)

        if err != nil {

            response.Errors[dbname] = err.Error()

            continue

        }


        var metricsData MetricsResponse

        if err := json.Unmarshal([]byte(metricsStr), &metricsData);
err != nil {

            response.Errors[dbname] = fmt.Sprintf("Failed to parse
metrics data: %v", err)

            continue

        }


        response.Results[dbname] = &metricsData

    }


    return response, nil

}
```

4. MultiWriter Implementation

Location: `internal/sinks/multiwriter.go`

Description: The MultiWriter implementation is shared with the Latest Metrics feature and is used to fetch metrics for multiple databases.

Key Code Changes:

```
func (mw *MultiWriter) GetLatestMetrics(dbname string) (*pgx.Rows,
error) {

    for _, w := range mw.writers {

        rows, err := w.GetLatestMetrics(dbname)

        if err != nil {

            return nil, err

        }

        if rows != nil {

            return rows, nil

        }

    }

    return nil, nil

}
```

5. SQL Query Implementation

Location: `internal/sinks/postgres.go`

Description: The SQL query implementation is shared with the Latest Metrics feature and is used to fetch metrics for comparison.

Key Code Changes:

```
func (pgw *PostgresWriter) GetLatestMetrics(dbname string) (*pgx.Rows,
error) {
	query := `
		WITH latest_metrics AS (
			SELECT
				time,
				data->>'xact_commit' as xact_commit,
				data->>'xact_rollback' as xact_rollback,
				data->>'numbackends' as numbackends,
				data->>'blks_hit' as blks_hit,
				data->>'blks_read' as blks_read,
				data->>'blks_dirtied' as blks_dirtied,
				data->>'blks_written' as blks_written,
				data->>'temp_bytes' as temp_bytes
			FROM db_stats
			WHERE dbname = $1
			ORDER BY time DESC
			LIMIT 2
		),
		// ... rest of the query as shown in Latest Metrics section
	`

	rows, err := pgw.sinkDb.Query(context.Background(), query, dbname)
	if err != nil {
		return nil, fmt.Errorf("failed to query metrics: %v", err)
	}
	return &rows, nil
}
```

6. Stub Implementations

Description: The stub implementations are shared with the Latest Metrics feature to maintain interface compatibility.

Location: `internal/sinks/rpc.go`

```
func (rw *RPCWriter) GetLatestMetrics(dbname string) (*pgx.Rows, error)
{

    // RPC writer doesn't support querying metrics, return nil

    return nil, nil

}
```

Location: `internal/sinks/json.go`

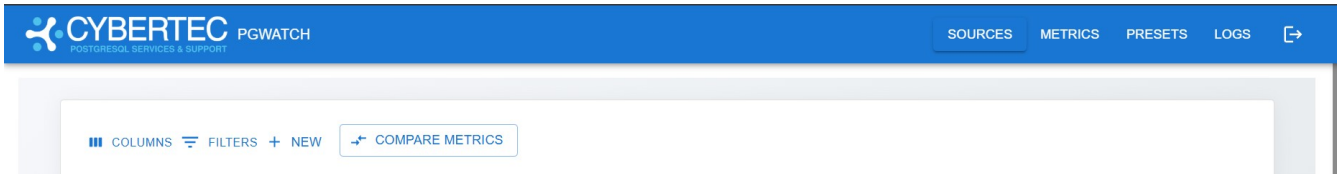
```
func (jw *JSONWriter) GetLatestMetrics(dbname string) (*pgx.Rows, error)
{

    // JSON writer doesn't support querying metrics, return nil

    return nil, nil

}
```

Location: `internal/sinks/prometheus.go`

```
func (pw *PrometheusWriter) GetLatestMetrics(dbname string) (*pgx.Rows,
error) {

