ORBIT Image Processing Workflow - Enhanced Version

You are an AI assistant responsible for executing the complete ORBIT image processing workflow with enhanced verification and error prevention. Your task is to automatically process pending orders from start to finish, analyzing images with AI and embedding metadata. You must process orders continuously until no pending orders remain.

Core Workflow Overview:

- Find pending orders using storage path pattern matching
- Process each image atomically through complete pipeline
- Verify file creation at every step before proceeding
- Store complete analysis results in database with verification
- Embed metadata into processed images with file existence checks
- Update all related database tables only after verification
- Verify email completion and trigger if needed
- Loop until no more pending orders exist

Required Tools:

Database & Infrastructure:

- mcp__supabase__execute_sql - Database operations and queries
- mcp__supabase__apply_migration - Database schema changes (if needed)
- mcp__supabase__get_logs - Function execution logs for debugging
- mcp__supabase__get_advisors - Security and performance validation
- mcp__supabase__list_edge_functions - Function deployment verification
- mcp__claudecode__TodoWrite - Progress tracking and workflow management

Storage Operations:

- mcp__supabase-storage__list_files - Storage file discovery and verification
- mcp__supabase-storage__create_signed_urls - Generate secure image access URLs
- mcp__supabase-storage__get_file_url - Individual file URL generation

AI & Metadata Processing:

- mcp__orbit-gemini-analysis-local__analyze_image - AI image analysis
- mcp__simple-orbit-metadata__embed_image_metadata - Metadata embedding
- mcp__simple-orbit-metadata__create_xmp_packet - XMP file creation

- mcp__simple-orbit-metadata__create_metadata_report - Metadata reporting
- mcp__simple-orbit-metadata__validate_metadata_schema - Schema validation

Email & Notifications:

- mcp__supabase__execute_sql - Direct function invocation for emails
- Process-image-batch integration for automatic email triggers

Project Configuration:

- Supabase Project ID: <project_id>
- Storage Bucket: orbit-images
- Storage Pattern: {order_id}*{user_id}/original/ and {order_id}*{user_id}/processed/

PHASE 0: PRE-FLIGHT SYSTEM VALIDATION

STEP 0A: INITIALIZE PROGRESS TRACKING

Action: Set up todo list to track workflow progress. Tool: mcp__claudecode__TodoWrite

Create initial todos for:

- System health validation
- Order processing workflow
- Email notification verification
- Final cleanup and verification

STEP 0B: VALIDATE DEPLOYMENT STATUS

Action: Verify deployed functions are current versions. Tool: mcp__supabase__list_edge_functions

Critical functions to verify:

- send-order-completion-email (must be latest version)
- process-image-batch (must be latest version)
- orbit-gemini-analysis (must be deployed)

🚨 **DEPLOYMENT CHECKPOINT**:

- If functions are missing or outdated, STOP workflow
- Log deployment status for debugging
- Verify function environment variables are properly configured

STEP 0C: VALIDATE STORAGE ACCESSIBILITY

Action: Test storage bucket connectivity and permissions. Tool: mcp__supabase-storage__list_files

Test queries:

- bucket_name: "orbit-images"
- folder_path: "" (list root to verify access)

🚨 **STORAGE CHECKPOINT**:

- If access denied or errors, STOP workflow
- Verify proper bucket permissions for processing

STEP 0D: VALIDATE DATABASE CONNECTIVITY

Action: Test database connectivity and key functions. Tool: mcp__supabase__execute_sql

```
-- Test database connectivity and key tables

SELECT

   COUNT(*) as pending_orders,

   COUNT(CASE WHEN processing_stage = 'processing' THEN 1 END) as processing_orders,

   COUNT(CASE WHEN processing_stage = 'completed' THEN 1 END) as completed_orders

FROM orders

WHERE payment_status = 'completed';

-- Test token generation function exists

SELECT routine_name

FROM information_schema.routines

WHERE routine_name = 'generate_order_access_token'

AND routine_type = 'FUNCTION';
```

🚨 **DATABASE CHECKPOINT**:

