ZZedc System Administrator Guide

Contents

ZZedc System Administrator Guide

For Biostatistics Support Teams and IT Administrators

Document Information

- **Version**: 1.0.0
- **Date**: December 2025
- **Audience**: Biostatisticians, System Administrators, DevOps Engineers
- **Prerequisites**: Linux/Unix administration, R environment management, database concepts

---

Table of Contents

---

1. Server Maintenance Procedures

1.1 Backup Procedures

Automated Daily Backups Configure automated backups using the built-in backup system:

```
#!/bin/bash
# /opt/zzedc/scripts/daily_backup.sh

BACKUP_DIR="/var/backups/zzedc"
DATE=$(date +%Y%m%d_%H%M%S)
DB_PATH="/var/data/zzedc/study.db"
RETENTION_DAYS=30

# Create backup directory
mkdir -p "${BACKUP_DIR}/daily"

# Backup database with integrity check
Rscript -e "
library(zzedc)
backup_result <- perform_database_backup(
  db_path = '${DB_PATH}',
  backup_dir = '${BACKUP_DIR}/daily',
  backup_name = 'zzedc_backup_${DATE}',
  verify = TRUE
)
if (!backup_result\$success) stop('Backup failed: ', backup_result\$error)
"

# Backup configuration files
tar -czf "${BACKUP_DIR}/daily/config_${DATE}.tar.gz" \
  /opt/zzedc/config.yml \
```

```
  /opt/zzedc/.Renviron

# Remove backups older than retention period
find "${BACKUP_DIR}/daily" -type f -mtime +${RETENTION_DAYS} -delete

# Log completion
echo "$(date): Daily backup completed" >> /var/log/zzedc/backup.log
```

Cron Schedule Configuration

```
# /etc/cron.d/zzedc-backups

# Daily backup at 2:00 AM
0 2 * * * zzedc /opt/zzedc/scripts/daily_backup.sh

# Weekly full backup on Sunday at 3:00 AM
0 3 * * 0 zzedc /opt/zzedc/scripts/weekly_backup.sh

# Monthly archive on 1st at 4:00 AM
0 4 1 * * zzedc /opt/zzedc/scripts/monthly_archive.sh
```

Backup Verification Procedure Run weekly verification of backup integrity:

```
# Verify backup can be restored
verify_backup_integrity <- function(backup_path) {
  temp_db <- tempfile(fileext = ".db")
  on.exit(unlink(temp_db))

  # Attempt restoration to temporary location
  restore_result <- restore_database_backup(
    backup_path = backup_path,
    restore_path = temp_db,
    verify_only = TRUE
  )

  if (!restore_result$success) {
    warning("Backup verification failed: ", restore_result$error)
    return(FALSE)
  }

  # Verify table counts
  con <- DBI::dbConnect(RSQLite::SQLite(), temp_db)
  on.exit(DBI::dbDisconnect(con), add = TRUE)

  tables <- DBI::dbListTables(con)
  expected_tables <- c("subjects", "visits", "form_data", "audit_trail",
                       "users", "data_dictionary")

  missing <- setdiff(expected_tables, tables)
  if (length(missing) > 0) {
    warning("Missing tables in backup: ", paste(missing, collapse = ", "))
    return(FALSE)
  }

  message("Backup verification successful")
```

```
  TRUE
}
```

1.2 Log Rotation

Application Log Configuration Create `/etc/logrotate.d/zzedc`:

```
/var/log/zzedc/*.log {
    daily
    rotate 14
    compress
    delaycompress
    missingok
    notifempty
    create 0640 zzedc zzedc
    sharedscripts
    postrotate
        systemctl reload zzedc 2>/dev/null || true
    endscript
}

/var/log/zzedc/audit/*.log {
    weekly
    rotate 52
    compress
    delaycompress
    missingok
    notifempty
    create 0640 zzedc zzedc
    # Audit logs require longer retention for compliance
}
```

Log Categories and Locations

Log Type	Location	Retention	Purpose
Application	/var/log/zzedc/app.log	14 days	General application events
Error	/var/log/zzedc/error.log	30 days	Error tracking
Audit	/var/log/zzedc/audit/audit.log	7 years	Regulatory compliance
Access	/var/log/zzedc/access.log	90 days	User access patterns
Security	/var/log/zzedc/security.log	1 year	Security events

1.3 System Updates and Patches

R Package Updates

```
# Monthly package update procedure
update_zzedc_packages <- function(dry_run = TRUE) {
  # Check for available updates
  old_packages <- old.packages()

  if (is.null(old_packages)) {
    message("All packages are up to date")
```

```
    return(invisible(NULL))
  }

  # Filter for ZZedc-related packages
  zzedc_deps <- c("shiny", "bslib", "RSQLite", "pool", "DT",
                  "ggplot2", "dplyr", "digest", "openxlsx")

  to_update <- old_packages[rownames(old_packages) %in% zzedc_deps, ]

  if (nrow(to_update) == 0) {
    message("No ZZedc dependency updates available")
    return(invisible(NULL))
  }

  message("Updates available:")
  print(to_update[, c("Package", "Installed", "ReposVer")])

  if (!dry_run) {
    # Create restore point
    message("Creating package restore point...")
    renv::snapshot()

    # Apply updates
    update.packages(oldPkgs = to_update, ask = FALSE)
    message("Updates applied. Restart ZZedc service to activate.")
  }
}
```

