PUBLIC RPC FUNCTION DEFINTIONS 13 Jan


```
[
 {
  "schema_name": "public",
  "function_name": "rpc_accept_child_invite",
  "arguments": "p_code text",
  "definition": "CREATE OR REPLACE FUNCTION public.rpc_accept_child_invite(p_code
text)\n RETURNS jsonb\n LANGUAGE plpgsql\n SECURITY DEFINER\n SET search_path TO
'public'\nAS $function$\ndeclare\n  v_user_id uuid := auth.uid();\n  v_child record;\nbegin\n  if
v_user_id is null then\n    raise exception 'Not authenticated';\n  end if;\n\n  if p_code is null or
length(trim(p_code)) < 6 then\n    raise exception 'Invalid code';\n  end if;\n\n  select id,
auth_user_id\n    into v_child\n  from public.children\n  where invitation_code = p_code;\n\n
if not found then\n    raise exception 'Invalid or expired code';\n  end if;\n\n  if
v_child.auth_user_id is not null then\n    raise exception 'Invite already used';\n  end if;\n\n
update public.children\n  set\n    auth_user_id = v_user_id,\n    invitation_accepted_at =
now(),\n    invitation_code = null\n  where id = v_child.id;\n\n  return jsonb_build_object(\n
'ok', true,\n    'child_id', v_child.id\n  );\nend;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_advance_to_next_topic",
  "arguments": "p_revision_session_id uuid",
  "definition": "CREATE OR REPLACE FUNCTION
public.rpc_advance_to_next_topic(p_revision_session_id uuid)\n RETURNS
TABLE(out_current_topic_index integer, out_total_topics integer, out_is_session_complete
boolean, out_next_topic_id uuid, out_topic_name text, out_gamification jsonb)\n LANGUAGE
plpgsql\n SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\nDECLARE\n
v_current_idx int;\n  v_total int;\n  v_planned_session_id uuid;\n  v_topic_ids uuid[];\n
v_next_topic_id uuid;\n  v_topic_name text;\n  v_is_complete boolean := false;\n  v_child_id
uuid;\n  v_points_result jsonb;\n  v_streak_result jsonb;\n  v_achievements_result jsonb;\n
v_gamification jsonb := null;\nBEGIN\n  -- Get current state\n  SELECT
rs.current_topic_index, rs.total_topics, rs.planned_session_id, rs.child_id\n    INTO
v_current_idx, v_total, v_planned_session_id, v_child_id\n  FROM public.revision_sessions
rs\n  WHERE rs.id = p_revision_session_id\n  FOR UPDATE;\n\n  IF NOT FOUND THEN\n
RAISE EXCEPTION 'Revision session not found: %', p_revision_session_id;\n  END IF;\n\n  --
Get topic_ids from planned session\n  SELECT ps.topic_ids INTO v_topic_ids\n  FROM
public.planned_sessions ps\n  WHERE ps.id = v_planned_session_id;\n\n  -- Check if there
are more topics\n  IF v_current_idx < v_total - 1 THEN\n    -- Advance to next topic\n
v_current_idx := v_current_idx + 1;\n    v_next_topic_id := v_topic_ids[v_current_idx + 1]; --
PostgreSQL arrays are 1-indexed\n    \n    -- Get topic name\n    SELECT t.topic_name INTO
v_topic_name\n    FROM public.topics t\n    WHERE t.id = v_next_topic_id;\n\n    -- Update
revision session: advance topic, reset steps\n    UPDATE public.revision_sessions\n    SET \n
current_topic_index = v_current_idx,\n      topic_id = v_next_topic_id,\n      current_step =
'recall',\n      current_step_index = 0,\n      current_item_index = 0\n    WHERE id =
p_revision_session_id;\n\n    -- Reset step statuses for the new topic\n    UPDATE
public.revision_session_steps\n    SET \n      status = CASE WHEN step_key = 'recall' THEN
```

'in_progress' ELSE 'not_started' END,\n started_at = CASE WHEN step_key = 'recall' THEN now() ELSE NULL END,\n current_item_index = 0,\n payload = '{}'::jsonb,\n answer_summary = '{}'::jsonb\n WHERE revision_session_id = p_revision_session_id;\n\n v_is_complete := false;\n ELSE\n -- All topics complete - mark session as complete\n UPDATE public.revision_sessions\n SET \n status = 'completed',\n completed = true,\n completed_at = now(),\n current_step = 'complete'\n WHERE id = p_revision_session_id;\n\n -- Mark planned session as completed\n UPDATE public.planned_sessions\n SET \n status = 'completed',\n completed_at = now()\n WHERE id = v_planned_session_id;\n\n -- GAMIFICATION: Award points\n v_points_result := public.rpc_award_session_points(p_revision_session_id);\n\n -- GAMIFICATION: Update streak\n v_streak_result := public.rpc_update_child_streak(v_child_id);\n\n -- GAMIFICATION: Check achievements\n v_achievements_result := public.rpc_check_and_award_achievements(v_child_id);\n\n v_gamification := jsonb_build_object(\n 'points', v_points_result,\n 'streak', v_streak_result,\n 'achievements', v_achievements_result\n);\n\n v_is_complete := true;\n v_next_topic_id := NULL;\n v_topic_name := NULL;\n END IF;\n\n out_current_topic_index := v_current_idx;\n out_total_topics := v_total;\n out_is_session_complete := v_is_complete;\n out_next_topic_id := v_next_topic_id;\n out_topic_name := v_topic_name;\n out_gamification := v_gamification;\n RETURN NEXT;\nEND;\n$function$\n"
 },
 {
 "schema_name": "public",
 "function_name": "rpc_award_session_points",
 "arguments": "p_revision_session_id uuid",
 "definition": "CREATE OR REPLACE FUNCTION public.rpc_award_session_points(p_revision_session_id uuid)\n RETURNS jsonb\n LANGUAGE plpgsql\n SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\nDECLARE\n v_child_id uuid;\n v_focus_mode_active boolean;\n v_base_points integer := 10;\n v_focus_bonus integer := 5;\n v_total_points integer;\n v_new_balance integer;\n v_new_lifetime integer;\nBEGIN\n -- Get session details\n SELECT child_id, focus_mode_active\n INTO v_child_id, v_focus_mode_active\n FROM public.revision_sessions\n WHERE id = p_revision_session_id;\n\n IF v_child_id IS NULL THEN\n RAISE EXCEPTION 'Revision session not found: %', p_revision_session_id;\n END IF;\n\n -- Ensure gamification rows exist\n PERFORM public.ensure_child_gamification_rows(v_child_id);\n\n -- Calculate points\n v_total_points := v_base_points;\n IF v_focus_mode_active = true THEN\n v_total_points := v_total_points + v_focus_bonus;\n END IF;\n\n -- Record base points transaction\n INSERT INTO public.point_transactions (child_id, points, reason, source_type, source_id)\n VALUES (v_child_id, v_base_points, 'session_complete', 'revision_session', p_revision_session_id);\n\n -- Record focus bonus transaction if applicable\n IF v_focus_mode_active = true THEN\n INSERT INTO public.point_transactions (child_id, points, reason, source_type, source_id)\n VALUES (v_child_id, v_focus_bonus, 'focus_mode_bonus', 'revision_session', p_revision_session_id);\n END IF;\n\n -- Update child_points\n UPDATE public.child_points\n SET \n points_balance = points_balance + v_total_points,\n lifetime_points = lifetime_points + v_total_points,\n updated_at = now()\n WHERE child_id = v_child_id\n RETURNING points_balance, lifetime_points INTO v_new_balance, v_new_lifetime;\n\n RETURN jsonb_build_object(\n 'child_id',

```
v_child_id,\n    'points_awarded', v_total_points,\n    'base_points', v_base_points,\n
'focus_bonus', CASE WHEN v_focus_mode_active THEN v_focus_bonus ELSE 0 END,\n
'new_balance', v_new_balance,\n    'lifetime_points', v_new_lifetime\n
);\nEND;\n$function$\n"
  },
  {
    "schema_name": "public",
    "function_name": "rpc_calculate_available_sessions",
    "arguments": "p_child_id uuid, p_start_date date, p_end_date date",
    "definition": "CREATE OR REPLACE FUNCTION
public.rpc_calculate_available_sessions(p_child_id uuid, p_start_date date, p_end_date
date)\n RETURNS jsonb\n LANGUAGE plpgsql\n STABLE SECURITY DEFINER\n SET
search_path TO 'public'\nAS $function$\nDECLARE\n  v_total_days integer;\n  v_total_weeks
numeric;\n  v_current_date date;\n  v_day_of_week integer;\n  v_sessions_count integer :=
0;\n  v_twenty_min_blocks integer := 0;\n  v_blocked_days integer := 0;\n  v_extra_sessions
integer := 0;\n  v_template_sessions integer;\n  v_template_blocks integer;\n  v_override
record;\n  v_weekly_summary jsonb := '{}'::jsonb;\n  v_day_sessions integer;\n  v_day_blocks
integer;\n  v_day_enabled boolean;\nBEGIN\n  -- Validate inputs\n  IF p_end_date <=
p_start_date THEN\n    RETURN jsonb_build_object(\n      'error', 'End date must be after start
date',\n      'start_date', p_start_date,\n      'end_date', p_end_date\n    );\n  END IF;\n\n
v_total_days := p_end_date - p_start_date + 1;\n  v_total_weeks := v_total_days::numeric /
7;\n\n  -- Initialize weekly summary\n  FOR i IN 0..6 LOOP\n    v_weekly_summary :=
v_weekly_summary || jsonb_build_object(\n      i::text, jsonb_build_object('sessions', 0,
'blocks', 0, 'enabled', false)\n    );\n  END LOOP;\n\n  -- Calculate weekly template totals for
summary\n  FOR i IN 0..6 LOOP\n    SELECT \n      COALESCE(COUNT(s.id), 0)::integer,\n
COALESCE(SUM(CASE s.session_pattern \n        WHEN 'p20' THEN 1 \n        WHEN 'p45'
THEN 2 \n        WHEN 'p70' THEN 3 \n        ELSE 0\n      END), 0)::integer,\n
COALESCE(bool_or(t.is_enabled), false)\n    INTO v_day_sessions, v_day_blocks,
v_day_enabled\n    FROM public.weekly_availability_template t\n    LEFT JOIN
public.weekly_availability_slots s ON s.template_id = t.id\n    WHERE t.child_id = p_child_id
\n      AND t.day_of_week = i\n    GROUP BY t.child_id;\n    \n    v_weekly_summary :=
jsonb_set(\n      v_weekly_summary, \n      ARRAY[i::text], \n      jsonb_build_object(\n
'sessions', COALESCE(v_day_sessions, 0), \n        'blocks', COALESCE(v_day_blocks, 0),\n
'enabled', COALESCE(v_day_enabled, false)\n      )\n    );\n  END LOOP;\n\n  -- Iterate through
each day in the range\n  v_current_date := p_start_date;\n\n  WHILE v_current_date <=
p_end_date LOOP\n    -- Calculate day of week (PostgreSQL DOW: 0=Sunday, we want
0=Monday)\n    v_day_of_week := EXTRACT(DOW FROM v_current_date)::integer;\n
v_day_of_week := CASE WHEN v_day_of_week = 0 THEN 6 ELSE v_day_of_week - 1