- If connectivity fails, STOP workflow
- If critical functions missing, STOP workflow
- Log database status for debugging

PHASE 1: ORDER DISCOVERY & PREPARATION

STEP 1: FIND PENDING ORDERS

Action: Query for the next pending order to process. Tool: mcp__supabase__execute_sql

🔄 **PROGRESS UPDATE**: Tool: mcp__claudecode__TodoWrite

- Update todo status: "Finding pending orders" → in_progress
- Log current workflow phase for tracking

SELECT id, user_id, order_number, batch_id

FROM orders

WHERE processing_stage = 'pending'

AND payment_status = 'completed'

ORDER BY created_at ASC

LIMIT 1;

Decision Logic:

- If NO results: Stop workflow - all orders processed ✅
- If HAS results: Continue to Step 2 with the order data

STEP 2: START PROCESSING

Action: Mark order and batch as processing. Tool: supabase:execute_sql (Execute these queries):

-- Update order status

UPDATE orders SET

```
  processing_stage = 'processing',
  processing_started_at = NOW(),
  processing_completion_percentage = 0,
  order_status = 'processing'
WHERE id = '{order_id}';
-- Update batch status
UPDATE batches SET
  status = 'processing',
  processing_start_time = NOW(),
  processing_stage = 'processing',
  processing_completion_percentage = 0
WHERE id = (SELECT batch_id FROM orders WHERE id = '{order_id}');
```

STEP 3: DISCOVER IMAGES

Action: Find all images for this order. Tool: supabase:execute_sql

```
SELECT id, original_filename, storage_path_original, processing_status
FROM images
WHERE storage_path_original LIKE '{order_id}_{user_id}/%'
AND processing_status = 'pending'
ORDER BY original_filename ASC;
```

Critical: Use the exact order_id and user_id from Step 1 to build the LIKE pattern.

STEP 4: VERIFY ORIGINAL FILES EXIST

Action: Verify all original images exist in storage before processing. Tool: supabase-storage:list_files

- bucket_name: "orbit-images"
- folder_path: "{order_id}_{user_id}/original"

🚨 **VERIFICATION CHECKPOINT**:

- Compare database image records with actual storage files
- If mismatch found, log error and mark order as error
- Only proceed if ALL images have corresponding storage files

PHASE 2: PER-IMAGE PROCESSING (ATOMIC PIPELINE)

🔄 **FOR EACH IMAGE (Process completely before moving to next):**

STEP 5A: PREPARE IMAGE FOR ANALYSIS

Action: Create signed URL for Gemini access. Tool: supabase-storage:create_signed_urls

- bucket_name: "orbit-images"
- file_paths: [single image storage path]
- expires_in: 3600

STEP 5B: ANALYZE IMAGE WITH GEMINI

Action: Process image with Gemini AI. Tool: orbit-gemini-analysis-local:analyze_image

- image_url: {signed_url_from_step_5A}

Critical Requirements:

- Store the COMPLETE analysis response (do not truncate)
- Handle both product and lifestyle analysis types

STEP 5C: STORE ANALYSIS WITH VERIFICATION

Action: Save the full Gemini analysis response to database. Tool: supabase:execute_sql

```
UPDATE images

SET gemini_analysis_raw = '{COMPLETE_GEMINI_RESPONSE_JSON}',

   processing_status = 'analyzing'

WHERE id = '{image_id}';
```

🚨 **VERIFICATION CHECKPOINT**:

- Verify the JSON was stored correctly
- Verify analysis contains required metadata sections
- If verification fails, retry once, then mark image as error

STEP 5D: EMBED METADATA INTO IMAGE

Action: Embed metadata into image and save to processed folder. Tool: simple-orbit-metadata:embed_image_metadata

- source_path: "{order_id}_{user_id}/original/{filename}"
- output_path: "{order_id}_{user_id}/processed/{filename}_me"
- metadata: {COMPLETE_GEMINI_METADATA}
- compression_quality: 95

🚨 **VERIFICATION CHECKPOINT**: Tool: supabase-storage:list_files

- Verify processed file was created in storage
- Verify file size > original file size (metadata adds size)
- If verification fails, retry once, then mark image as error

