Database Functions Documentation

1. Introduction

This document provides comprehensive documentation for all PostgreSQL functions implemented in the geometric maturity tracking application. These functions handle business logic, data validation, security, status calculations, and maintenance operations.

2. Function Categories

2.1. Security and Authorization

- `has_role()` - Role-based access control
- `is_assigned_to_project()` - Project assignment verification

2.2. Status Calculation and Business Logic

- `update_parent_component_status()` - Core status calculation trigger
- `update_parent_status_from_spc()` - SPC-triggered status updates
- `check_spc_conformity()` - SPC compliance validation
- `recalculate_all_parent_statuses()` - Bulk status recalculation

2.3. Data Validation and Integrity

- `validate_data_integrity()` - Comprehensive integrity checks
- `check_cavity_number_against_mold()` - Cavity number validation
- `cleanup_orphaned_data()` - Orphaned data cleanup

2.4. Dashboard and Reporting

- `get_projects_with_details()` - Project dashboard data
- `get_recent_projects_for_user()` - User-specific recent projects
- `get_global_dashboard_stats()` - Global dashboard statistics
- `get_upcoming_project_timelines()` - Project timeline data

2.5. Audit and Maintenance

- `log_changes()` - Audit trail logging
- `archive_old_audit_logs()` - Audit log archiving
- `cleanup_old_notifications()` - Notification cleanup

2.6. Utility Functions

- `update_updated_at_column()` - Automatic timestamp updates
- `update_notification_read_at()` - Notification read status
- `update_project_last_accessed()` - Project access tracking

---

3. Detailed Function Documentation

3.1. Security Functions

3.1.1. has_role(role_to_check TEXT)

```sql
RETURNS boolean
LANGUAGE plpgsql SECURITY DEFINER
```

Purpose: Checks if the current authenticated user has a specific role.

Parameters:

- `role_to_check`: The role name to verify

Returns: `boolean` - True if user has the role, false otherwise

Security Features:

- Uses `SECURITY DEFINER` for elevated privileges
- Returns false for unauthenticated users
- Error handling with warnings logged

Usage Example:

```sql
SELECT has_role('Super User');
SELECT has_role('Supplier Quality');
```

Business Logic:

1. Get current user ID from `auth.uid()`
2. Return false if not authenticated or invalid role
3. Query `user_role_assignments` joined with `user_roles`
4. Handle exceptions gracefully

3.1.2. is_assigned_to_project(project_id_to_check BIGINT)

```sql
RETURNS boolean
LANGUAGE plpgsql SECURITY DEFINER
```

Purpose: Determines if current user can access a specific project.

Parameters:

- `project_id_to_check`: Project ID to verify access for

Returns: `boolean` - True if user has access, false otherwise

Business Logic:

1. High-privilege roles (Super User, Supplier Quality, Engineering) get automatic access
2. Other roles checked against `user_project_assignments`
3. Optimized with role hierarchy check first

Usage in RLS Policies:

```sql
CREATE POLICY "Allow read access to assigned projects"
ON projects FOR SELECT
USING (is_assigned_to_project(id));
```

3.2. Status Calculation Functions

3.2.1. update_parent_component_status()

```sql
RETURNS trigger
LANGUAGE plpgsql
```

Purpose: Core business logic function that calculates parent component status based on cavity evaluations.

Trigger Events:

- AFTER INSERT/UPDATE/DELETE on `cavity_evaluations`

Status Calculation Logic:

1. NEW: No evaluations exist
2. INCOMPLETE_DATA: Not all cavities evaluated
3. KO: Any cavity has dimensional or process issues
4. NOT_OFF_TOOL: Prototype molds with good evaluations
5. OT: Definitive/Low Volume molds at toolmaker with dimensional compliance
6. P_OTOP: Definitive/Low Volume molds with dimensional but not process compliance
7. OTOP: Full compliance (dimensional + process)

Key Features:

- Handles mold type logic (Prototype vs Definitive/Low Volume)
- Location-aware (toolmaker vs production)
- SPC conformity integration
- Comprehensive error handling
- Performance optimized with CTEs

Dependencies:

- `check_spc_conformity()` for process validation
- Mold location `is_toolmaker_location` flag

3.2.2. update_parent_status_from_spc()

sql

```
RETURNS trigger
LANGUAGE plpgsql
```

Purpose: Triggers parent status recalculation when SPC values change.