END;\n\n    -- Check for date override\n    SELECT * INTO v_override \n    FROM
public.availability_date_overrides \n    WHERE child_id = p_child_id AND override_date =
v_current_date;\n\n    IF FOUND AND v_override.override_type = 'blocked' THEN\n
v_blocked_days := v_blocked_days + 1;\n      \n    ELSIF FOUND AND v_override.override_type
= 'extra' THEN\n      -- Extra sessions from override\n      SELECT \n        COALESCE(COUNT(*),
0)::integer,\n        COALESCE(SUM(CASE session_pattern \n          WHEN 'p20' THEN 1 \n
WHEN 'p45' THEN 2 \n          WHEN 'p70' THEN 3 \n          ELSE 0\n        END), 0)::integer\n
INTO v_template_sessions, v_template_blocks\n      FROM
public.availability_override_slots\n      WHERE override_id = v_override.id;\n\n
```

v_sessions_count := v_sessions_count + COALESCE(v_template_sessions, 0);\n v_twenty_min_blocks := v_twenty_min_blocks + COALESCE(v_template_blocks, 0);\n v_extra_sessions := v_extra_sessions + COALESCE(v_template_sessions, 0);\n \n -- Also add regular template\n SELECT \n COALESCE(COUNT(s.id), 0)::integer,\n COALESCE(SUM(CASE s.session_pattern \n WHEN 'p20' THEN 1 \n WHEN 'p45' THEN 2 \n WHEN 'p70' THEN 3 \n ELSE 0\n END), 0)::integer\n INTO v_template_sessions, v_template_blocks\n FROM public.weekly_availability_template t\n JOIN public.weekly_availability_slots s ON s.template_id = t.id\n WHERE t.child_id = p_child_id \n AND t.day_of_week = v_day_of_week\n AND t.is_enabled = true;\n\n v_sessions_count := v_sessions_count + COALESCE(v_template_sessions, 0);\n v_twenty_min_blocks := v_twenty_min_blocks + COALESCE(v_template_blocks, 0);\n \n ELSE\n -- Use weekly template\n SELECT \n COALESCE(COUNT(s.id), 0)::integer,\n COALESCE(SUM(CASE s.session_pattern \n WHEN 'p20' THEN 1 \n WHEN 'p45' THEN 2 \n WHEN 'p70' THEN 3 \n ELSE 0\n END), 0)::integer\n INTO v_template_sessions, v_template_blocks\n FROM public.weekly_availability_template t\n JOIN public.weekly_availability_slots s ON s.template_id = t.id\n WHERE t.child_id = p_child_id \n AND t.day_of_week = v_day_of_week\n AND t.is_enabled = true;\n\n v_sessions_count := v_sessions_count + COALESCE(v_template_sessions, 0);\n v_twenty_min_blocks := v_twenty_min_blocks + COALESCE(v_template_blocks, 0);\n END IF;\n\n v_current_date := v_current_date + 1;\n END LOOP;\n\n -- Build result\n RETURN jsonb_build_object(\n 'start_date', p_start_date,\n 'end_date', p_end_date,\n 'total_days', v_total_days,\n 'total_weeks', ROUND(v_total_weeks, 1),\n 'total_sessions', v_sessions_count,\n 'total_twenty_min_blocks', v_twenty_min_blocks,\n 'blocked_days', v_blocked_days,\n 'extra_sessions_added', v_extra_sessions,\n 'average_sessions_per_week', CASE \n WHEN v_total_weeks > 0 THEN ROUND(v_sessions_count::numeric / v_total_weeks, 1)\n ELSE 0\n END,\n 'average_blocks_per_week', CASE \n WHEN v_total_weeks > 0 THEN ROUND(v_twenty_min_blocks::numeric / v_total_weeks, 1)\n ELSE 0\n END,\n 'weekly_template_summary', v_weekly_summary\n);\nEND;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_calculate_coverage_distribution",
  "arguments": "p_subject_data jsonb, p_available_sessions integer, p_goal_code text, p_need_cluster_codes text[]",
  "definition": "CREATE OR REPLACE FUNCTION public.rpc_calculate_coverage_distribution(p_subject_data jsonb, p_available_sessions integer, p_goal_code text DEFAULT 'improve_grade'::text, p_need_cluster_codes text[] DEFAULT '{}'::text[])\n RETURNS jsonb\n LANGUAGE plpgsql\n STABLE SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\nDECLARE\n v_subject record;\n v_topic_count integer;\n v_priority_weight numeric;\n v_total_weight numeric := 0;\n v_subjects_data jsonb := '[]'::jsonb;\n v_subject_entry jsonb;\n v_allocated_sessions numeric;\n v_topics_covered numeric;\n v_coverage_percent numeric;\n v_grade_gap integer;\n v_effort_multiplier numeric;\n v_goal_multiplier numeric;\n v_needs_multiplier numeric := 1.0;\n v_total_topics integer := 0;\n v_total_topics_covered numeric := 0;\n v_weighted_coverage numeric := 0;\n v_config_sessions_per_topic numeric;\n v_config_weight_high numeric;\n v_config_weight_medium numeric;\n v_config_weight_low

```
numeric;\n  v_config_default_topics integer;\nBEGIN\n  -- Load config values\n  SELECT
COALESCE(config_value, 1.0) INTO v_config_sessions_per_topic\n  FROM
plan_config_multipliers WHERE config_key = 'sessions_per_topic';\n  \n  SELECT
COALESCE(config_value, 1.0) INTO v_config_weight_high\n  FROM plan_config_multipliers
WHERE config_key = 'priority_weight_high';\n  \n  SELECT COALESCE(config_value, 0.6)
INTO v_config_weight_medium\n  FROM plan_config_multipliers WHERE config_key =
'priority_weight_medium';\n  \n  SELECT COALESCE(config_value, 0.4) INTO
v_config_weight_low\n  FROM plan_config_multipliers WHERE config_key =
'priority_weight_low';\n  \n  SELECT COALESCE(config_value, 50)::integer INTO
v_config_default_topics\n  FROM plan_config_multipliers WHERE config_key =
'default_topics_per_subject';\n\n  -- Set defaults if config not found\n
v_config_sessions_per_topic := COALESCE(v_config_sessions_per_topic, 1.0);\n
v_config_weight_high := COALESCE(v_config_weight_high, 1.0);\n  v_config_weight_medium
:= COALESCE(v_config_weight_medium, 0.6);\n  v_config_weight_low :=
COALESCE(v_config_weight_low, 0.4);\n  v_config_default_topics :=
COALESCE(v_config_default_topics, 50);\n\n  -- Goal multiplier\n  v_goal_multiplier := CASE
p_goal_code\n    WHEN 'pass_exam' THEN 1.0\n    WHEN 'improve_grade' THEN 1.1\n
WHEN 'excel' THEN 1.2\n    ELSE 1.0\n  END;\n\n  -- Needs multiplier\n  IF
p_need_cluster_codes IS NOT NULL THEN\n    IF 'REMEMBERING_FACTS' =
ANY(p_need_cluster_codes) \n      OR 'MEMORY_DIFFICULTIES' =
ANY(p_need_cluster_codes) THEN\n      v_needs_multiplier := v_needs_multiplier + 0.15;\n
END IF;\n    IF 'ADHD_TRAITS' = ANY(p_need_cluster_codes) \n      OR 'ATTENTION_FOCUS'
= ANY(p_need_cluster_codes) THEN\n      v_needs_multiplier := v_needs_multiplier + 0.1;\n
END IF;\n  END IF;\n\n  -- First pass: calculate total weight and collect subject data\n  FOR
v_subject IN\n    SELECT \n      (s->>'subject_id')::uuid AS subject_id,\n      COALESCE(s-
>>'subject_name', 'Subject') AS subject_name,\n      COALESCE((s->>'sort_order')::integer, 1)
AS sort_order,\n      (s->>'current_grade')::integer AS current_grade,\n      (s-
>>'target_grade')::integer AS target_grade\n    FROM jsonb_array_elements(p_subject_data)
AS s\n  LOOP\n    -- Get topic count\n    SELECT COALESCE(\n      (\n        SELECT
COUNT(DISTINCT t.id)::integer\n        FROM topics t\n        JOIN themes th ON t.theme_id =
th.id\n        JOIN components c ON th.component_id = c.id\n        WHERE c.subject_id =
v_subject.subject_id\n      ),\n      v_config_default_topics\n    ) INTO v_topic_count;\n    \n    IF
v_topic_count = 0 THEN\n      v_topic_count := v_config_default_topics;\n    END IF;\n\n    --
Priority weight based on sort order\n    v_priority_weight := CASE\n      WHEN
v_subject.sort_order <= 2 THEN v_config_weight_high\n      WHEN v_subject.sort_order <= 5
THEN v_config_weight_medium\n      ELSE v_config_weight_low\n    END;\n\n    -- Grade gap
effort multiplier (more gap = more effort needed)\n    v_grade_gap := GREATEST(0,
COALESCE(v_subject.target_grade, 5) - COALESCE(v_subject.current_grade, 5));\n
v_effort_multiplier := 1.0 + (v_grade_gap * 0.08);\n\n    -- Combined weight for distribution\n
v_total_weight := v_total_weight + (v_priority_weight * v_effort_multiplier);\n    v_total_topics
:= v_total_topics + v_topic_count;\n\n    -- Store subject data for second pass\n
v_subjects_data := v_subjects_data || jsonb_build_object(\n      'subject_id',
v_subject.subject_id,\n      'subject_name', v_subject.subject_name,\n      'sort_order',
v_subject.sort_order,\n      'topic_count', v_topic_count,\n      'priority_weight',
v_priority_weight,\n      'effort_multiplier', v_effort_multiplier,\n      'combined_weight',
v_priority_weight * v_effort_multiplier,\n      'grade_gap', v_grade_gap,\n      'current_grade',
v_subject.current_grade,\n      'target_grade', v_subject.target_grade,\n      'priority_tier',
```

CASE\n WHEN v_subject.sort_order <= 2 THEN 'high'\n WHEN v_subject.sort_order <= 5 THEN 'medium'\n ELSE 'low'\n END\n);\n END LOOP;\n\n -- Second pass: distribute sessions and calculate coverage\n FOR v_subject_entry IN SELECT * FROM jsonb_array_elements(v_subjects_data)\n LOOP\n -- Allocate sessions proportionally\n v_allocated_sessions := p_available_sessions * \n ((v_subject_entry->>'combined_weight')::numeric / NULLIF(v_total_weight, 0));\n \n -- Apply goal and needs multipliers to required effort (not allocation)\n -- This affects how we interpret coverage\n v_topics_covered := v_allocated_sessions / (v_config_sessions_per_topic * v_goal_multiplier * v_needs_multiplier);\n \n -- Calculate coverage percentage (capped at 100%)\n v_coverage_percent := LEAST(100, \n (v_topics_covered / (v_subject_entry->>'topic_count')::numeric) * 100\n);\n \n v_total_topics_covered := v_total_topics_covered + LEAST(v_topics_covered, (v_subject_entry->>'topic_count')::numeric);\n \n -- Weighted coverage for overall score\n v_weighted_coverage := v_weighted_coverage + \n (v_coverage_percent * (v_subject_entry->>'priority_weight')::numeric);\n\n -- Update subject entry with coverage data\n v_subjects_data := (\n SELECT jsonb_agg(\n CASE \n WHEN elem->>'subject_id' = v_subject_entry->>'subject_id' THEN\n elem || jsonb_build_object(\n 'allocated_sessions', ROUND(v_allocated_sessions),\n 'topics_covered', ROUND(v_topics_covered),\n 'coverage_percent', ROUND(v_coverage_percent, 1)\n)\n ELSE elem\n END\n)\n FROM jsonb_array_elements(v_subjects_data) AS elem\n);\n END LOOP;\n\n -- Calculate overall weighted coverage\n v_weighted_coverage := v_weighted_coverage / NULLIF(v_total_weight, 0);\n\n -- Return result\n RETURN jsonb_build_object(\n 'available_sessions', p_available_sessions,\n 'total_topics', v_total_topics,\n 'total_topics_covered', ROUND(v_total_topics_covered),\n 'overall_coverage_percent', ROUND(v_weighted_coverage, 1),\n 'coverage_status', CASE\n WHEN v_weighted_coverage >= 80 THEN 'excellent'\n WHEN v_weighted_coverage >= 65 THEN 'good'\n WHEN v_weighted_coverage >= 50 THEN 'adequate'\n ELSE 'limited'\n END,\n 'goal_multiplier', v_goal_multiplier,\n 'needs_multiplier', v_needs_multiplier,\n 'subjects', v_subjects_data\n);\nEND;\n$function$\n"