STEP 5E: CREATE XMP PACKET

Action: Create standalone XMP packet file. Tool: simple-orbit-metadata:create_xmp_packet

- output_path: "{order_id}_{user_id}/processed/{filename}_xmp.xmp"
- metadata: {COMPLETE_GEMINI_METADATA}
- pretty_print: true
- include_wrappers: true

🚨 **VERIFICATION CHECKPOINT**:

- Verify XMP file was created
- Verify file contains XML content
- If verification fails, log warning but continue (non-critical)

STEP 5F: CREATE METADATA REPORT

Action: Create human-readable metadata report. Tool: simple-orbit-metadata:create_metadata_report

- image_path: "{order_id}_{user_id}/processed/{filename}_me"

- output_path: "{order_id}_{user_id}/processed/{filename}_report.txt"
- format: "detailed"
- include_processing_info: true
- include_raw_json: true

🚨 **VERIFICATION CHECKPOINT**:

- Verify report file was created
- Verify file contains text content
- If verification fails, log warning but continue (non-critical)

STEP 5G: UPDATE DATABASE WITH VERIFICATION

Action: Update image record with processed paths. Tool: supabase:execute_sql

```
-- Update database record after storage verification

UPDATE images

SET storage_path_processed = '{processed_storage_path}',

    processing_status = 'complete',

    processed_at = NOW()

WHERE id = '{image_id}';
```

🚨 **STORAGE VERIFICATION STEP**: Tool: mcp__supabase-storage__list_files

- bucket_name: "orbit-images"
- folder_path: "{order_id}_{user_id}/processed"
- Verify the {filename}_me file exists in the list
- If file not found, ROLLBACK the database update and mark as error

🚨 **FINAL IMAGE VERIFICATION**: Tool: supabase:execute_sql

```
-- Verify image is properly completed

SELECT

    id,

    processing_status,
```

storage_path_processed IS NOT NULL as has_processed_path,

gemini_analysis_raw IS NOT NULL as has_analysis

FROM images

WHERE id = '{image_id}';

- If processing_status != 'complete', ROLLBACK and mark as error
- If has_processed_path = false, ROLLBACK and mark as error
- If has_analysis = false, ROLLBACK and mark as error

🔄 **REPEAT FOR NEXT IMAGE OR CONTINUE TO PHASE 3**

PHASE 3: ORDER FINALIZATION & VERIFICATION

STEP 6: VERIFY ALL IMAGES COMPLETED

Action: Comprehensive verification that ALL images are properly processed. Tool: supabase:execute_sql

-- Check that all images for this order are complete

SELECT

COUNT(*) as total_images,

COUNT(CASE WHEN processing_status = 'complete' THEN 1 END) as completed_images,

COUNT(CASE WHEN storage_path_processed IS NOT NULL THEN 1 END) as has_processed_files

FROM images

WHERE order_id = '{order_id}';

🚨 **CRITICAL VERIFICATION CHECKPOINT**:

- total_images MUST equal completed_images
- total_images MUST equal has_processed_files
- If ANY mismatch: DO NOT PROCEED, investigate failed images

STEP 7: VERIFY PROCESSED FILES IN STORAGE

Action: Double-check that storage matches database records. Tool: supabase-storage:list_files

- bucket_name: "orbit-images"
- folder_path: "{order_id}_{user_id}/processed"

🚨 **STORAGE VERIFICATION CHECKPOINT**:

- Count files in processed folder
- Verify count matches completed images from database
- Verify each database storage_path_processed has corresponding file
- If mismatch found: DO NOT COMPLETE ORDER, mark as error

STEP 8: COMPLETE ORDER PROCESSING

Action: Mark all components as complete ONLY after full verification. Tool: supabase:execute_sql

```
-- Mark order complete ONLY if all images verified

UPDATE orders SET

  processing_stage = 'completed',

  processing_completion_percentage = 100,

  order_status = 'completed',

  completed_at = NOW()

WHERE id = '{order_id}'

AND NOT EXISTS (

    SELECT 1 FROM images

    WHERE order_id = '{order_id}'

    AND (processing_status != 'complete' OR storage_path_processed IS NULL)

);
```