Trigger Events:

- AFTER INSERT/UPDATE/DELETE on `spc_values`

Mechanism:

- Updates `last_modified` field on related `cavity_evaluations`
- Indirectly triggers `update_parent_component_status()`

3.2.3. check_spc_conformity(p_cavity_evaluation_id BIGINT)

sql

```
RETURNS boolean
LANGUAGE plpgsql STABLE
```

Purpose: Validates SPC conformity for a cavity evaluation.

Parameters:

- `p_cavity_evaluation_id`: Cavity evaluation to check

Conformity Criteria:

- Cp ≥ 1.33
- Cpk ≥ 1.33

Returns:

- `true`: All SPC values meet criteria or no SPC values exist
- `false`: Any SPC value fails criteria or evaluation doesn't exist

Usage:

```sql
SELECT check_spc_conformity(evaluation_id) FROM cavity_evaluations;
```

3.2.4. recalculate_all_parent_statuses(p_project_id BIGINT DEFAULT NULL)

```sql
RETURNS TABLE(parent_id bigint, old_status parent_status_enum,
              new_status parent_status_enum, calculation_time interval)
LANGUAGE plpgsql
```

Purpose: Forces recalculation of parent component statuses.

Parameters:

- `p_project_id`: Optional project filter (NULL = all active projects)

Returns: Table showing status changes with timing information

Mechanism:

- Forces trigger execution by updating `last_modified`
- Only processes active projects
- Returns changes for monitoring

Usage:

```sql
-- Recalculate all active projects
SELECT * FROM recalculate_all_parent_statuses();

-- Recalculate specific project
SELECT * FROM recalculate_all_parent_statuses(123);
```

3.3. Dashboard Functions

3.3.1. get_projects_with_details()

```sql
RETURNS TABLE(...)
LANGUAGE plpgsql SECURITY DEFINER
```

Purpose: Provides comprehensive project data for dashboards.

Returns:

- Project basic info (id, name, code, status)
- Status percentages (OTOP, OT, KO)
- Component counts
- Overdue action plans count
- Next milestone information

Performance Features:

- Optimized CTEs for statistics calculation
- Single query execution
- Pre-calculated aggregations

Usage:

```sql
SELECT * FROM get_projects_with_details()
WHERE project_status = 'Active';
```

3.3.2. get_recent_projects_for_user(user_id_param UUID, limit_param INT DEFAULT 4)

```sql
RETURNS TABLE(...)
LANGUAGE plpgsql SECURITY DEFINER
```

Purpose: Gets recently accessed projects for a specific user.

Parameters:

- `user_id_param`: User UUID
- `limit_param`: Maximum projects to return (default 4)

Ordering Logic:

- `last_accessed` timestamp (DESC)
- Falls back to `assigned_at` if never accessed
- Only active projects

Integration:

- Uses `get_projects_with_details()` for consistency
- Joins with `user_project_assignments`

3.3.3. get_global_dashboard_stats()

```sql
RETURNS json
LANGUAGE plpgsql SECURITY DEFINER
```

Purpose: Provides high-level dashboard statistics.

Returns JSON:

```json
```

```
{
 "active_projects": 15,
 "projects_at_risk": 3,
 "upcoming_deadlines": 8,
 "next_milestones": [
  {
   "project_name": "Project A",
   "milestone_name": "PP",
   "milestone_target_date": "2025-08-15"
  }
 ]
}
```


Risk Calculation:

- Projects with overdue action plans
- Active projects only

3.3.4. get_upcoming_project_timelines(project_limit INT DEFAULT 3)

```sql
RETURNS json
LANGUAGE plpgsql SECURITY DEFINER
```

Purpose: Returns project timeline data for Gantt charts or timeline views.