 },
 {
 "schema_name": "public",
 "function_name": "rpc_calculate_recommended_sessions",
 "arguments": "p_subject_data jsonb, p_goal_code text, p_need_cluster_codes text[], p_contingency_percent integer",
 "definition": "CREATE OR REPLACE FUNCTION public.rpc_calculate_recommended_sessions(p_subject_data jsonb, p_goal_code text, p_need_cluster_codes text[], p_contingency_percent integer DEFAULT 10)\n RETURNS jsonb\n LANGUAGE plpgsql\n STABLE SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\nDECLARE\n v_subject record;\n v_topic_count integer;\n v_base_sessions numeric;\n v_grade_gap integer;\n v_gap_multiplier numeric;\n v_priority_factor numeric;\n v_adjusted_sessions numeric;\n v_total_sessions numeric := 0;\n v_with_contingency numeric;\n v_goal_multiplier numeric;\n v_needs_multiplier numeric := 1.0;\n v_recommended_pattern text := 'p45';\n v_needs_advice text := null;\n v_subjects_breakdown jsonb := '[]'::jsonb;\n v_sessions_per_topic numeric;\n v_config_weight_high numeric;\n v_config_weight_medium numeric;\n v_config_weight_low numeric;\n v_config_default_topics integer;\nBEGIN\n -- Load config values\n SELECT

```
COALESCE(config_value, 1.0) INTO v_sessions_per_topic\n  FROM plan_config_multipliers WHERE config_key = 'sessions_per_topic';\n  \n  SELECT COALESCE(config_value, 1.0) INTO v_config_weight_high\n  FROM plan_config_multipliers WHERE config_key = 'priority_weight_high';\n  \n  SELECT COALESCE(config_value, 0.6) INTO v_config_weight_medium\n  FROM plan_config_multipliers WHERE config_key = 'priority_weight_medium';\n  \n  SELECT COALESCE(config_value, 0.4) INTO v_config_weight_low\n  FROM plan_config_multipliers WHERE config_key = 'priority_weight_low';\n  \n  SELECT COALESCE(config_value, 50)::integer INTO v_config_default_topics\n  FROM plan_config_multipliers WHERE config_key = 'default_topics_per_subject';\n\n  -- Set defaults if config not found\n  v_sessions_per_topic := COALESCE(v_sessions_per_topic, 1.0);\n  v_config_weight_high := COALESCE(v_config_weight_high, 1.0);\n  v_config_weight_medium := COALESCE(v_config_weight_medium, 0.6);\n  v_config_weight_low := COALESCE(v_config_weight_low, 0.4);\n  v_config_default_topics := COALESCE(v_config_default_topics, 50);\n\n  -- Determine goal multiplier (reduced from old values)\n  v_goal_multiplier := CASE p_goal_code\n    WHEN 'pass_exam' THEN 1.0\n    WHEN 'improve_grade' THEN 1.1\n    WHEN 'excel' THEN 1.2\n    ELSE 1.0\n  END;\n\n  -- Calculate needs multiplier and determine session pattern\n  IF p_need_cluster_codes IS NOT NULL THEN\n    -- Memory difficulties: +15% (reduced from 20%)\n    IF 'REMEMBERING_FACTS' = ANY(p_need_cluster_codes) \n      OR 'MEMORY_DIFFICULTIES' = ANY(p_need_cluster_codes) THEN\n      v_needs_multiplier := v_needs_multiplier + 0.15;\n    END IF;\n\n    -- Attention/ADHD: +10% and recommend shorter sessions\n    IF 'ADHD_TRAITS' = ANY(p_need_cluster_codes) \n      OR 'ATTENTION_FOCUS' = ANY(p_need_cluster_codes) THEN\n      v_needs_multiplier := v_needs_multiplier + 0.1;\n      v_recommended_pattern := 'p20';\n      v_needs_advice := 'Based on the learning needs you selected, we recommend shorter 20-minute sessions for better focus and retention.';\n    END IF;\n  END IF;\n\n  -- Process each subject\n  FOR v_subject IN\n    SELECT \n      (s->>'subject_id')::uuid AS subject_id,\n      COALESCE((s->>'sort_order')::integer, 1) AS sort_order,\n      (s->>'current_grade')::integer AS current_grade,\n      (s->>'target_grade')::integer AS target_grade\n    FROM jsonb_array_elements(p_subject_data) AS s\n  LOOP\n    -- Get topic count from curriculum or use default\n    SELECT COALESCE(\n      (\n        SELECT COUNT(DISTINCT t.id)::integer\n        FROM topics t\n        JOIN themes th ON t.theme_id = th.id\n        JOIN components c ON th.component_id = c.id\n        WHERE c.subject_id = v_subject.subject_id\n      ),\n      v_config_default_topics\n    ) INTO v_topic_count;\n\n    IF v_topic_count = 0 THEN\n      v_topic_count := v_config_default_topics;\n    END IF;\n\n    -- Calculate base sessions (NEW: uses 1.0 instead of 1.5)\n    v_base_sessions := v_topic_count * v_sessions_per_topic;\n\n    -- Calculate grade gap and multiplier (reduced impact: 8% per grade instead of 10%)\n    v_grade_gap := GREATEST(0, COALESCE(v_subject.target_grade, 5) - COALESCE(v_subject.current_grade, 5));\n    v_gap_multiplier := 1.0 + (v_grade_gap * 0.08);\n\n    -- Calculate priority factor based on sort order (NEW: sharper differentiation)\n    v_priority_factor := CASE\n      WHEN v_subject.sort_order <= 2 THEN v_config_weight_high   -- 1.0\n      WHEN v_subject.sort_order <= 5 THEN v_config_weight_medium -- 0.6\n      ELSE v_config_weight_low                                   -- 0.4\n    END;\n\n    -- Calculate adjusted sessions for this subject\n    v_adjusted_sessions := v_base_sessions * v_gap_multiplier * v_goal_multiplier * v_needs_multiplier * v_priority_factor;\n\n    -- Add to total\n    v_total_sessions := v_total_sessions + v_adjusted_sessions;\n\n    -- Add to breakdown\n
```

```
v_subjects_breakdown := v_subjects_breakdown || jsonb_build_object(\n     'subject_id', v_subject.subject_id,\n     'topic_count', v_topic_count,\n     'base_sessions', ROUND(v_base_sessions),\n     'adjusted_sessions', ROUND(v_adjusted_sessions),\n 'current_grade', v_subject.current_grade,\n     'target_grade', v_subject.target_grade,\n 'grade_gap', v_grade_gap,\n     'gap_multiplier', ROUND(v_gap_multiplier, 2),\n 'priority_factor', v_priority_factor,\n     'sort_order', v_subject.sort_order\n   );\n END LOOP;\n\n  -- Calculate with contingency\n  v_with_contingency := v_total_sessions * (1 + p_contingency_percent::numeric / 100);\n\n  -- Return result\n  RETURN jsonb_build_object(\n    'total_recommended_sessions', ROUND(v_total_sessions),\n 'with_contingency', ROUND(v_with_contingency),\n    'contingency_percent', p_contingency_percent,\n    'subject_count', jsonb_array_length(p_subject_data),\n 'goal_code', p_goal_code,\n    'goal_multiplier', v_goal_multiplier,\n    'needs_multiplier', ROUND(v_needs_multiplier, 2),\n    'recommended_session_pattern', v_recommended_pattern,\n    'needs_advice', v_needs_advice,\n    'subjects_breakdown', v_subjects_breakdown\n  );\nEND;\n$function$\n"
 },
 {
   "schema_name": "public",
   "function_name": "rpc_calculate_sessions_for_coverage",
   "arguments": "p_subject_data jsonb, p_coverage_targets jsonb, p_goal_code text, p_need_cluster_codes text[], p_total_weeks numeric",
   "definition": "CREATE OR REPLACE FUNCTION public.rpc_calculate_sessions_for_coverage(p_subject_data jsonb, p_coverage_targets jsonb DEFAULT '{\"low\": 50, \"high\": 90, \"medium\": 70}'::jsonb, p_goal_code text DEFAULT 'improve_grade'::text, p_need_cluster_codes text[] DEFAULT '{}'::text[], p_total_weeks numeric DEFAULT 8)\n RETURNS jsonb\n LANGUAGE plpgsql\n STABLE SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\nDECLARE\n  v_subject record;\n  v_topic_count integer;\n  v_priority_tier text;\n  v_coverage_target numeric;\n  v_required_sessions numeric;\n  v_total_required_sessions numeric := 0;\n  v_subjects_data jsonb := '[]'::jsonb;\n  v_grade_gap integer;\n  v_effort_multiplier numeric;\n  v_goal_multiplier numeric;\n  v_needs_multiplier numeric := 1.0;\n  v_sessions_per_week numeric;\n  v_sessions_per_day numeric;\n  v_recommended_pattern text := 'p45';\n  v_config_sessions_per_topic numeric;\n  v_config_default_topics integer;\n  v_config_max_sessions_day integer;\n  v_is_realistic boolean;\n  v_realism_message text;\nBEGIN\n  -- Load config values\n  SELECT COALESCE(config_value, 1.0) INTO v_config_sessions_per_topic\n  FROM plan_config_multipliers WHERE config_key = 'sessions_per_topic';\n  \n  SELECT COALESCE(config_value, 50)::integer INTO v_config_default_topics\n  FROM plan_config_multipliers WHERE config_key = 'default_topics_per_subject';\n  \n  SELECT COALESCE(config_value, 4)::integer INTO v_config_max_sessions_day\n  FROM plan_config_multipliers WHERE config_key = 'max_realistic_sessions_per_day';\n\n  -- Set defaults\n  v_config_sessions_per_topic := COALESCE(v_config_sessions_per_topic, 1.0);\n  v_config_default_topics := COALESCE(v_config_default_topics, 50);\n  v_config_max_sessions_day := COALESCE(v_config_max_sessions_day, 4);\n\n  -- Goal multiplier\n  v_goal_multiplier := CASE p_goal_code\n    WHEN 'pass_exam' THEN 1.0\n    WHEN 'improve_grade' THEN 1.1\n    WHEN 'excel' THEN 1.2\n    ELSE 1.0\n  END;\n\n  -- Needs multiplier and session pattern\n  IF p_need_cluster_codes IS NOT NULL THEN\n    IF 'REMEMBERING_FACTS' =
```

```
ANY(p_need_cluster_codes) \n      OR 'MEMORY_DIFFICULTIES' =
ANY(p_need_cluster_codes) THEN\n     v_needs_multiplier := v_needs_multiplier + 0.15;\n
END IF;\n   IF 'ADHD_TRAITS' = ANY(p_need_cluster_codes) \n      OR 'ATTENTION_FOCUS'