System Update Procedure

```
#!/bin/bash
# /opt/zzedc/scripts/system_update.sh

# 1. Create backup before update
/opt/zzedc/scripts/daily_backup.sh

# 2. Stop ZZedc service
sudo systemctl stop zzedc

# 3. Update system packages
sudo apt update && sudo apt upgrade -y

# 4. Update R packages (in maintenance mode)
Rscript -e "source('/opt/zzedc/scripts/update_packages.R')"

# 5. Run database migrations if needed
Rscript -e "zzedc::run_migrations()"

# 6. Verify application integrity
Rscript -e "zzedc::verify_installation()"

# 7. Restart ZZedc service
sudo systemctl start zzedc
```

```
# 8. Verify service health
sleep 10
curl -s http://localhost:3838/health | grep -q "ok" || {
    echo "Health check failed, rolling back..."
    sudo systemctl stop zzedc
    /opt/zzedc/scripts/restore_backup.sh
    sudo systemctl start zzedc
}

echo "Update completed successfully"
```

---

2. Database Administration

2.1 SQLite Maintenance

Database Vacuum and Optimization Schedule weekly optimization:

```
# Database optimization procedure
optimize_database <- function(db_path, verbose = TRUE) {
  con <- DBI::dbConnect(RSQLite::SQLite(), db_path)
  on.exit(DBI::dbDisconnect(con))

  # Get initial size
  initial_size <- file.size(db_path)

  if (verbose) message("Running integrity check...")
  integrity <- DBI::dbGetQuery(con, "PRAGMA integrity_check")
  if (integrity$integrity_check != "ok") {
    stop("Database integrity check failed: ", integrity$integrity_check)
  }

  if (verbose) message("Analyzing tables...")
  DBI::dbExecute(con, "ANALYZE")

  if (verbose) message("Running VACUUM...")
  DBI::dbExecute(con, "VACUUM")

  if (verbose) message("Optimizing indexes...")
  DBI::dbExecute(con, "REINDEX")

  final_size <- file.size(db_path)
  savings <- initial_size - final_size

  if (verbose) {
    message(sprintf("Optimization complete. Space reclaimed: %.2f MB",
                    savings / 1024^2))
  }

  list(
    initial_size = initial_size,
    final_size = final_size,
    space_reclaimed = savings,
    integrity = "ok"
```

```
  )
}
```

Integrity Check Procedures

```
# Comprehensive database health check
check_database_health <- function(db_path) {
  con <- DBI::dbConnect(RSQLite::SQLite(), db_path)
  on.exit(DBI::dbDisconnect(con))

  results <- list()

  # 1. Integrity check
  results$integrity <- DBI::dbGetQuery(con, "PRAGMA integrity_check")$integrity_check


  # 2. Foreign key check
  results$foreign_keys <- DBI::dbGetQuery(con, "PRAGMA foreign_key_check")
  results$fk_violations <- nrow(results$foreign_keys)

  # 3. Table statistics
  tables <- DBI::dbListTables(con)
  results$table_stats <- lapply(tables, function(tbl) {
    count <- DBI::dbGetQuery(con, sprintf("SELECT COUNT(*) as n FROM %s", tbl))$n
    list(table = tbl, row_count = count)
  })

  # 4. Index health
  results$indexes <- DBI::dbGetQuery(con,
    "SELECT name, tbl_name FROM sqlite_master WHERE type='index'")

  # 5. Database size
  results$size_bytes <- file.size(db_path)
  results$size_mb <- results$size_bytes / 1024^2

  # 6. Page statistics
  results$page_count <- DBI::dbGetQuery(con, "PRAGMA page_count")$page_count
  results$page_size <- DBI::dbGetQuery(con, "PRAGMA page_size")$page_size
  results$freelist_count <- DBI::dbGetQuery(con, "PRAGMA freelist_count")$freelist_count

  # Calculate fragmentation
  used_pages <- results$page_count - results$freelist_count
  results$fragmentation_pct <- (results$freelist_count / results$page_count) * 100

  # Generate health score
  results$health_score <- calculate_health_score(results)

  class(results) <- c("zzedc_db_health", "list")
  results
}

calculate_health_score <- function(results) {
  score <- 100
```

```
  if (results$integrity != "ok") score <- score - 50
  if (results$fk_violations > 0) score <- score - 20
  if (results$fragmentation_pct > 20) score <- score - 10
  if (results$fragmentation_pct > 40) score <- score - 10

  max(0, score)
}
```

2.2 Performance Tuning

SQLite Pragma Configuration

```
# Apply performance pragmas to database connection
configure_db_performance <- function(con) {
  # Write-ahead logging for better concurrency
  DBI::dbExecute(con, "PRAGMA journal_mode = WAL")

  # Increase cache size (negative = KB, positive = pages)
  DBI::dbExecute(con, "PRAGMA cache_size = -64000")  # 64 MB cache

  # Synchronous mode: NORMAL balances safety and speed
  DBI::dbExecute(con, "PRAGMA synchronous = NORMAL")

  # Memory-mapped I/O (256 MB)
  DBI::dbExecute(con, "PRAGMA mmap_size = 268435456")

  # Temporary storage in memory
  DBI::dbExecute(con, "PRAGMA temp_store = MEMORY")

  # Enable foreign keys

  DBI::dbExecute(con, "PRAGMA foreign_keys = ON")

  invisible(con)
}
```