    // Prometheus writer doesn't support querying metrics, return nil

    return nil, nil

}
```

Final Output

The button:



On Clicking:

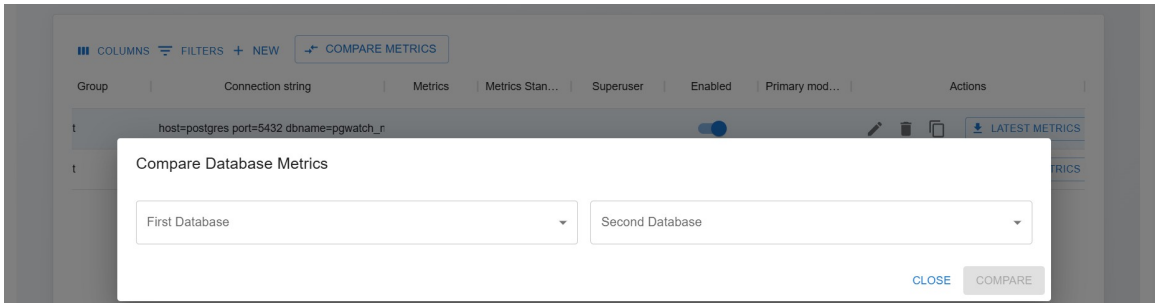


Table:

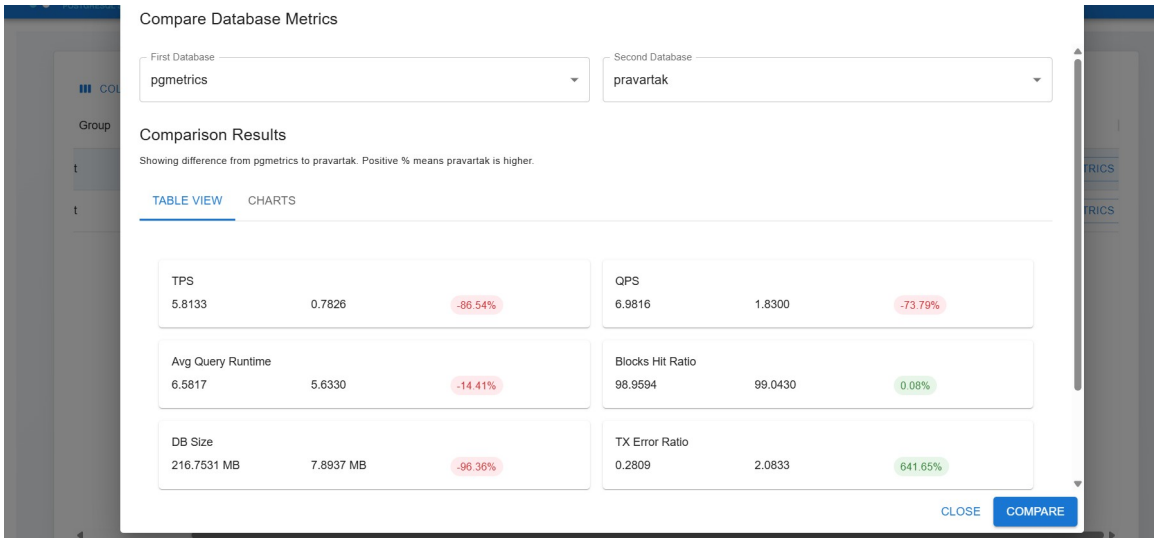
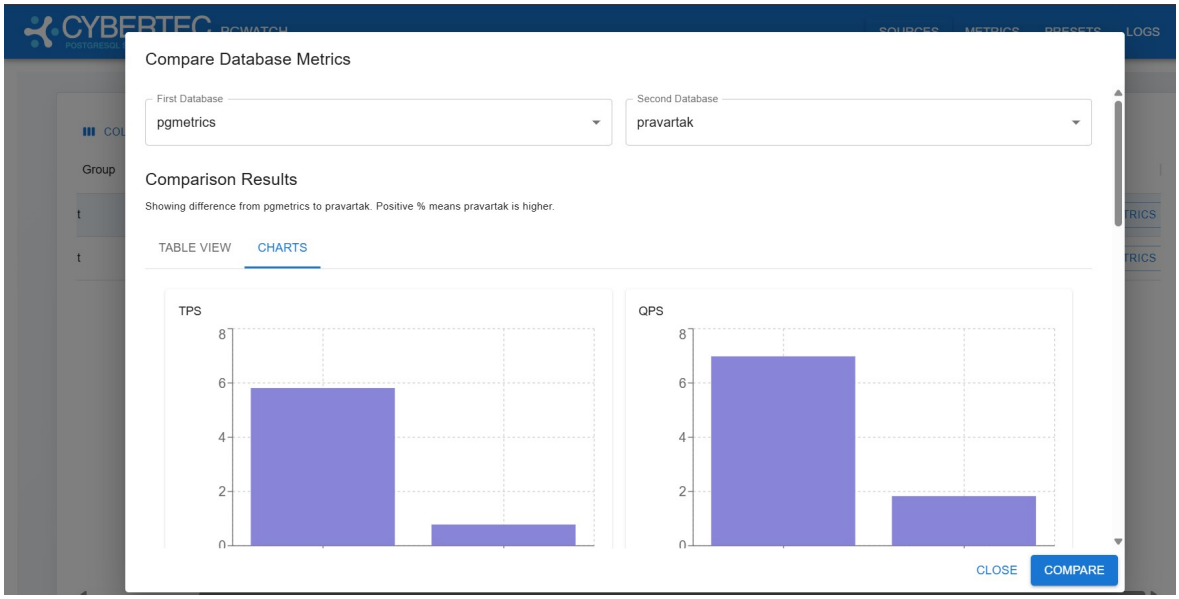


Chart:



Conclusion

The Latest Metrics and Compare Metrics features successfully enhance PGWatch2 by providing users with immediate access to current database performance metrics and the ability to compare metrics between databases. The implementation follows best practices in both frontend and backend development:

Frontend:

- Clean, reusable component design

- Intuitive user interface with loading states and error handling

- Seamless integration with existing grid actions

- Consistent styling with the application's design system

- Multiple view options (table and charts) for metric comparison

- Export functionality for metrics data

Backend:

- Optimized SQL queries using CTEs for efficient data retrieval

- Robust error handling and null value management

- Secure API endpoints with JWT authentication

- Extensible interface design supporting multiple writer types

- Batch processing for efficient multi-database metric retrieval

The features deliver immediate value to users while maintaining the system's performance and security standards.