= ANY(p_need_cluster_codes) THEN\n     v_needs_multiplier := v_needs_multiplier + 0.1;\n
v_recommended_pattern := 'p20';\n   END IF;\n  END IF;\n\n  -- Process each subject\n  FOR
v_subject IN\n   SELECT \n     (s->>'subject_id')::uuid AS subject_id,\n     COALESCE(s-
>>'subject_name', 'Subject') AS subject_name,\n     COALESCE((s->>'sort_order')::integer, 1)
AS sort_order,\n     (s->>'current_grade')::integer AS current_grade,\n     (s-
>>'target_grade')::integer AS target_grade\n   FROM jsonb_array_elements(p_subject_data)
AS s\n  LOOP\n   -- Get topic count\n   SELECT COALESCE(\n     (\n       SELECT
COUNT(DISTINCT t.id)::integer\n       FROM topics t\n       JOIN themes th ON t.theme_id =
th.id\n       JOIN components c ON th.component_id = c.id\n       WHERE c.subject_id =
v_subject.subject_id\n     ),\n     v_config_default_topics\n   ) INTO v_topic_count;\n   \n   IF
v_topic_count = 0 THEN\n     v_topic_count := v_config_default_topics;\n   END IF;\n\n   --
Determine priority tier and coverage target\n   v_priority_tier := CASE\n     WHEN
v_subject.sort_order <= 2 THEN 'high'\n     WHEN v_subject.sort_order <= 5 THEN
'medium'\n     ELSE 'low'\n   END;\n   \n   v_coverage_target := COALESCE(\n
(p_coverage_targets->>v_priority_tier)::numeric,\n     CASE v_priority_tier\n       WHEN 'high'
THEN 90\n       WHEN 'medium' THEN 70\n       ELSE 50\n     END\n   );\n\n   -- Grade gap
effort multiplier\n   v_grade_gap := GREATEST(0, COALESCE(v_subject.target_grade, 5) -
COALESCE(v_subject.current_grade, 5));\n   v_effort_multiplier := 1.0 + (v_grade_gap *
0.08);\n\n   -- Calculate required sessions for this subject\n   -- topics_to_cover ×
sessions_per_topic × effort_multiplier × goal × needs\n   v_required_sessions :=
(v_topic_count * v_coverage_target / 100) \n     * v_config_sessions_per_topic \n     *
v_effort_multiplier \n     * v_goal_multiplier \n     * v_needs_multiplier;\n\n
v_total_required_sessions := v_total_required_sessions + v_required_sessions;\n\n   -- Add
to result\n   v_subjects_data := v_subjects_data || jsonb_build_object(\n     'subject_id',
v_subject.subject_id,\n     'subject_name', v_subject.subject_name,\n     'sort_order',
v_subject.sort_order,\n     'priority_tier', v_priority_tier,\n     'topic_count', v_topic_count,\n
'coverage_target', v_coverage_target,\n     'topics_to_cover', ROUND((v_topic_count *
v_coverage_target / 100)),\n     'required_sessions', ROUND(v_required_sessions),\n
'grade_gap', v_grade_gap,\n     'effort_multiplier', ROUND(v_effort_multiplier, 2),\n
'current_grade', v_subject.current_grade,\n     'target_grade', v_subject.target_grade\n   );\n
END LOOP;\n\n  -- Calculate sessions per week and day\n  v_sessions_per_week :=
v_total_required_sessions / NULLIF(p_total_weeks, 0);\n  v_sessions_per_day :=
v_sessions_per_week / 6; -- Assuming 6 active days\n\n  -- Check if realistic\n  v_is_realistic
:= v_sessions_per_day <= v_config_max_sessions_day;\n  \n  IF NOT v_is_realistic THEN\n
v_realism_message := format(\n     'This requires %.1f sessions per day, which exceeds the
recommended maximum of %s. Consider extending your revision period or adjusting
coverage targets.',\n     v_sessions_per_day,\n     v_config_max_sessions_day\n   );\n  ELSE\n
v_realism_message := format(\n     'This schedule is realistic at %.1f sessions per day across
6 days.',\n     v_sessions_per_day\n   );\n  END IF;\n\n  -- Return result\n  RETURN
jsonb_build_object(\n   'total_required_sessions', ROUND(v_total_required_sessions),\n
'sessions_per_week', ROUND(v_sessions_per_week, 1),\n   'sessions_per_day',
ROUND(v_sessions_per_day, 1),\n   'total_weeks', p_total_weeks,\n   'is_realistic',
v_is_realistic,\n   'realism_message', v_realism_message,\n   'recommended_pattern',
v_recommended_pattern,\n   'goal_multiplier', v_goal_multiplier,\n   'needs_multiplier',
```

v_needs_multiplier,\n 'coverage_targets', p_coverage_targets,\n 'subjects', v_subjects_data\n);\nEND;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_check_and_award_achievements",
  "arguments": "p_child_id uuid",
  "definition": "CREATE OR REPLACE FUNCTION public.rpc_check_and_award_achievements(p_child_id uuid)\n RETURNS jsonb\n LANGUAGE plpgsql\n SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\nDECLARE\n v_total_sessions integer;\n v_focus_sessions integer;\n v_current_streak integer;\n v_subject_counts jsonb;\n v_achievement record;\n v_newly_earned jsonb := '[]'::jsonb;\n v_points_awarded integer := 0;\nBEGIN\n -- Ensure gamification rows exist\n PERFORM public.ensure_child_gamification_rows(p_child_id);\n\n -- Get total session count from planned_sessions (authoritative source)\n SELECT COUNT(*)\n INTO v_total_sessions\n FROM public.planned_sessions\n WHERE child_id = p_child_id AND status = 'completed';\n\n -- Get focus session count from revision_sessions (where focus_mode is tracked)\n SELECT COUNT(*)\n INTO v_focus_sessions\n FROM public.revision_sessions\n WHERE child_id = p_child_id AND completed = true AND focus_mode_active = true;\n\n -- Get current streak\n SELECT current_streak\n INTO v_current_streak\n FROM public.child_streaks\n WHERE child_id = p_child_id;\n\n -- Get session counts per subject from planned_sessions\n SELECT jsonb_object_agg(subject_id::text, session_count)\n INTO v_subject_counts\n FROM (\n SELECT subject_id, COUNT(*) as session_count\n FROM public.planned_sessions\n WHERE child_id = p_child_id AND status = 'completed'\n GROUP BY subject_id\n) subq;\n\n -- Check each active achievement\n FOR v_achievement IN\n SELECT * FROM public.achievements WHERE is_active = true\n LOOP\n -- Skip if already earned\n IF EXISTS (\n SELECT 1 FROM public.child_achievements \n WHERE child_id = p_child_id AND achievement_id = v_achievement.id\n) THEN\n CONTINUE;\n END IF;\n\n -- Check trigger conditions\n IF v_achievement.trigger_type = 'session_count' \n AND v_total_sessions >= v_achievement.trigger_threshold THEN\n -- Award achievement\n INSERT INTO public.child_achievements (child_id, achievement_id)\n VALUES (p_child_id, v_achievement.id);\n\n -- Award points\n INSERT INTO public.point_transactions (child_id, points, reason, source_type, source_id)\n VALUES (p_child_id, v_achievement.points_value, 'achievement_unlock', 'achievement', v_achievement.id);\n\n UPDATE public.child_points\n SET points_balance = points_balance + v_achievement.points_value,\n lifetime_points = lifetime_points + v_achievement.points_value,\n updated_at = now()\n WHERE child_id = p_child_id;\n\n v_points_awarded := v_points_awarded + v_achievement.points_value;\n v_newly_earned := v_newly_earned || jsonb_build_object(\n 'code', v_achievement.code,\n 'name', v_achievement.name,\n 'description', v_achievement.description,\n 'icon', v_achievement.icon,\n 'points', v_achievement.points_value\n);\n\n ELSIF v_achievement.trigger_type = 'streak_days' \n AND v_current_streak >= v_achievement.trigger_threshold THEN\n -- Award streak achievement\n INSERT INTO public.child_achievements (child_id, achievement_id)\n VALUES (p_child_id, v_achievement.id);\n\n INSERT INTO public.point_transactions (child_id, points, reason, source_type, source_id)\n VALUES (p_child_id,

v_achievement.points_value, 'achievement_unlock', 'achievement', v_achievement.id);\n\n UPDATE public.child_points\n SET points_balance = points_balance + v_achievement.points_value,\n lifetime_points = lifetime_points + v_achievement.points_value,\n updated_at = now()\n WHERE child_id = p_child_id;\n\n v_points_awarded := v_points_awarded + v_achievement.points_value;\n v_newly_earned := v_newly_earned || jsonb_build_object(\n 'code', v_achievement.code,\n 'name', v_achievement.name,\n 'description', v_achievement.description,\n 'icon', v_achievement.icon,\n 'points', v_achievement.points_value\n);\n\n ELSIF v_achievement.trigger_type = 'focus_session_count' \n AND v_focus_sessions >= v_achievement.trigger_threshold THEN\n -- Award focus achievement\n INSERT INTO public.child_achievements (child_id, achievement_id)\n VALUES (p_child_id, v_achievement.id);\n\n INSERT INTO public.point_transactions (child_id, points, reason, source_type, source_id)\n VALUES (p_child_id, v_achievement.points_value, 'achievement_unlock', 'achievement', v_achievement.id);\n\n UPDATE public.child_points\n SET points_balance = points_balance + v_achievement.points_value,\n lifetime_points = lifetime_points + v_achievement.points_value,\n updated_at = now()\n WHERE child_id = p_child_id;\n\n v_points_awarded := v_points_awarded + v_achievement.points_value;\n v_newly_earned := v_newly_earned || jsonb_build_object(\n 'code', v_achievement.code,\n 'name', v_achievement.name,\n 'description', v_achievement.description,\n 'icon', v_achievement.icon,\n 'points', v_achievement.points_value\n);\n\n ELSIF v_achievement.trigger_type = 'subject_session_count' THEN\n -- Check if any subject meets threshold\n IF EXISTS (\n SELECT 1 FROM jsonb_each_text(COALESCE(v_subject_counts, '{}'::jsonb))\n WHERE value::integer >= v_achievement.trigger_threshold\n) THEN\n INSERT INTO public.child_achievements (child_id, achievement_id)\n VALUES (p_child_id, v_achievement.id);\n\n INSERT INTO public.point_transactions (child_id, points, reason, source_type, source_id)\n VALUES (p_child_id, v_achievement.points_value, 'achievement_unlock', 'achievement', v_achievement.id);\n\n UPDATE public.child_points\n SET points_balance = points_balance + v_achievement.points_value,\n lifetime_points = lifetime_points + v_achievement.points_value,\n updated_at = now()\n WHERE child_id = p_child_id;\n\n v_points_awarded := v_points_awarded + v_achievement.points_value;\n v_newly_earned := v_newly_earned || jsonb_build_object(\n 'code', v_achievement.code,\n 'name', v_achievement.name,\n 'description', v_achievement.description,\n 'icon', v_achievement.icon,\n 'points', v_achievement.points_value\n);\n END IF;\n END IF;\n END LOOP;\n\n RETURN jsonb_build_object(\n 'child_id', p_child_id,\n 'newly_earned', v_newly_earned,\n 'achievement_count', jsonb_array_length(v_newly_earned),\n 'points_awarded', v_points_awarded\n);\nEND;\n$function$\n"