Query Performance Analysis

```
# Analyze slow queries
analyze_query_performance <- function(con, query, explain = TRUE) {
  if (explain) {
    # Get query plan
    plan <- DBI::dbGetQuery(con, paste("EXPLAIN QUERY PLAN", query))
    message("Query Plan:")
    print(plan)
  }

  # Time execution
  start <- Sys.time()
  result <- DBI::dbGetQuery(con, query)
  elapsed <- Sys.time() - start

  list(
    rows_returned = nrow(result),
    execution_time = elapsed,
```

```
    query_plan = if (explain) plan else NULL
  )
}
```

2.3 Connection Pool Management

```
# Connection pool configuration
create_db_pool <- function(db_path, min_size = 1, max_size = 5) {
  pool::dbPool(
    drv = RSQLite::SQLite(),
    dbname = db_path,
    minSize = min_size,
    maxSize = max_size,
    idleTimeout = 60000,  # 60 seconds
    validationInterval = 10000,  # 10 seconds
    onCreate = function(con) {
      configure_db_performance(con)
    }
  )
}

# Monitor pool health
monitor_pool_health <- function(pool) {
  list(
    size = pool::poolSize(pool),
    idle = pool::poolIdle(pool),
    active = pool::poolSize(pool) - pool::poolIdle(pool),
    valid = pool::dbIsValid(pool)
  )
}
```

---

3. Troubleshooting Guide

3.1 Common Issues and Solutions

Issue: Application Fails to Start **Symptoms**: Service fails to start, port already in use, package loading errors.

Diagnostic Steps:

```
# Check service status
sudo systemctl status zzedc

# View recent logs
sudo journalctl -u zzedc -n 50 --no-pager

# Check port availability
sudo lsof -i :3838

# Verify R environment
Rscript -e "sessionInfo()"
```

Solutions:

```
# Kill orphaned processes
```

```
sudo fuser -k 3838/tcp

# Reset application state
sudo systemctl stop zzedc
rm -f /var/run/zzedc/*.pid
sudo systemctl start zzedc

# Reinstall packages if corrupted
Rscript -e "remove.packages('shiny'); install.packages('shiny')"
```

Issue: Database Connection Errors **Symptoms**: "Database is locked", connection timeouts, read-only errors.

Diagnostic Steps:

```
# Check database file permissions
file.info("/var/data/zzedc/study.db")

# Check for lock files
list.files("/var/data/zzedc", pattern = "\\.db-", full.names = TRUE)

# Test basic connectivity
con <- DBI::dbConnect(RSQLite::SQLite(), "/var/data/zzedc/study.db")
DBI::dbGetQuery(con, "SELECT 1")
DBI::dbDisconnect(con)
```

Solutions:

```
# Fix permissions
sudo chown zzedc:zzedc /var/data/zzedc/study.db
chmod 660 /var/data/zzedc/study.db

# Remove stale locks (CAUTION: ensure no active connections)
sudo systemctl stop zzedc
rm -f /var/data/zzedc/study.db-shm /var/data/zzedc/study.db-wal
sudo systemctl start zzedc

# Recover from WAL
Rscript -e "
con <- DBI::dbConnect(RSQLite::SQLite(), '/var/data/zzedc/study.db')
DBI::dbExecute(con, 'PRAGMA wal_checkpoint(TRUNCATE)')
DBI::dbDisconnect(con)
"
```

Issue: Memory Exhaustion **Symptoms**: Application crashes, R session aborts, out of memory errors.

Diagnostic Steps:

```
# Check system memory
free -h

# Check R memory usage
Rscript -e "gc(); print(gc())"

# Monitor real-time memory
watch -n 1 'ps aux | grep -E "R|zzedc" | grep -v grep'
```

Solutions:

```
# Configure R memory limits in .Renviron
# R_MAX_VSIZE=8Gb
# R_MAX_MEM_SIZE=8Gb

# Implement data pagination for large queries
fetch_paginated <- function(con, table, page_size = 1000) {
  total <- DBI::dbGetQuery(con, sprintf("SELECT COUNT(*) FROM %s", table))[[1]]
  pages <- ceiling(total / page_size)

  lapply(seq_len(pages), function(page) {
    offset <- (page - 1) * page_size
    DBI::dbGetQuery(con, sprintf(
      "SELECT * FROM %s LIMIT %d OFFSET %d",
      table, page_size, offset
    ))
  })
}
```

Issue: Slow Performance **Symptoms**: Page load times > 5 seconds, report generation timeouts.