Parameters:

- `project_limit`: Maximum projects to return (1-100)

Prioritization Logic:

1. Projects with "In Progress" milestones
2. Projects by nearest milestone date
3. Includes project date estimation fallbacks

Returns JSON:

```json
```

```
[
 {
  "project_id": 1,
  "project_name": "Project Alpha",
  "project_start_date": "2025-01-15",
  "project_end_date": "2025-12-31",
  "milestones": [...]
 }
]
```

3.4. Data Validation Functions

3.4.1. validate_data_integrity()

```sql
RETURNS TABLE(check_name text, status text, details text, affected_count bigint)
LANGUAGE plpgsql
```

Purpose: Comprehensive data integrity validation.

Validation Checks:

1. **Cavity Numbers**: Valid ranges against mold total_cavities
2. **Date Consistency**: OT/OTOP date ordering
3. **Orphaned Evaluations**: Evaluations without parent components
4. **Orphaned SPC Values**: SPC values without evaluations

Return Format:

- `check_name`: Description of check
- `status`: 'PASS' or 'FAIL'
- `details`: Human-readable results
- `affected_count`: Number of problematic records

Usage:

```sql
SELECT * FROM validate_data_integrity();
```

3.4.2. check_cavity_number_against_mold()

```sql
RETURNS trigger
LANGUAGE plpgsql
```

Purpose: Validates cavity numbers against mold specifications.

Trigger Events:

- BEFORE INSERT/UPDATE on `cavity_evaluations`

Validation Logic:

- Cavity number must be > 0
- Cavity number must be ≤ mold's `total_cavities`
- Raises exception for invalid values

3.5. Maintenance Functions

3.5.1. archive_old_audit_logs()

```sql
RETURNS TABLE(records_archived bigint, oldest_archived timestamptz, newest_archived timestamptz)
LANGUAGE plpgsql SECURITY DEFINER
```

Purpose: Archives audit log records older than 3 months.

Process:

1. Creates `audit_log_archive` table if not exists
2. Moves records older than 3 months
3. Handles duplicates with `ON CONFLICT`
4. Returns archival statistics

Automation:

- Intended for pg_cron scheduling
- Security definer for proper permissions

3.5.2. cleanup_old_notifications()

```sql
```

```
RETURNS TABLE(deleted_read_count bigint, deleted_unread_count bigint)
LANGUAGE plpgsql SECURITY DEFINER
```

Purpose: Removes old notifications based on read status.

Cleanup Rules:

- Read notifications: > 15 days old
- Unread notifications: > 5 days old

Returns: Count of deleted notifications by type

3.5.3. cleanup_orphaned_data()

```
sql

RETURNS TABLE(cleanup_type text, records_deleted bigint)
LANGUAGE plpgsql SECURITY DEFINER
```

Purpose: Removes orphaned records from related tables.

Cleanup Operations:

1. Cavity evaluations without parent components
2. SPC values without cavity evaluations

Returns: Cleanup statistics by type

3.6. Utility Functions

3.6.1. update_updated_at_column()

```
sql

RETURNS trigger
LANGUAGE plpgsql
```

Purpose: Automatically updates `updated_at` timestamp fields.

Trigger Events:

- BEFORE UPDATE on multiple tables

Implementation:

```
sql
```

```
NEW.updated_at = NOW();
RETURN NEW;
```

3.6.2. update_notification_read_at()

```
sql

RETURNS trigger
LANGUAGE plpgsql
```

Purpose: Manages `read_at` timestamp for notifications.