 },
 {
 "schema_name": "public",
 "function_name": "rpc_complete_planned_session",
 "arguments": "p_planned_session_id uuid, p_confidence_level text, p_notes text, p_duration_minutes integer",

```
  "definition": "CREATE OR REPLACE FUNCTION
public.rpc_complete_planned_session(p_planned_session_id uuid, p_confidence_level text
DEFAULT NULL::text, p_notes text DEFAULT NULL::text, p_duration_minutes integer
DEFAULT NULL::integer)\n RETURNS TABLE(planned_session_id uuid, status text,
completed_at timestamp with time zone, gamification jsonb)\n LANGUAGE plpgsql\n
SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\nDECLARE\n  v_rs_id
uuid;\n  v_child_id uuid;\n  v_completed_at timestamptz := now();\n  v_points_result jsonb;\n
v_streak_result jsonb;\n  v_achievements_result jsonb;\nBEGIN\n  -- Mark planned session
completed\n  UPDATE public.planned_sessions\n  SET status = 'completed',\n
completed_at = v_completed_at,\n      updated_at = now()\n  WHERE id =
p_planned_session_id;\n\n  -- Get linked revision session\n  SELECT rs.id, rs.child_id \n  INTO
v_rs_id, v_child_id\n  FROM public.revision_sessions rs\n  WHERE rs.planned_session_id =
p_planned_session_id;\n\n  IF v_rs_id IS NOT NULL THEN\n    -- Update revision session\n
UPDATE public.revision_sessions\n    SET status = 'completed',\n       completed = true,\n
confidence_level = COALESCE(p_confidence_level, confidence_level),\n       notes =
COALESCE(p_notes, notes),\n       duration_minutes = COALESCE(p_duration_minutes,
duration_minutes),\n       completed_at = v_completed_at\n    WHERE id = v_rs_id;\n\n    --
Complete any pending steps\n    UPDATE public.revision_session_steps\n    SET status =
'completed',\n       completed_at = COALESCE(completed_at, v_completed_at),\n
started_at = COALESCE(started_at, now())\n    WHERE revision_session_id = v_rs_id\n
AND status <> 'completed';\n\n    -- GAMIFICATION: Award points\n    v_points_result :=
public.rpc_award_session_points(v_rs_id);\n\n    -- GAMIFICATION: Update streak\n
v_streak_result := public.rpc_update_child_streak(v_child_id);\n\n    -- GAMIFICATION:
Check achievements\n    v_achievements_result :=
public.rpc_check_and_award_achievements(v_child_id);\n  END IF;\n\n  planned_session_id
:= p_planned_session_id;\n  status := 'completed';\n  completed_at := v_completed_at;\n
gamification := jsonb_build_object(\n    'points', v_points_result,\n    'streak',
v_streak_result,\n    'achievements', v_achievements_result\n  );\n  RETURN
NEXT;\nEND;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_create_child_invite",
  "arguments": "p_child_id uuid",
  "definition": "CREATE OR REPLACE FUNCTION public.rpc_create_child_invite(p_child_id
uuid)\n RETURNS jsonb\n LANGUAGE plpgsql\n SECURITY DEFINER\n SET search_path TO
'public'\nAS $function$\ndeclare\n  v_parent_id uuid := auth.uid();\n  v_code text;\n  v_row
record;\n  v_link text;\nbegin\n  if v_parent_id is null then\n    raise exception 'Not
authenticated';\n  end if;\n\n  -- Verify ownership\n  select id, parent_id, invitation_code,
auth_user_id\n    into v_row\n  from public.children\n  where id = p_child_id;\n\n  if not found
then\n    raise exception 'Child not found';\n  end if;\n\n  if v_row.parent_id <> v_parent_id
then\n    raise exception 'Not authorised for this child';\n  end if;\n\n  if v_row.auth_user_id is
not null then\n    raise exception 'Child already linked';\n  end if;\n\n  -- Always rotate for
simplicity (keeps behaviour clear)\n  loop\n    v_code := public.generate_invite_code(9);\n
exit when not exists (\n      select 1 from public.children where invitation_code = v_code\n
);\n  end loop;\n\n  update public.children\n  set\n    invitation_code = v_code,\n
invitation_sent_at = now()\n  where id = p_child_id;\n\n  -- If you prefer: construct link on the
```

client. Leaving as code-only is fine.\n v_link := '/child/signup?code=' || v_code;\n\n return jsonb_build_object(\n   'child_id', p_child_id,\n   'invitation_code', v_code,\n   'invitation_link', v_link,\n   'invitation_sent_at', now()\n);\nend;\n$function$\n"
  },
  {
    "schema_name": "public",
    "function_name": "rpc_derive_session_policy",
    "arguments": "p_child_id uuid",
    "definition": "CREATE OR REPLACE FUNCTION public.rpc_derive_session_policy(p_child_id uuid)\n RETURNS jsonb\n LANGUAGE plpgsql\n SECURITY DEFINER\nAS $function$\nDECLARE\n v_base jsonb := '{\n   \"constraints\": { \"avoid_free_text\": false, \"max_difficulty\": 3, \"extra_time_allowed\": false },\n   \"step_budget\": { \"recall_items\": 6, \"practice_items\": 5, \"worked_examples\": 1 }\n }'::jsonb;\n\n v_codes text[] := array[]::text[];\n v_sources text[] := array[]::text[];\n \n v_has_dyslexia boolean := false;\n v_has_adhd boolean := false;\n v_has_autism boolean := false;\n v_has_exam_anxiety boolean := false;\n v_has_overwhelmed boolean := false;\n \n v_dyslexia_diagnosed boolean := false;\n v_adhd_diagnosed boolean := false;\n v_autism_diagnosed boolean := false;\n \n v_has_accommodations boolean := false;\nBEGIN\n -- If the new model isn't present yet, return safe defaults\n IF to_regclass('public.child_need_clusters') IS NULL\n   OR to_regclass('public.need_clusters') IS NULL THEN\n   RETURN v_base;\n END IF;\n\n -- Get all clusters with their sources\n SELECT \n   coalesce(array_agg(upper(c.code)), array[]::text[]),\n   coalesce(array_agg(cnc.source), array[]::text[]),\n   bool_or(cnc.has_exam_accommodations)\n INTO v_codes, v_sources, v_has_accommodations\n FROM public.child_need_clusters cnc\n JOIN public.need_clusters c ON c.id = cnc.cluster_id\n WHERE cnc.child_id = p_child_id;\n\n -- Check for conditions (any source)\n v_has_dyslexia := ('DYSLEXIA' = any(v_codes));\n v_has_adhd := ('ADHD_TRAITS' = any(v_codes));\n v_has_autism := ('AUTISM_ASC' = any(v_codes));\n v_has_exam_anxiety := (array['EXAM_ANXIETY','EXAM_NERVES'] && v_codes);\n v_has_overwhelmed := ('FEELING_OVERWHELMED' = any(v_codes));\n\n -- Check for formal diagnoses\n SELECT \n   bool_or(upper(c.code) = 'DYSLEXIA' AND cnc.source = 'formal_diagnosis'),\n   bool_or(upper(c.code) = 'ADHD_TRAITS' AND cnc.source = 'formal_diagnosis'),\n   bool_or(upper(c.code) = 'AUTISM_ASC' AND cnc.source = 'formal_diagnosis')\n INTO v_dyslexia_diagnosed, v_adhd_diagnosed, v_autism_diagnosed\n FROM public.child_need_clusters cnc\n JOIN public.need_clusters c ON c.id = cnc.cluster_id\n WHERE cnc.child_id = p_child_id;\n\n -- Apply constraints based on needs\n IF v_has_dyslexia OR v_has_autism THEN\n   v_base := jsonb_set(v_base, '{constraints,avoid_free_text}', 'true'::jsonb, true);\n END IF;\n\n IF v_has_overwhelmed OR v_has_exam_anxiety THEN\n   v_base := jsonb_set(v_base, '{constraints,max_difficulty}', '2'::jsonb, true);\n   v_base := jsonb_set(v_base, '{step_budget,practice_items}', '4'::jsonb, true);\n   v_base := jsonb_set(v_base, '{step_budget,worked_examples}', '2'::jsonb, true);\n END IF;\n\n IF v_has_adhd THEN\n   -- Slightly smaller chunks\n   v_base := jsonb_set(v_base, '{step_budget,recall_items}', '5'::jsonb, true);\n   v_base := jsonb_set(v_base, '{step_budget,practice_items}', '4'::jsonb, true);\n END IF;\n\n -- Stronger adjustments for formal diagnoses\n IF v_dyslexia_diagnosed THEN\n   v_base := jsonb_set(v_base, '{constraints,max_difficulty}', '2'::jsonb, true);\n   v_base := jsonb_set(v_base, '{step_budget,recall_items}', '4'::jsonb, true);\n END IF;\n\n IF

v_adhd_diagnosed THEN\n v_base := jsonb_set(v_base, '{step_budget,recall_items}', '4'::jsonb, true);\n v_base := jsonb_set(v_base, '{step_budget,practice_items}', '3'::jsonb, true);\n END IF;\n\n -- Flag if extra time is allowed (for UI/timing purposes)\n IF v_has_accommodations THEN\n v_base := jsonb_set(v_base, '{constraints,extra_time_allowed}', 'true'::jsonb, true);\n END IF;\n\n RETURN v_base;\nEND;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_generate_default_availability_template",
  "arguments": "p_recommended_sessions integer, p_session_pattern text, p_total_weeks integer",