Diagnostic Steps:

```
# Profile application
profvis::profvis({
  # Run problematic operation
})

# Check database query times
system.time({
  DBI::dbGetQuery(con, "SELECT * FROM form_data WHERE visit_id = 1")
})

# Monitor connection pool
monitor_pool_health(db_pool)
```

Solutions:

```
# Add missing indexes
DBI::dbExecute(con, "CREATE INDEX IF NOT EXISTS idx_form_data_visit
                     ON form_data(visit_id)")
DBI::dbExecute(con, "CREATE INDEX IF NOT EXISTS idx_form_data_subject
                     ON form_data(subject_id)")
DBI::dbExecute(con, "CREATE INDEX IF NOT EXISTS idx_audit_timestamp
                     ON audit_trail(timestamp)")

# Enable query caching
options(zzedc.query_cache_size = 100)
options(zzedc.query_cache_ttl = 300)  # 5 minutes
```

3.2 Diagnostic Commands Reference

```
# System diagnostics
/opt/zzedc/scripts/diagnostics.sh

# Database diagnostics
```

```
Rscript -e "zzedc::diagnose_database('/var/data/zzedc/study.db')"

# Connection diagnostics
Rscript -e "zzedc::test_connections()"

# Full health check
Rscript -e "zzedc::full_health_check()"
```

3.3 Recovery Procedures

Database Recovery from Corruption

```
# Attempt database recovery
recover_database <- function(db_path) {
  backup_path <- paste0(db_path, ".corrupt.", format(Sys.time(), "%Y%m%d%H%M%S"))

  # Preserve corrupted database
  file.copy(db_path, backup_path)
  message("Corrupted database preserved at: ", backup_path)

  # Attempt recovery using .dump
  recovery_path <- paste0(db_path, ".recovered")

  system2("sqlite3", args = c(
    db_path,
    sprintf(".output %s.sql", recovery_path),
    ".dump"
  ))

  # Recreate from dump
  if (file.exists(paste0(recovery_path, ".sql"))) {
    unlink(recovery_path)
    system2("sqlite3", args = c(
      recovery_path,
      sprintf(".read %s.sql", recovery_path)
    ))

    # Verify recovery
    con <- DBI::dbConnect(RSQLite::SQLite(), recovery_path)
    integrity <- DBI::dbGetQuery(con, "PRAGMA integrity_check")
    DBI::dbDisconnect(con)

    if (integrity$integrity_check == "ok") {
      file.rename(recovery_path, db_path)
      message("Database recovered successfully")
      return(TRUE)
    }
  }

  warning("Automatic recovery failed. Manual intervention required.")
  FALSE
}
```

Service Recovery Procedure

```bash
#!/bin/bash
# /opt/zzedc/scripts/emergency_recovery.sh

echo "ZZedc Emergency Recovery Procedure"
echo "=================================="

# Stop all services
sudo systemctl stop zzedc

# Clear temporary files
rm -rf /tmp/zzedc_*
rm -rf /var/run/zzedc/*

# Reset log files if corrupted
if ! file /var/log/zzedc/app.log | grep -q "text"; then
    mv /var/log/zzedc/app.log /var/log/zzedc/app.log.corrupt
    touch /var/log/zzedc/app.log
    chown zzedc:zzedc /var/log/zzedc/app.log
fi

# Verify database integrity
Rscript -e "
result <- zzedc::check_database_health('/var/data/zzedc/study.db')
if (result\$health_score < 50) {
  stop('Database health critical. Restore from backup required.')
}
"

# Restart services
sudo systemctl start zzedc

# Verify recovery
sleep 5
if curl -s http://localhost:3838/health | grep -q "ok"; then
    echo "Recovery successful"
else
    echo "Recovery failed. Restore from backup."
    /opt/zzedc/scripts/restore_latest_backup.sh
fi
```

---

4. Monitoring and Alerting

4.1 Health Check Endpoints

Implementing Health Check API The application exposes health check endpoints for monitoring:

```r
# Health check endpoint handler (in server.R)
observeEvent(input$health_check, {
  health <- list(
    status = "ok",
    timestamp = Sys.time(),
    version = packageVersion("zzedc"),
    database = check_db_connection(),
    memory = check_memory_usage(),
```

```
    disk = check_disk_space(),
    sessions = get_active_session_count()
  )

  # Return appropriate status code
  if (any(sapply(health[-c(1,2,3)], function(x) x$status == "critical"))) {
    health$status <- "critical"
  } else if (any(sapply(health[-c(1,2,3)], function(x) x$status == "warning"))) {
    health$status <- "warning"
  }

  health
})
```

Health Check Script for External Monitoring

```
#!/bin/bash
# /opt/zzedc/scripts/health_check.sh

HEALTH_URL="http://localhost:3838/health"
ALERT_EMAIL="admin@example.org"

response=$(curl -s -w "\n%{http_code}" "$HEALTH_URL" 2>/dev/null)
http_code=$(echo "$response" | tail -n 1)
body=$(echo "$response" | head -n -1)

if [ "$http_code" != "200" ]; then
    echo "CRITICAL: ZZedc health check failed (HTTP $http_code)" |
        mail -s "ZZedc Alert: Health Check Failed" "$ALERT_EMAIL"
    exit 2
fi

status=$(echo "$body" | jq -r '.status')

case "$status" in
    "ok")
        echo "OK: ZZedc is healthy"
        exit 0
        ;;
    "warning")
        echo "WARNING: ZZedc has warnings"
        echo "$body" | mail -s "ZZedc Warning" "$ALERT_EMAIL"
        exit 1
        ;;
    "critical")
        echo "CRITICAL: ZZedc is in critical state"
        echo "$body" | mail -s "ZZedc Critical Alert" "$ALERT_EMAIL"
        exit 2
        ;;
esac
```