Logic:

- Sets `read_at` to NOW() when `is_read` changes to true
- Clears `read_at` when `is_read` changes to false

3.6.3. update_project_last_accessed(project_id_param BIGINT, user_id_param UUID)

```
sql

RETURNS void
LANGUAGE plpgsql SECURITY DEFINER
```

Purpose: Updates last access timestamp for user-project combinations.

Parameters:

- `project_id_param`: Project being accessed
- `user_id_param`: User accessing (defaults to current user)

Usage: Called when users navigate to project pages

3.6.4. log_changes()

```
sql

RETURNS trigger
LANGUAGE plpgsql
```

Purpose: Creates audit trail entries for data changes.

Trigger Events:

- AFTER INSERT/UPDATE/DELETE on audited tables

Audit Data:

- Table name and record ID
- Action type (INSERT/UPDATE/DELETE)
- Old and new values (JSONB)
- User making change
- Timestamp

---

4. Function Dependencies

4.1. Inter-Function Dependencies

```
update_parent_component_status()
├── check_spc_conformity()
└── Mold/Location data

get_recent_projects_for_user()
└── get_projects_with_details()

update_parent_status_from_spc()
└── Triggers update_parent_component_status()
```

4.2. External Dependencies

- **Supabase Auth**: `auth.uid()`, `auth.role()`
- **Extensions**: `uuid-ossp`, `pg_cron`

---

5. Performance Considerations

5.1. Optimized Functions

- `get_projects_with_details()`: Uses CTEs for single-pass aggregation
- `update_parent_component_status()`: Optimized queries with proper indexing
- Dashboard functions: Pre-calculated statistics

5.2. Security Definer Usage

- Used sparingly for functions requiring elevated privileges
- Proper input validation and error handling
- Limited to administrative and system functions

5.3. Trigger Performance

- Status calculation triggers optimized for minimal overhead
- Audit triggers use efficient JSONB operations
- Validation triggers fail fast on invalid data

---

6. Security Model

6.1. SECURITY DEFINER Functions

Functions with elevated privileges:

- `has_role()`
- `is_assigned_to_project()`
- `get_*_dashboard_*()` functions
- Maintenance functions (`archive_*`, `cleanup_*`)

6.2. Input Validation

- Null parameter checking
- Range validation (e.g., project_limit bounds)
- SQL injection prevention through parameterized queries

6.3. Error Handling

- Graceful degradation on errors
- Comprehensive logging for debugging
- User-friendly error messages

---

7. Maintenance and Monitoring

7.1. Automated Maintenance

Functions designed for pg_cron automation:

- `archive_old_audit_logs()`: Monthly execution
- `cleanup_old_notifications()`: Weekly execution
- `cleanup_orphaned_data()`: Monthly execution

7.2. Health Checks

- `validate_data_integrity()`: Regular integrity validation
- `recalculate_all_parent_statuses()`: Status consistency verification

7.3. Performance Monitoring

- Functions return timing information where applicable
- Audit logs track function execution
- Dashboard functions optimized for real-time use

---

8. Usage Examples

8.1. Security Checks

```sql
-- Check if current user is Super User
SELECT has_role('Super User');

-- Verify project access
SELECT is_assigned_to_project(123);
```

8.2. Dashboard Data

```sql
-- Get all project details
SELECT * FROM get_projects_with_details();

-- Get user's recent projects
SELECT * FROM get_recent_projects_for_user(auth.uid(), 5);

-- Get global stats
SELECT get_global_dashboard_stats();
```

8.3. Maintenance Operations

```sql
-- Validate system integrity
SELECT * FROM validate_data_integrity();

-- Force status recalculation
SELECT * FROM recalculate_all_parent_statuses();

-- Clean up old data
SELECT * FROM cleanup_old_notifications();
```

8.4. Project Access Tracking

```sql
-- Update last accessed time
SELECT update_project_last_accessed(123);
```

This comprehensive function documentation provides complete coverage of all database functions, their purposes, parameters, business logic, and usage patterns for the geometric maturity tracking application.