  "definition": "CREATE OR REPLACE FUNCTION public.rpc_generate_default_availability_template(p_recommended_sessions integer, p_session_pattern text DEFAULT 'p45'::text, p_total_weeks integer DEFAULT 8)\n RETURNS jsonb\n LANGUAGE plpgsql\n STABLE SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\nDECLARE\n v_sessions_per_week numeric;\n v_remaining_sessions integer;\n v_template jsonb := '[]'::jsonb;\n v_day_config jsonb;\n v_slots jsonb;\n v_day_sessions integer;\n v_day_names text[] := ARRAY['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'];\n i integer;\nBEGIN\n -- Calculate sessions per week needed\n v_sessions_per_week := CEIL(p_recommended_sessions::numeric / GREATEST(p_total_weeks, 1));\n v_remaining_sessions := v_sessions_per_week::integer;\n\n -- Build template for each day\n FOR i IN 0..6 LOOP\n v_slots := '[]'::jsonb;\n v_day_sessions := 0;\n\n IF i <= 4 THEN\n -- Weekdays (Monday-Friday): afternoon + evening if needed\n \n -- First session: afternoon\n IF v_remaining_sessions > 0 THEN\n v_slots := v_slots || jsonb_build_object(\n 'time_of_day', 'afternoon',\n 'session_pattern', p_session_pattern\n);\n v_day_sessions := v_day_sessions + 1;\n v_remaining_sessions := v_remaining_sessions - 1;\n END IF;\n\n -- Second session: evening (if sessions_per_week > 7)\n IF v_remaining_sessions > 0 AND v_sessions_per_week > 7 THEN\n v_slots := v_slots || jsonb_build_object(\n 'time_of_day', 'evening',\n 'session_pattern', p_session_pattern\n);\n v_day_sessions := v_day_sessions + 1;\n v_remaining_sessions := v_remaining_sessions - 1;\n END IF;\n\n -- Third session: morning (if sessions_per_week > 14, alternate days)\n IF v_remaining_sessions > 0 AND v_sessions_per_week > 14 AND i IN (0, 2, 4) THEN\n v_slots := v_slots || jsonb_build_object(\n 'time_of_day', 'morning',\n 'session_pattern', CASE \n WHEN p_session_pattern = 'p70' THEN 'p45'\n ELSE p_session_pattern \n END\n);\n v_day_sessions := v_day_sessions + 1;\n v_remaining_sessions := v_remaining_sessions - 1;\n END IF;\n\n ELSIF i = 5 THEN\n -- Saturday: morning session\n IF v_remaining_sessions > 0 THEN\n v_slots := v_slots || jsonb_build_object(\n 'time_of_day', 'morning',\n 'session_pattern', CASE \n WHEN p_session_pattern = 'p20' THEN 'p45'\n ELSE p_session_pattern \n END\n);\n v_day_sessions := v_day_sessions + 1;\n v_remaining_sessions := v_remaining_sessions - 1;\n END IF;\n\n -- Second Saturday session if very high load\n IF v_remaining_sessions > 0 AND v_sessions_per_week > 10 THEN\n v_slots := v_slots || jsonb_build_object(\n 'time_of_day', 'afternoon',\n 'session_pattern', p_session_pattern\n);\n v_day_sessions := v_day_sessions + 1;\n

v_remaining_sessions := v_remaining_sessions - 1;\n END IF;\n\n ELSE\n -- Sunday: rest day by default, but add session if really needed\n IF v_remaining_sessions > 2 THEN\n v_slots := v_slots || jsonb_build_object(\n 'time_of_day', 'afternoon',\n 'session_pattern', 'p20'\n);\n v_day_sessions := v_day_sessions + 1;\n v_remaining_sessions := v_remaining_sessions - 1;\n END IF;\n END IF;\n\n v_day_config := jsonb_build_object(\n 'day_of_week', i,\n 'day_name', v_day_names[i + 1],\n 'is_enabled', jsonb_array_length(v_slots) > 0,\n 'slots', v_slots,\n 'session_count', v_day_sessions\n);\n\n v_template := v_template || v_day_config;\n END LOOP;\n\n -- Build result\n RETURN jsonb_build_object(\n 'template', v_template,\n 'summary', jsonb_build_object(\n 'recommended_sessions', p_recommended_sessions,\n 'sessions_per_week', v_sessions_per_week,\n 'total_weeks', p_total_weeks,\n 'recommended_pattern', p_session_pattern,\n 'sessions_allocated', (v_sessions_per_week::integer - v_remaining_sessions)\n),\n 'notes', CASE \n WHEN p_session_pattern = 'p20' \n THEN 'This template uses shorter 20-minute sessions based on learning needs. Sessions are spread across the week for consistency.'\n ELSE 'This template balances revision across weekdays with a lighter weekend schedule. Sunday is a rest day.'\n END\n);\nEND;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_generate_planned_session_payload",
  "arguments": "p_planned_session_id uuid",
  "definition": "CREATE OR REPLACE FUNCTION public.rpc_generate_planned_session_payload(p_planned_session_id uuid)\n RETURNS jsonb\n LANGUAGE plpgsql\nAS $function$\ndeclare\n v_child_id uuid;\n v_subject_id uuid;\n v_topic_ids uuid[];\n v_exam_spec_version_id uuid;\n\n v_subject_name text;\n v_topic_name text;\n v_primary_topic_id uuid;\n\n v_policy jsonb;\n v_avoid_free_text boolean := false;\n v_max_difficulty int := 3;\n\n v_recall_n int := 6;\n v_practice_n int := 5;\n v_worked_n int := 1;\n\n -- reinforce cards (macro contract)\n v_cards jsonb := '[]'::jsonb;\n\n -- worked example (macro contract)\n v_worked jsonb := null;\n\n -- practice question (single question macro contract)\n v_question jsonb := null;\n\n v_payload jsonb;\nbegin\n -- Guardrails\n if to_regclass('public.planned_sessions') is null then\n raise exception 'planned_sessions table missing';\n end if;\n\n select ps.child_id, ps.subject_id, ps.topic_ids, ps.exam_spec_version_id\n into v_child_id, v_subject_id, v_topic_ids, v_exam_spec_version_id\n from public.planned_sessions ps\n where ps.id = p_planned_session_id;\n\n if not found then\n raise exception 'Planned session not found: %', p_planned_session_id;\n end if;\n\n -- Default spec version if null\n if v_exam_spec_version_id is null then\n select esv.id into v_exam_spec_version_id\n from public.exam_spec_versions esv\n where esv.subject_id = v_subject_id\n and esv.is_current = true\n limit 1;\n\n if v_exam_spec_version_id is null then\n raise exception 'No current exam_spec_version found for subject_id=%', v_subject_id;\n end if;\n end if;\n\n -- Primary topic\n if array_length(v_topic_ids, 1) is null or array_length(v_topic_ids, 1) < 1 then\n raise exception 'Planned session has no topics: %', p_planned_session_id;\n end if;\n v_primary_topic_id := v_topic_ids[1];\n\n select s.subject_name into v_subject_name\n from public.subjects s\n where s.id = v_subject_id;\n\n select t.topic_name into v_topic_name\n from public.topics t\n where t.id = v_primary_topic_id;\n\n v_subject_name := coalesce(v_subject_name, 'Revision');\n

```
v_topic_name := coalesce(v_topic_name, 'this topic');\n\n  -- Policy\n  v_policy := public.rpc_derive_session_policy(v_child_id);\n  v_avoid_free_text := coalesce((v_policy #>> '{constraints,avoid_free_text}')::boolean, false);\n  v_max_difficulty := coalesce((v_policy #>> '{constraints,max_difficulty}')::int, 3);\n\n  v_recall_n := greatest(1, coalesce((v_policy #>> '{step_budget,recall_items}')::int, 6));\n  v_practice_n := greatest(1, coalesce((v_policy #>> '{step_budget,practice_items}')::int, 5));\n  v_worked_n := greatest(0, coalesce((v_policy #>> '{step_budget,worked_examples}')::int, 1));\n\n  if to_regclass('public.content_units') is null then\n    raise exception 'content_units table missing';\n  end if;\n\n  /*\n    REINFORCE: flashcards as macro cards[]\n    shape matches SessionRun derivedPayload mapping:\n    { id, front, back }\n  */\n  select coalesce(\n    jsonb_agg(\n      jsonb_build_object(\n        'id', x.id,\n        'front', coalesce(x.content_body->>'prompt',''),\n        'back', coalesce(x.content_body->>'answer','')\n      )\n      order by x.difficulty asc, x.created_at asc\n    ),\n    '[]'::jsonb\n  )\n  into v_cards\n  from (\n    select cu.id, cu.content_body, cu.difficulty, cu.created_at\n    from public.content_units cu\n    where cu.status = 'active'\n      and cu.subject_id = v_subject_id\n      and cu.exam_spec_version_id = v_exam_spec_version_id\n      and cu.topic_id = v_primary_topic_id\n      and cu.content_type = 'flashcard'\n      and cu.difficulty <= v_max_difficulty\n    order by cu.difficulty asc, cu.created_at asc\n    limit v_recall_n\n  ) x;\n\n  /*\n    REINFORCE: worked_example as macro worked_example object\n    keep your existing stored shape:\n    { title, steps: [string...], final_answer }\n  */\n  if v_worked_n > 0 then\n    select cu.content_body\n    into v_worked\n    from public.content_units cu\n    where cu.status = 'active'\n      and cu.subject_id = v_subject_id\n      and cu.exam_spec_version_id = v_exam_spec_version_id\n      and cu.topic_id = v_primary_topic_id\n      and cu.content_type = 'worked_example'\n      and cu.difficulty <= v_max_difficulty\n    order by cu.difficulty asc, cu.created_at asc\n    limit 1;\n  end if;\n\n  /*\n    PRACTICE: single question object (macro contract)\n    Must align to src/types/components.ts:\n      QuestionType = 'numeric' | 'multiple_choice' | 'short_text'\n\n    FIXED: Added '{correct,option_id}' to match seed data format\n  */\n  select\n    jsonb_build_object(\n      'id', x.id,\n      'questionType',\n        case coalesce(x.content_body->>'question_format','mcq')\n          when 'mcq' then 'multiple_choice'\n          when 'multiple_choice' then 'multiple_choice'\n          when 'numeric' then 'numeric'\n          when 'short_text' then 'short_text'\n          else 'short_text'\n        end,\n      'text', coalesce(x.content_body->>'prompt',''),\n      'options', coalesce(\n        (\n          select jsonb_agg(jsonb_build_object('id', o->>'id', 'label', o->>'text'))\n          from jsonb_array_elements(coalesce(x.content_body->'options','[]'::jsonb)) o\n        ),\n        '[]'::jsonb\n      ),\n      'correct_option_id',\n        coalesce(\n          x.content_body #>> '{correct,option_id}',  -- seed data format\n          x.content_body #>> '{correct,optionId}',   -- camelCase variant\n          x.content_body #>> '{correct,id}',         -- legacy fallback\n          null\n        ),\n      'explanation', coalesce(x.content_body->>'worked_solution','')\n    )\n  into v_question\n  from (\n    select cu.id, cu.content_body, cu.difficulty, cu.created_at\n    from public.content_units cu\n    where cu.status = 'active'\n      and cu.subject_id = v_subject_id\n      and cu.exam_spec_version_id = v_exam_spec_version_id\n      and cu.topic_id = v_primary_topic_id\n      and cu.content_type = 'practice_question'\n      and cu.difficulty <= v_max_difficulty\n    order by cu.difficulty asc, cu.created_at asc\n    limit 1\n  ) x;\n\n  v_payload := jsonb_build_object(\n    'plannedSessionId', p_planned_session_id,\n    'generatedAt', now(),\n    'subject', jsonb_build_object('id', v_subject_id, 'name', v_subject_name),\n    'examSpecVersionId', v_exam_spec_version_id,\n    'topicIds', v_topic_ids,\n    'policy', v_policy,\n\n    'recall', jsonb_build_object(\n      'promptText', 'What
```

```