4.2 Metrics Collection

System Metrics Dashboard

```r
# Collect system metrics for monitoring
collect_system_metrics <- function() {
  metrics <- list(
    timestamp = Sys.time(),

    # Application metrics
    active_users = get_active_session_count(),
    requests_per_minute = get_request_rate(),
    avg_response_time_ms = get_avg_response_time(),

    # Database metrics
    db_connections_active = pool::poolSize(db_pool) - pool::poolIdle(db_pool),
    db_connections_idle = pool::poolIdle(db_pool),
    db_size_mb = file.size(db_path) / 1024^2,

    # System metrics
    memory_used_pct = get_memory_usage_pct(),
    disk_used_pct = get_disk_usage_pct(),
    cpu_load_avg = get_cpu_load(),

    # Data metrics
    total_subjects = get_subject_count(),
    total_records = get_record_count(),
    pending_queries = get_pending_query_count()
  )

  # Log to time series
  append_metrics_log(metrics)

  metrics
}
```

4.3 Alert Configuration

Alert Thresholds

```yaml
# /opt/zzedc/config/alerts.yml

alerts:
  memory:
    warning: 75
    critical: 90

  disk:
    warning: 80
    critical: 95

  response_time_ms:
    warning: 2000
    critical: 5000

  database_connections:
    warning: 8  # out of 10
    critical: 10
```

```
  failed_logins:
    window_minutes: 15
    warning: 5
    critical: 10

  error_rate:
    window_minutes: 5
    warning: 0.05  # 5%
    critical: 0.10  # 10%

notifications:
  email:
    enabled: true
    recipients:
      - admin@example.org
      - biostat-support@example.org

  slack:
    enabled: false
    webhook_url: ""
```

Alert Handler

```
# Process alerts based on thresholds
process_alerts <- function(metrics, config) {
  alerts <- list()

  # Memory alert
  if (metrics$memory_used_pct >= config$alerts$memory$critical) {
    alerts$memory <- list(
      level = "critical",
      message = sprintf("Memory usage critical: %.1f%%", metrics$memory_used_pct)
    )
  } else if (metrics$memory_used_pct >= config$alerts$memory$warning) {
    alerts$memory <- list(
      level = "warning",
      message = sprintf("Memory usage elevated: %.1f%%", metrics$memory_used_pct)
    )
  }

  # Disk alert
  if (metrics$disk_used_pct >= config$alerts$disk$critical) {
    alerts$disk <- list(
      level = "critical",
      message = sprintf("Disk space critical: %.1f%%", metrics$disk_used_pct)
    )
  }

  # Response time alert
  if (metrics$avg_response_time_ms >= config$alerts$response_time_ms$critical) {
    alerts$performance <- list(
      level = "critical",
      message = sprintf("Response time critical: %d ms", metrics$avg_response_time_ms)
    )
```

```
  }

  # Send notifications
  if (length(alerts) > 0) {
    send_alert_notifications(alerts, config$notifications)
  }

  alerts
}
```

4.4 Log Monitoring

Centralized Log Aggregation

```
# /etc/rsyslog.d/zzedc.conf

# Forward ZZedc logs to central syslog server
if $programname == 'zzedc' then @@logserver.example.org:514

# Local logging with structured format
template(name="zzedc-json" type="list") {
    constant(value="{")
    constant(value="\"timestamp\":\"") property(name="timereported" dateFormat="rfc3339")
    constant(value="\",\"severity\":\"") property(name="syslogseverity-text")
    constant(value="\",\"message\":\"") property(name="msg" format="json")
    constant(value="\"}\n")
}

if $programname == 'zzedc' then /var/log/zzedc/structured.log;zzedc-json
```

Log Analysis Queries

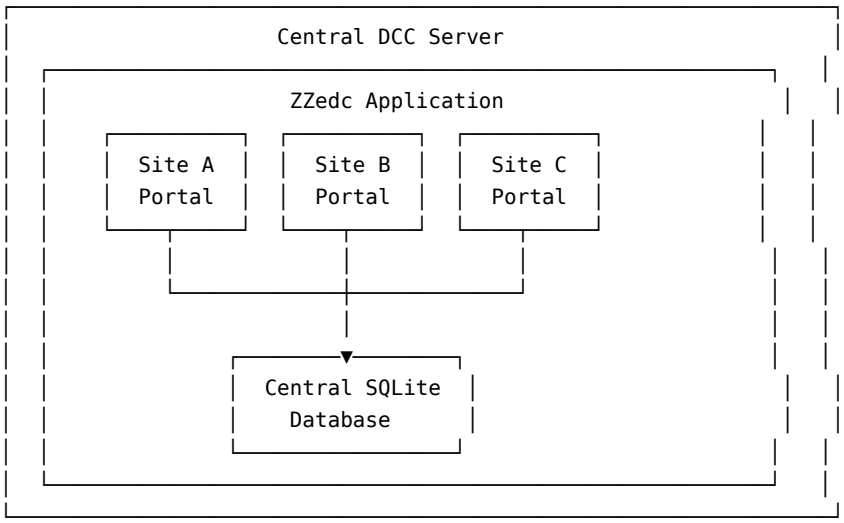
```
# Find errors in last hour
grep -E "ERROR|CRITICAL" /var/log/zzedc/app.log |
  awk -v cutoff="$(date -d '1 hour ago' '+%Y-%m-%d %H:%M')" '$1" "$2 >= cutoff'

# Count errors by type
grep "ERROR" /var/log/zzedc/app.log |
  sed 's/.*ERROR: //' |
  sort | uniq -c | sort -rn

# Find slow queries (>1 second)
grep "Query time:" /var/log/zzedc/app.log |
  awk -F': ' '$2 > 1000 {print}'

# Authentication failures
grep "Authentication failed" /var/log/zzedc/security.log |
  awk '{print $NF}' | sort | uniq -c | sort -rn
```