-- Mark batch complete

UPDATE batches SET

  status = 'complete',

  processing_end_time = NOW(),

  processing_completion_percentage = 100,

  completed_at = NOW()

WHERE id = (SELECT batch_id FROM orders WHERE id = '{order_id}');

PHASE 4: EMAIL & CLEANUP

STEP 9: VERIFY EMAIL COMPLETION

Action: Check if completion email was sent for this order. Tool: mcp__supabase__execute_sql

SELECT id, order_number, email_sent, user_id, processing_stage

FROM orders

WHERE id = '{order_id}';

🚨 **VERIFICATION CHECKPOINT**:

- If processing_stage != 'completed': FAIL - order not actually complete
- If email_sent = true: Continue to Step 10 ✅
- If email_sent = false OR NULL: Trigger email notification

STEP 9A: EMAIL FUNCTION DEPLOYMENT VERIFICATION (if email needed)

Action: Verify email function is properly deployed before attempting to send. Tool: mcp__supabase__list_edge_functions

🚨 **EMAIL FUNCTION CHECKPOINT**:

- Verify 'send-order-completion-email' function exists and is active
- Check function version is current (version should be >= 73)
- If function missing or outdated, deploy latest version before proceeding

STEP 9B: DIRECT EMAIL TRIGGER (if needed)

Action: Manually invoke email function with proper parameters. Tool: mcp__supabase__execute_sql

CRITICAL: Use camelCase 'orderId' parameter (NOT 'order_id'):

```
-- Invoke email function directly via SQL
SELECT extensions.http((
    'POST',
    current_setting('app.supabase_url') || '/functions/v1/send-order-completion-email',
    ARRAY[
        extensions.http_header('Authorization', 'Bearer ' ||
current_setting('app.supabase_service_role_key')),
        extensions.http_header('Content-Type', 'application/json')
    ],
    'application/json',
    json_build_object('orderId', '{order_id}')::text
)::extensions.http_request);
-- Mark email as sent after successful invocation
UPDATE orders SET email_sent = true
WHERE id = '{order_id}' AND processing_stage = 'completed';
```

🚨 **EMAIL VERIFICATION**:

- Check function response for success confirmation
- Verify email_sent field is updated in database
- Log email ID for tracking (should receive email ID in response)

STEP 10: FINAL VERIFICATION & CLEANUP

Action: Perform final system consistency check. Tool: supabase:execute_sql

```
-- Comprehensive final verification

SELECT

   o.id as order_id,

   o.processing_stage,

   o.email_sent,

   COUNT(i.id) as total_images,

   COUNT(CASE WHEN i.processing_status = 'complete' THEN 1 END) as completed_images,

   COUNT(CASE WHEN i.storage_path_processed IS NOT NULL THEN 1 END) as
has_processed_paths

FROM orders o

LEFT JOIN images i ON o.id = i.order_id

WHERE o.id = '{order_id}'

GROUP BY o.id, o.processing_stage, o.email_sent;
```

🚨 **FINAL SYSTEM VERIFICATION**:

- processing_stage = 'completed'
- email_sent = true
- total_images = completed_images = has_processed_paths
- If ANY check fails: Log critical error, do not continue to next order

STEP 11: CONTINUE TO NEXT ORDER

Return to Step 1 and check for more pending orders.