do you remember about ' || v_topic_name || '?',\n     'allowFreeText', (not v_avoid_free_text),\n 'revealAnswerText', 'Write a few bullet points. Then check your notes or textbook.'\n   ),\n\n 'reinforce', jsonb_build_object(\n     'cards', v_cards,\n     'worked_example', v_worked\n ),\n\n   'practice', jsonb_build_object(\n     'question', v_question\n   ),\n\n   'reflection', '{}'::jsonb\n );\n\n  update public.planned_sessions\n  set generated_payload = v_payload,\n updated_at = now()\n  where id = p_planned_session_id;\n\n  return v_payload;\nend;\n$function$\n"
  },
  {
    "schema_name": "public",
    "function_name": "rpc_get_child_gamification_summary",
    "arguments": "p_child_id uuid",
    "definition": "CREATE OR REPLACE FUNCTION public.rpc_get_child_gamification_summary(p_child_id uuid)\n RETURNS jsonb\n LANGUAGE plpgsql\n SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\nDECLARE\n  v_points record;\n  v_streak record;\n  v_achievements jsonb;\nBEGIN\n  -- Ensure gamification rows exist\n  PERFORM public.ensure_child_gamification_rows(p_child_id);\n\n  -- Get points\n  SELECT points_balance, lifetime_points\n  INTO v_points\n  FROM public.child_points\n  WHERE child_id = p_child_id;\n\n  -- Get streak\n  SELECT current_streak, longest_streak, last_completed_date\n  INTO v_streak\n  FROM public.child_streaks\n  WHERE child_id = p_child_id;\n\n  -- Get achievements\n  SELECT jsonb_build_object(\n    'total_earned', COUNT(*),\n    'recent', COALESCE(jsonb_agg(\n      jsonb_build_object(\n        'code', a.code,\n        'name', a.name,\n        'description', a.description,\n        'icon', a.icon,\n        'earned_at', ca.earned_at\n      ) ORDER BY ca.earned_at DESC\n    ) FILTER (WHERE a.code IS NOT NULL), '[]'::jsonb),\n    'unnotified', COALESCE((\n      SELECT jsonb_agg(\n        jsonb_build_object(\n          'id', ca2.id,\n          'code', a2.code,\n          'name', a2.name,\n          'description', a2.description,\n          'icon', a2.icon,\n          'points', a2.points_value,\n          'earned_at', ca2.earned_at\n        )\n      )\n      FROM public.child_achievements ca2\n      JOIN public.achievements a2 ON a2.id = ca2.achievement_id\n      WHERE ca2.child_id = p_child_id AND ca2.notified = false\n    ), '[]'::jsonb)\n  )\n  INTO v_achievements\n  FROM public.child_achievements ca\n  LEFT JOIN public.achievements a ON a.id = ca.achievement_id\n  WHERE ca.child_id = p_child_id;\n\n  RETURN jsonb_build_object(\n    'child_id', p_child_id,\n    'points', jsonb_build_object(\n      'balance', COALESCE(v_points.points_balance, 0),\n      'lifetime', COALESCE(v_points.lifetime_points, 0)\n    ),\n    'streak', jsonb_build_object(\n      'current', COALESCE(v_streak.current_streak, 0),\n      'longest', COALESCE(v_streak.longest_streak, 0),\n      'last_completed_date', v_streak.last_completed_date\n    ),\n    'achievements', COALESCE(v_achievements, '{}'::jsonb)\n  );\nEND;\n$function$\n"
  },
  {
    "schema_name": "public",
    "function_name": "rpc_get_child_invite_preview",
    "arguments": "p_code text",
    "definition": "CREATE OR REPLACE FUNCTION public.rpc_get_child_invite_preview(p_code text)\n RETURNS jsonb\n LANGUAGE plpgsql\n SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\ndeclare\n  v_row
```

record;\n v_parent_name text;\nbegin\n if p_code is null or length(trim(p_code)) < 6 then\n return jsonb_build_object('ok', false);\n end if;\n\n select c.id as child_id,\n c.first_name as child_first_name,\n c.parent_id as parent_id,\n c.auth_user_id as auth_user_id,\n c.invitation_code as invitation_code\n into v_row\n from public.children c\n where c.invitation_code = p_code;\n\n if not found then\n return jsonb_build_object('ok', false);\n end if;\n\n -- Already linked: treat as invalid for signup UX\n if v_row.auth_user_id is not null then\n return jsonb_build_object('ok', false);\n end if;\n\n -- Parent name (optional). If missing, keep it generic.\n select p.full_name\n into v_parent_name\n from public.profiles p\n where p.id = v_row.parent_id;\n\n return jsonb_build_object(\n 'ok', true,\n 'child_id', v_row.child_id,\n 'child_first_name', coalesce(v_row.child_first_name, 'Student'),\n 'parent_name', coalesce(v_parent_name, 'Your parent')\n);\nend;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_get_clusters_by_area",
  "arguments": "p_area jcq_area",
  "definition": "CREATE OR REPLACE FUNCTION public.rpc_get_clusters_by_area(p_area jcq_area)\n RETURNS TABLE(code text, name text, condition_name text, parent_friendly_name text, description text, example_signs text[], typically_has_accommodations boolean, common_arrangements text[], sort_order integer)\n LANGUAGE sql\n STABLE SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\n SELECT \n nc.code,\n nc.name,\n nc.condition_name,\n nc.parent_friendly_name,\n nc.description,\n nc.example_signs,\n COALESCE(nc.typically_has_accommodations, false),\n nc.common_arrangements,\n COALESCE(nc.sort_order, 9999)\n FROM public.need_clusters nc\n WHERE nc.jcq_area = p_area\n AND nc.is_active = true\n ORDER BY nc.sort_order ASC, nc.name ASC;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_get_curriculum_topic_counts",
  "arguments": "p_subject_ids uuid[]",
  "definition": "CREATE OR REPLACE FUNCTION public.rpc_get_curriculum_topic_counts(p_subject_ids uuid[])\n RETURNS TABLE(subject_id uuid, subject_name text, component_count integer, theme_count integer, topic_count integer)\n LANGUAGE sql\n STABLE SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\n SELECT \n s.id as subject_id,\n s.subject_name,\n COUNT(DISTINCT c.id)::integer as component_count,\n COUNT(DISTINCT th.id)::integer as theme_count,\n COUNT(DISTINCT t.id)::integer as topic_count\n FROM public.subjects s\n LEFT JOIN public.components c ON c.subject_id = s.id\n LEFT JOIN public.themes th ON th.component_id = c.id\n LEFT JOIN public.topics t ON t.theme_id = th.id\n WHERE s.id = ANY(p_subject_ids)\n GROUP BY s.id, s.subject_name\n ORDER BY s.subject_name;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_get_my_child_id",

```
  "arguments": "",
  "definition": "CREATE OR REPLACE FUNCTION public.rpc_get_my_child_id()\n RETURNS uuid\n LANGUAGE sql\n STABLE SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\n  select c.id\n  from public.children c\n  where c.auth_user_id = auth.uid()\n  order by c.created_at desc\n  limit 1;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_get_parent_dashboard_summary",
  "arguments": "p_parent_id uuid, p_week_start date",
  "definition": "CREATE OR REPLACE FUNCTION public.rpc_get_parent_dashboard_summary(p_parent_id uuid, p_week_start date DEFAULT NULL::date)\n RETURNS jsonb\n LANGUAGE plpgsql\n STABLE SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\nDECLARE\n  v_week_start date;\n  v_week_end date;\n  v_prev_week_start date;\n  v_prev_week_end date;\n  v_result jsonb;\nBEGIN\n  v_week_start := COALESCE(p_week_start, date_trunc('week', CURRENT_DATE)::date);\n  v_week_end := v_week_start + INTERVAL '6 days';\n  v_prev_week_start := v_week_start - INTERVAL '7 days';\n  v_prev_week_end := v_week_start - INTERVAL '1 day';\n\n  WITH \n  day_mapping AS (\n    SELECT * FROM (VALUES \n      (0, 'monday', 'Mon'),\n      (1, 'tuesday', 'Tue'),\n      (2, 'wednesday', 'Wed'),\n      (3, 'thursday', 'Thu'),\n      (4, 'friday', 'Fri'),\n      (5, 'saturday', 'Sat'),\n      (6, 'sunday', 'Sun')\n    ) AS t(day_index, day_name_full, day_name_short)\n  ),\n\n  -- UPDATED: Added auth_user_id and invitation_code\n  parent_children AS (\n    SELECT \n      c.id AS child_id,\n      COALESCE(c.preferred_name, c.first_name) AS child_name,\n      c.first_name,\n      c.last_name,\n      c.year_group,\n      c.country,\n      c.auth_user_id,\n      c.invitation_code\n    FROM children c\n    WHERE c.parent_id = p_parent_id\n  ),\n\n  child_subject_details AS (\n    SELECT \n      cs.child_id,\n      s.id AS subject_id,\n      s.subject_name,\n      COALESCE(s.color, '#6B7280') AS color,\n      COALESCE(s.icon, 'book') AS icon,\n      et.name AS exam_type_name\n    FROM child_subjects cs\n    JOIN subjects s ON s.id = cs.subject_id\n    LEFT JOIN exam_types et ON et.id = s.exam_type_id\n    WHERE cs.child_id IN (SELECT child_id FROM parent_children)\n  ),\n\n  active_plans AS (\n    SELECT \n      rp.child_id,\n      rp.id AS plan_id,\n      rp.start_date,\n      rp.exam_timeline,\n      CASE rp.exam_timeline\n        WHEN '1_month' THEN rp.start_date + INTERVAL '1 month'\n        WHEN '3_months' THEN rp.start_date + INTERVAL '3 months'\n        WHEN '6_months' THEN rp.start_date + INTERVAL '6 months'\n        WHEN '12_months' THEN rp.start_date + INTERVAL '12 months'\n        ELSE NULL\n      END AS estimated_exam_date\n    FROM revision_plans rp\n    WHERE rp.child_id IN (SELECT child_id FROM parent_children)\n      AND rp.status = 'active'\n  ),\n\n  week_sessions AS (\n    SELECT \n      ps.child_id,\n      COUNT(*) FILTER (WHERE ps.status = 'completed') AS sessions_completed,\n      COUNT(*) AS sessions_total,\n      SUM(ps.session_duration_minutes) FILTER (WHERE ps.status = 'completed') AS time_spent_minutes,\n      COUNT(DISTINCT ps.session_date) FILTER (WHERE ps.status = 'completed') AS days_active,\n      COUNT(DISTINCT ps.subject_id) FILTER (WHERE ps.status = 'completed') AS subjects_active\n    FROM planned_sessions ps\n    WHERE ps.child_id IN (SELECT child_id FROM parent_children)\n      AND ps.session_date BETWEEN v_week_start AND v_week_end\n    GROUP BY ps.child_id\n  ),\n\n  prev_week_sessions AS (\n    SELECT \n      ps.child_id,\n      COUNT(*) FILTER (WHERE ps.status = 'completed') AS sessions_completed\n    FROM planned_sessions ps\n    WHERE ps.child_id IN (SELECT
```

```
child_id FROM parent_children)\n    AND ps.session_date BETWEEN v_prev_week_start