---

5. Multi-Site Coordination

5.1 Multi-Site Architecture

Centralized Database Model



5.2 Site Configuration

Site-Specific Settings

```yaml
# /opt/zzedc/config/sites.yml

sites:
  site_001:
    name: "University Medical Center"
    code: "UMC"
    timezone: "America/New_York"
    contact: "pi@umc.edu"
    ip_whitelist:
      - "192.168.1.0/24"
      - "10.0.0.0/8"

  site_002:
    name: "Regional Hospital"
    code: "RGH"
    timezone: "America/Chicago"
    contact: "coordinator@rgh.org"
    ip_whitelist:
      - "172.16.0.0/16"

  site_003:
    name: "Research Clinic"
    code: "RCL"
    timezone: "America/Los_Angeles"
    contact: "admin@rcl.edu"
    ip_whitelist:
      - "203.0.113.0/24"

global:
  session_timeout_minutes: 30
  max_concurrent_users_per_site: 10
  data_lock_timeout_seconds: 300
```

5.3 Site Access Control

```r
# Site-based access control
validate_site_access <- function(user_id, site_id, action, session_info) {
  user <- get_user_by_id(user_id)

  # Check site assignment
  if (!site_id %in% user$assigned_sites) {
    log_security_event("unauthorized_site_access", user_id, site_id)
    return(list(allowed = FALSE, reason = "User not assigned to site"))
  }

  # Check IP whitelist
  if (!is_ip_allowed(session_info$ip, get_site_whitelist(site_id))) {
    log_security_event("ip_not_whitelisted", user_id, site_id)
    return(list(allowed = FALSE, reason = "Access from unauthorized IP"))
  }

  # Check role permissions for action
  if (!has_permission(user$role, action)) {
    log_security_event("permission_denied", user_id, action)
    return(list(allowed = FALSE, reason = "Insufficient permissions"))
  }

  list(allowed = TRUE)
}
```

5.4 Cross-Site Data Queries

```r
# Query data across sites with appropriate filtering
query_cross_site_data <- function(user_id, query_params) {
  user <- get_user_by_id(user_id)

  # DCC staff can see all sites
  if (user$role %in% c("dcc_admin", "biostatistician")) {
    site_filter <- NULL
  } else {
    # Others see only their assigned sites
    site_filter <- user$assigned_sites
  }

  # Build query with site filter
  base_query <- "SELECT * FROM form_data fd
                 INNER JOIN subjects s ON fd.subject_id = s.id
                 WHERE 1=1"

  if (!is.null(site_filter)) {
    base_query <- paste(base_query,
      sprintf("AND s.site_id IN (%s)",
              paste(shQuote(site_filter), collapse = ",")))
  }

  # Add user-specified filters
  if (!is.null(query_params$visit_id)) {
    base_query <- paste(base_query,
```

```
      sprintf("AND fd.visit_id = %d", query_params$visit_id))
  }

  DBI::dbGetQuery(db_con, base_query)
}
```

5.5 Site Synchronization Monitoring

```
# Monitor site activity and data submission
get_site_status_summary <- function() {
  query <- "
    SELECT
      s.site_id,
      sites.name as site_name,
      COUNT(DISTINCT s.id) as enrolled_subjects,
      COUNT(DISTINCT CASE WHEN s.status = 'active' THEN s.id END) as active_subjects,
      MAX(fd.entry_timestamp) as last_data_entry,
      COUNT(CASE WHEN q.status = 'open' THEN 1 END) as open_queries
    FROM subjects s
    LEFT JOIN form_data fd ON s.id = fd.subject_id
    LEFT JOIN data_queries q ON s.id = q.subject_id
    LEFT JOIN site_config sites ON s.site_id = sites.site_id
    GROUP BY s.site_id, sites.name
    ORDER BY sites.name
  "

  status <- DBI::dbGetQuery(db_con, query)

  # Flag sites with no recent activity
  status$days_since_entry <- as.numeric(
    difftime(Sys.time(), status$last_data_entry, units = "days")
  )
  status$activity_flag <- ifelse(
    status$days_since_entry > 7, "warning",
    ifelse(status$days_since_entry > 14, "critical", "ok")
  )

  status
}
```