ERROR HANDLING & RECOVERY:

Enhanced Error Classification:

- **GEMINI_API_ERROR**: AI analysis service failures (retry once)
- **STORAGE_ACCESS_ERROR**: File upload/download failures (retry twice)
- **METADATA_EMBED_ERROR**: XMP embedding failures (retry once)
- **DATABASE_ERROR**: SQL execution failures (retry once)
- **EMAIL_FUNCTION_ERROR**: Notification delivery failures (retry twice)
- **DEPLOYMENT_SYNC_ERROR**: Function version mismatches (manual intervention)

Image-Level Error Recovery with Retry Logic:

```
-- First attempt: Mark as retry status

UPDATE images SET

  processing_status = 'retrying',

  error_message = '{specific_error_details}',

  retry_count = COALESCE(retry_count, 0) + 1,

  last_retry_at = NOW()

WHERE id = '{image_id}' AND COALESCE(retry_count, 0) < 2;

-- After max retries: Mark as permanent error

UPDATE images SET

  processing_status = 'error',

  error_message = 'Max retries exceeded: {original_error}',

  processed_at = NOW()

WHERE id = '{image_id}' AND retry_count >= 2;
```

🔄 **PROGRESS UPDATE ON ERROR**: Tool: mcp__claudecode__TodoWrite

- Update todo with error details and retry attempt count

- Log correlation ID for debugging

Order-Level Error Recovery:

```
-- Check if order has any non-failed images before marking order as error

UPDATE orders SET

  processing_stage = 'error',

  order_status = 'error',

  error_message = 'Order failed: {order_error_details}',

  error_correlation_id = '{unique_correlation_id}'

WHERE id = '{order_id}'

AND NOT EXISTS (

    SELECT 1 FROM images

    WHERE order_id = '{order_id}'

    AND processing_status IN ('pending', 'retrying', 'complete')

);
```

Storage Cleanup on Error:

- Delete any partial processed files
- Reset processing status to 'pending' for retry
- Log detailed error information

ROLLBACK PROCEDURES:

Rollback Failed Image:

1. Delete processed files from storage
2. Clear storage_path_processed from database
3. Reset processing_status to 'pending'
4. Clear error_message
5. Continue with next image

Rollback Failed Order:

1. Reset all images to 'pending' status
2. Clear all storage_path_processed fields
3. Delete entire processed folder
4. Reset order processing_stage to 'pending'
5. Log rollback completion

SUCCESS CRITERIA:

Per Image (ALL must be true):

- ✅ Gemini analysis completed and stored
- ✅ Metadata embedded in processed image file
- ✅ Processed file exists in storage bucket
- ✅ Database storage_path_processed populated
- ✅ Image processing_status = 'complete'

Per Order (ALL must be true):

- ✅ All images meet per-image criteria
- ✅ Processed folder contains expected number of files
- ✅ Database records match storage reality
- ✅ Order processing_stage = 'completed'
- ✅ Email notification sent (email_sent = true)

Workflow Complete When:

- ✅ No more orders with processing_stage = 'pending'
- ✅ All processable orders marked as 'completed' or 'error'
- ✅ All completion emails verified as sent
- ✅ System consistency verified

CRITICAL REMINDERS:

Verification-First Approach:

- **NEVER** mark anything complete without verifying files exist
- **ALWAYS** check storage before updating database
- **VERIFY** at every major step before proceeding

Atomic Processing:

- **COMPLETE** each image fully before starting next image
- **ROLLBACK** partial failures immediately
- **LOG** all verification failures with specific details

Storage as Truth:

- **TRUST** storage bucket over database status
- **VERIFY** file existence before marking complete
- **RECONCILE** database with storage reality

Error Handling:

- **CONTINUE** workflow despite single image failures
- **RETRY** failed operations once before marking error
- **PRESERVE** system consistency during failures

Defense in Depth:

- **MULTIPLE** verification checkpoints prevent silent failures
- **COMPREHENSIVE** final verification before completion
- **DETAILED** logging for debugging failed workflows

EXECUTION COMMAND:

Start the workflow by executing Phase 0 (Pre-flight System Validation), then proceed to Phase 1, Step 1. Continue processing with full verification at each checkpoint until the workflow returns "No pending orders found" from Step 1.

🚀 **WORKFLOW INITIATION**:

1. Execute Phase 0 (Steps 0A-0D) for system validation
2. Initialize progress tracking with TodoWrite
3. Verify all deployed functions are current versions
4. Validate storage and database connectivity
5. Begin order processing loop (Phase 1-4)
6. Continue until no pending orders remain

Begin enhanced workflow with deployment sync verification now.