---

6. Security Operations

6.1 Security Monitoring

Failed Login Monitoring

```
# Monitor and respond to failed login attempts
monitor_failed_logins <- function(window_minutes = 15, threshold = 5) {
  cutoff <- Sys.time() - (window_minutes * 60)

  failed_attempts <- DBI::dbGetQuery(db_con, sprintf("
    SELECT
      username,
      ip_address,
```

```
      COUNT(*) as attempt_count,
      MAX(timestamp) as last_attempt
    FROM security_events
    WHERE event_type = 'failed_login'
      AND timestamp > '%s'
    GROUP BY username, ip_address
    HAVING COUNT(*) >= %d
  ", format(cutoff, "%Y-%m-%d %H:%M:%S"), threshold))

  if (nrow(failed_attempts) > 0) {
    for (i in seq_len(nrow(failed_attempts))) {
      row <- failed_attempts[i, ]

      # Lock account if threshold exceeded
      if (row$attempt_count >= threshold * 2) {
        lock_user_account(row$username,
          reason = sprintf("Excessive failed logins from %s", row$ip_address),
          duration_minutes = 30)
      }

      # Log security alert
      log_security_event("brute_force_detected", list(
        username = row$username,
        ip_address = row$ip_address,
        attempts = row$attempt_count
      ))
    }
  }

  failed_attempts
}
```

6.2 Session Management

```
# Session security configuration
session_security_config <- list(
  # Session timeout (30 minutes of inactivity)
  timeout_minutes = 30,

  # Absolute session lifetime (8 hours)
  max_lifetime_hours = 8,

  # Concurrent session limit per user
  max_concurrent_sessions = 2,

  # Session regeneration interval
  regenerate_id_minutes = 15,

  # Require re-authentication for sensitive operations
  reauth_timeout_minutes = 5
)

# Enforce session limits
enforce_session_limits <- function(user_id) {
```

```
  active_sessions <- get_user_active_sessions(user_id)

  if (length(active_sessions) >= session_security_config$max_concurrent_sessions) {
    # Terminate oldest session
    oldest <- active_sessions[order(active_sessions$created_at)][1]
    terminate_session(oldest$session_id,
      reason = "Concurrent session limit exceeded")
  }
}
```

6.3 Audit Log Protection

```
# Ensure audit log integrity
verify_audit_integrity <- function(start_date = NULL, end_date = NULL) {
  where_clause <- "WHERE 1=1"

  if (!is.null(start_date)) {
    where_clause <- paste(where_clause,
      sprintf("AND timestamp >= '%s'", start_date))
  }
  if (!is.null(end_date)) {
    where_clause <- paste(where_clause,
      sprintf("AND timestamp <= '%s'", end_date))
  }

  # Verify hash chain integrity
  audit_records <- DBI::dbGetQuery(db_con, sprintf("
    SELECT id, timestamp, event_type, details, previous_hash, record_hash
    FROM audit_trail
    %s
    ORDER BY id
  ", where_clause))

  if (nrow(audit_records) == 0) {
    return(list(valid = TRUE, message = "No records in range"))
  }

  # Verify each record's hash
  broken_chain <- NULL
  for (i in seq_len(nrow(audit_records))) {
    record <- audit_records[i, ]

    expected_hash <- compute_audit_hash(
      record$timestamp,
      record$event_type,
      record$details,
      record$previous_hash
    )

    if (expected_hash != record$record_hash) {
      broken_chain <- c(broken_chain, record$id)
    }

    # Verify chain link
```

```
    if (i > 1 && record$previous_hash != audit_records$record_hash[i - 1]) {
      broken_chain <- c(broken_chain, record$id)
    }
  }

  if (length(broken_chain) > 0) {
    log_security_event("audit_integrity_violation", list(
      affected_records = broken_chain
    ))

    return(list(
      valid = FALSE,
      message = "Audit chain integrity violation detected",
      affected_records = broken_chain
    ))
  }

  list(valid = TRUE, message = "Audit integrity verified", records_checked = nrow(audit_records))
}
```

---

7. Upgrade and Patch Management

7.1 Version Upgrade Procedure

Pre-Upgrade Checklist

```
## Pre-Upgrade Checklist

- [ ] Review release notes for breaking changes
- [ ] Verify backup completed within last 24 hours
- [ ] Test upgrade in staging environment
- [ ] Notify users of maintenance window
- [ ] Confirm rollback procedure is ready
- [ ] Document current version and configuration
- [ ] Verify disk space for upgrade (>= 2GB free)
- [ ] Check R package compatibility matrix
```

Upgrade Script

```
#!/bin/bash
# /opt/zzedc/scripts/upgrade.sh

set -e

NEW_VERSION=$1
BACKUP_DIR="/var/backups/zzedc/pre-upgrade"
INSTALL_DIR="/opt/zzedc"

if [ -z "$NEW_VERSION" ]; then
    echo "Usage: $0 <version>"
    exit 1
fi

echo "Upgrading ZZedc to version $NEW_VERSION"
```

```
# 1. Create pre-upgrade backup
echo "Creating pre-upgrade backup..."
mkdir -p "$BACKUP_DIR"
/opt/zzedc/scripts/daily_backup.sh
cp -r "$INSTALL_DIR" "$BACKUP_DIR/zzedc_$(date +%Y%m%d)"

# 2. Stop service
echo "Stopping ZZedc service..."
sudo systemctl stop zzedc

# 3. Download new version
echo "Downloading version $NEW_VERSION..."
cd /tmp
wget "https://github.com/org/zzedc/archive/v${NEW_VERSION}.tar.gz"
tar -xzf "v${NEW_VERSION}.tar.gz"

# 4. Install new version
echo "Installing new version..."
cp -r "zzedc-${NEW_VERSION}/"* "$INSTALL_DIR/"

# 5. Update R packages
echo "Updating R dependencies..."
cd "$INSTALL_DIR"
Rscript -e "renv::restore()"

# 6. Run database migrations
echo "Running database migrations..."
Rscript -e "zzedc::run_migrations()"

# 7. Verify installation
echo "Verifying installation..."
Rscript -e "zzedc::verify_installation()"

# 8. Start service
echo "Starting ZZedc service..."
sudo systemctl start zzedc

# 9. Health check
sleep 10
if curl -s http://localhost:3838/health | grep -q "ok"; then
    echo "Upgrade to version $NEW_VERSION completed successfully"
else
    echo "Health check failed. Initiating rollback..."
    /opt/zzedc/scripts/rollback.sh
    exit 1
fi
```

7.2 Rollback Procedure

```
#!/bin/bash
# /opt/zzedc/scripts/rollback.sh

BACKUP_DIR="/var/backups/zzedc/pre-upgrade"
```

```
INSTALL_DIR="/opt/zzedc"

# Find most recent backup
LATEST_BACKUP=$(ls -td "$BACKUP_DIR"/zzedc_* | head -1)

if [ -z "$LATEST_BACKUP" ]; then
    echo "No backup found for rollback"
    exit 1
fi

echo "Rolling back to: $LATEST_BACKUP"

# Stop service
sudo systemctl stop zzedc

# Restore application
rm -rf "$INSTALL_DIR"
cp -r "$LATEST_BACKUP" "$INSTALL_DIR"

# Restore database
LATEST_DB_BACKUP=$(ls -t "$BACKUP_DIR"/*.db 2>/dev/null | head -1)
if [ -n "$LATEST_DB_BACKUP" ]; then
    cp "$LATEST_DB_BACKUP" /var/data/zzedc/study.db
fi

# Start service
sudo systemctl start zzedc

echo "Rollback completed"
```

7.3 Security Patch Policy

```
## Security Patch Response Times

| Severity | Description | Response Time | Examples |
|----------|-------------|---------------|----------|
| Critical | Active exploitation, data breach risk | 24 hours | RCE, SQL injection, auth bypass |
| High | Significant vulnerability, no active exploit | 7 days | XSS, CSRF, privilege escalation |
| Medium | Limited impact or difficult to exploit | 30 days | Information disclosure, DoS |
| Low | Minimal security impact | Next release | UI issues, minor info leaks |

## Patch Verification

After applying security patches:

1. Run full test suite: `Rscript -e "devtools::test()"`
2. Verify affected functionality manually
3. Check security scanner: `Rscript -e "zzedc::security_scan()"`
4. Review audit logs for anomalies
5. Document patch application in change log
```

Appendix A: Command Reference

System Management

```
# Service control
sudo systemctl start zzedc
sudo systemctl stop zzedc
sudo systemctl restart zzedc
sudo systemctl status zzedc

# Log viewing
sudo journalctl -u zzedc -f
tail -f /var/log/zzedc/app.log
tail -f /var/log/zzedc/error.log

# Database management
sqlite3 /var/data/zzedc/study.db ".tables"
sqlite3 /var/data/zzedc/study.db "PRAGMA integrity_check"
sqlite3 /var/data/zzedc/study.db "VACUUM"
```

R Console Commands

```
# Database health
zzedc::check_database_health("/var/data/zzedc/study.db")

# User management
zzedc::list_users()
zzedc::reset_user_password("username")
zzedc::unlock_user_account("username")

# Audit review
zzedc::get_recent_audit_events(n = 100)
zzedc::search_audit_trail(user = "admin", action = "delete")

# System status
zzedc::get_system_status()
zzedc::get_active_sessions()
```

Appendix B: Monitoring Dashboards

Recommended Metrics for Grafana/Prometheus

```
# Prometheus scrape configuration
scrape_configs:
  - job_name: 'zzedc'
    scrape_interval: 30s
    static_configs:
      - targets: ['localhost:3838']
    metrics_path: '/metrics'
```

Key Performance Indicators

Metric	Warning	Critical	Description
response_time_p95	>2s	>5s	95th percentile response time
error_rate	>1%	>5%	Request error rate
active_sessions	>80% capacity	>95% capacity	Concurrent users
db_query_time_avg	>500ms	>2s	Average query execution
memory_usage	>75%	>90%	System memory utilization
disk_usage	>80%	>95%	Database disk utilization

Document History

Version	Date	Author	Changes
1.0.0	December 2025	ZZedc Team	Initial release