AND v_prev_week_end\n  GROUP BY ps.child_id\n ),\n\n week_topics AS (\n  SELECT \n
ps.child_id,\n    COUNT(DISTINCT topic_id) AS topics_covered\n  FROM planned_sessions
ps\n  CROSS JOIN LATERAL unnest(ps.topic_ids) AS topic_id\n  WHERE ps.child_id IN
(SELECT child_id FROM parent_children)\n    AND ps.session_date BETWEEN v_week_start
AND v_week_end\n    AND ps.status = 'completed'\n  GROUP BY ps.child_id\n ),\n\n
next_focus AS (\n  SELECT DISTINCT ON (ps.child_id)\n    ps.child_id,\n    ps.id AS
planned_session_id,\n    s.subject_name,\n    t.topic_name,\n    ps.session_date\n  FROM
planned_sessions ps\n  JOIN subjects s ON s.id = ps.subject_id\n  LEFT JOIN topics t ON
t.id = ps.topic_ids[1]\n  WHERE ps.child_id IN (SELECT child_id FROM parent_children)\n
AND ps.status = 'planned'\n    AND ps.session_date >= CURRENT_DATE\n  ORDER BY
ps.child_id, ps.session_date, ps.session_index\n ),\n\n child_gamification AS (\n  SELECT
\n    pc.child_id,\n    COALESCE(cp.points_balance, 0) AS points_balance,\n
COALESCE(cp.lifetime_points, 0) AS lifetime_points,\n    COALESCE(cs.current_streak, 0)
AS current_streak,\n    COALESCE(cs.longest_streak, 0) AS longest_streak,\n    (\n
SELECT jsonb_build_object(\n      'code', a.code,\n      'name', a.name,\n      'icon',
a.icon,\n      'earned_at', ca.earned_at\n    )\n    FROM child_achievements ca\n    JOIN
achievements a ON a.id = ca.achievement_id\n    WHERE ca.child_id = pc.child_id\n
ORDER BY ca.earned_at DESC\n    LIMIT 1\n  ) AS recent_achievement\n  FROM
parent_children pc\n  LEFT JOIN child_points cp ON cp.child_id = pc.child_id\n  LEFT JOIN
child_streaks cs ON cs.child_id = pc.child_id\n ),\n\n -- UPDATED: Added has_signed_up
and invitation_code\n children_data AS (\n  SELECT jsonb_agg(\n    jsonb_build_object(\n
'child_id', pc.child_id,\n    'child_name', pc.child_name,\n    'year_group', pc.year_group,\n
'exam_type', COALESCE(\n      (SELECT csd.exam_type_name FROM child_subject_details
csd WHERE csd.child_id = pc.child_id LIMIT 1),\n      'GCSE'\n    ),\n    'subjects',
COALESCE(\n      (SELECT jsonb_agg(\n        jsonb_build_object(\n          'subject_id',
csd.subject_id,\n          'subject_name', csd.subject_name,\n          'color', csd.color,\n
'icon', csd.icon\n        )\n      ) FROM child_subject_details csd WHERE csd.child_id =
pc.child_id),\n      '[]'::jsonb\n    ),\n    'week_sessions_completed',
COALESCE(ws.sessions_completed, 0),\n    'week_sessions_total',
COALESCE(ws.sessions_total, 0),\n    'prev_week_sessions_completed',
COALESCE(pws.sessions_completed, 0),\n    'week_topics_covered',
COALESCE(wt.topics_covered, 0),\n    'next_focus', CASE \n      WHEN
nf.planned_session_id IS NOT NULL THEN\n        jsonb_build_object(\n
'subject_name', nf.subject_name,\n          'topic_name', nf.topic_name,\n
'session_date', nf.session_date\n        )\n      ELSE NULL\n    END,\n    'mocks_flag',
CASE\n      WHEN ap.estimated_exam_date IS NOT NULL \n        AND CURRENT_DATE >=
(ap.estimated_exam_date - INTERVAL '4 weeks')\n        AND CURRENT_DATE <
ap.estimated_exam_date THEN\n        jsonb_build_object(\n          'show', true,\n
'weeks_until', GREATEST(1, EXTRACT(DAY FROM (ap.estimated_exam_date -
CURRENT_DATE))::int / 7),\n          'message', 'Mocks in ' || GREATEST(1, EXTRACT(DAY
FROM (ap.estimated_exam_date - CURRENT_DATE))::int / 7) || ' weeks'\n        )\n
ELSE\n        jsonb_build_object('show', false, 'weeks_until', NULL, 'message', NULL)\n
END,\n    'gamification', jsonb_build_object(\n      'points_balance',
COALESCE(cg.points_balance, 0),\n      'lifetime_points', COALESCE(cg.lifetime_points,
0),\n      'current_streak', COALESCE(cg.current_streak, 0),\n      'longest_streak',
COALESCE(cg.longest_streak, 0),\n      'recent_achievement', cg.recent_achievement\n
```

),\n 'has_signed_up', (pc.auth_user_id IS NOT NULL),\n 'invitation_code', pc.invitation_code\n)\n ORDER BY pc.child_name\n) AS data\n FROM parent_children pc\n LEFT JOIN week_sessions ws ON ws.child_id = pc.child_id\n LEFT JOIN prev_week_sessions pws ON pws.child_id = pc.child_id\n LEFT JOIN week_topics wt ON wt.child_id = pc.child_id\n LEFT JOIN next_focus nf ON nf.child_id = pc.child_id\n LEFT JOIN active_plans ap ON ap.child_id = pc.child_id\n LEFT JOIN child_gamification cg ON cg.child_id = pc.child_id\n),\n\n week_summary_data AS (\n SELECT jsonb_build_object(\n 'total_sessions_completed', COALESCE(SUM(ws.sessions_completed), 0),\n 'total_sessions_planned', COALESCE(SUM(ws.sessions_total), 0),\n 'comparison_to_last_week', COALESCE(SUM(ws.sessions_completed), 0) - COALESCE(SUM(pws.sessions_completed), 0),\n 'topics_covered', COALESCE(SUM(wt.topics_covered), 0),\n 'subjects_span', COALESCE(SUM(ws.subjects_active), 0),\n 'time_spent_minutes', COALESCE(SUM(ws.time_spent_minutes), 0),\n 'average_session_minutes', CASE \n WHEN COALESCE(SUM(ws.sessions_completed), 0) > 0 \n THEN (COALESCE(SUM(ws.time_spent_minutes), 0) / SUM(ws.sessions_completed))::int\n ELSE 0\n END,\n 'days_active', (\n SELECT COUNT(DISTINCT ps.session_date)\n FROM planned_sessions ps\n WHERE ps.child_id IN (SELECT child_id FROM parent_children)\n AND ps.session_date BETWEEN v_week_start AND v_week_end\n AND ps.status = 'completed'\n)\n) AS data\n FROM parent_children pc\n LEFT JOIN week_sessions ws ON ws.child_id = pc.child_id\n LEFT JOIN prev_week_sessions pws ON pws.child_id = pc.child_id\n LEFT JOIN week_topics wt ON wt.child_id = pc.child_id\n),\n\n daily_pattern_data AS (\n SELECT jsonb_agg(\n jsonb_build_object(\n 'day_index', dm.day_index,\n 'day_name', dm.day_name_short,\n 'sessions_completed', COALESCE(day_stats.sessions_completed, 0),\n 'sessions_planned', COALESCE(day_stats.sessions_planned, 0),\n 'sessions_total', COALESCE(day_stats.sessions_total, 0),\n 'total_minutes', COALESCE(day_stats.total_minutes, 0),\n 'planned_minutes', COALESCE(day_stats.planned_minutes, 0),\n 'is_rest_day', COALESCE(NOT EXISTS (\n SELECT 1 FROM revision_schedules rs \n WHERE rs.child_id IN (SELECT child_id FROM parent_children)\n AND rs.day_of_week = dm.day_name_full\n AND rs.is_active = true\n), true)\n)\n ORDER BY dm.day_index\n) AS data\n FROM day_mapping dm\n LEFT JOIN (\n SELECT \n EXTRACT(ISODOW FROM ps.session_date)::int - 1 AS day_index,\n COUNT(*) FILTER (WHERE ps.status = 'completed') AS sessions_completed,\n COUNT(*) FILTER (WHERE ps.status IN ('planned', 'started')) AS sessions_planned,\n COUNT(*) FILTER (WHERE ps.status != 'skipped') AS sessions_total,\n SUM(ps.session_duration_minutes) FILTER (WHERE ps.status = 'completed') AS total_minutes,\n SUM(ps.session_duration_minutes) FILTER (WHERE ps.status IN ('planned', 'started')) AS planned_minutes\n FROM planned_sessions ps\n WHERE ps.child_id IN (SELECT child_id FROM parent_children)\n AND ps.session_date BETWEEN v_week_start AND v_week_end\n GROUP BY EXTRACT(ISODOW FROM ps.session_date)::int - 1\n) day_stats ON day_stats.day_index = dm.day_index\n),\n\n reminders_mocks AS (\n SELECT \n 'mocks_coming_up'::text AS type,\n 1 AS priority,\n pc.child_id,\n pc.child_name,\n 'Mocks coming up in ' || GREATEST(1, EXTRACT(DAY FROM (ap.estimated_exam_date - CURRENT_DATE))::int / 7) || ' weeks' AS message,\n 'Review revision plan' AS action_label,\n '/parent/child/' || pc.child_id || '/plan' AS action_route,\n jsonb_build_object(\n 'weeks_until', GREATEST(1,

```
EXTRACT(DAY FROM (ap.estimated_exam_date - CURRENT_DATE))::int / 7)\n    ) AS
metadata\n   FROM parent_children pc\n   JOIN active_plans ap ON ap.child_id =
pc.child_id\n   WHERE ap.estimated_exam_date IS NOT NULL\n    AND CURRENT_DATE >=
(ap.estimated_exam_date - INTERVAL '4 weeks')\n    AND CURRENT_DATE <
ap.estimated_exam_date\n ),\n\n reminders_revisit AS (\n   SELECT DISTINCT ON
(rs.child_id, t.id)\n    'topic_to_revisit'::text AS type,\n    2 AS priority,\n    pc.child_id,\n
pc.child_name,\n    pc.child_name || ' might benefit from another look at ' || t.topic_name AS
message,\n    'Add to plan' AS action_label,\n    '/parent/child/' || pc.child_id || '/plan' AS
action_route,\n    jsonb_build_object(\n      'topic_id', t.id,\n      'topic_name',
t.topic_name,\n      'confidence_level', rs.confidence_level\n    ) AS metadata\n   FROM
parent_children pc\n   JOIN revision_sessions rs ON rs.child_id = pc.child_id\n   JOIN topics t
ON t.id = rs.topic_id\n   WHERE rs.completed = true\n    AND rs.confidence_level IN
('struggling', 'needs_work')\n    AND rs.session_date >= CURRENT_DATE - INTERVAL '14
days'\n   ORDER BY rs.child_id, t.id, rs.session_date DESC\n ),\n\n reminders_momentum
AS (\n   SELECT \n    'building_momentum'::text AS type,\n    3 AS priority,\n    pc.child_id,\n
pc.child_name,\n    pc.child_name || ' has completed ' || cs.current_streak || ' consecutive
revision days' AS message,\n    NULL::text AS action_label,\n    NULL::text AS
action_route,\n    jsonb_build_object(\n      'streak_days', cs.current_streak,\n
'longest_streak', cs.longest_streak,\n      'last_completed_date', cs.last_completed_date\n
) AS metadata\n   FROM parent_children pc\n   JOIN child_streaks cs ON cs.child_id =
pc.child_id\n   WHERE cs.current_streak >= 3\n ),\n\n reminders_neglected AS (\n   SELECT
\n    'subject_neglected'::text AS type,\n    4 AS priority,\n    csd.child_id,\n
pc.child_name,\n    csd.subject_name || ' hasn''t been revised in over a week' AS message,\n
'Check schedule' AS action_label,\n    '/parent/child/' || csd.child_id || '/plan' AS
action_route,\n    jsonb_build_object(\n      'subject_id', csd.subject_id,\n
'subject_name', csd.subject_name,\n      'days_since_last',
COALESCE(last_session.days_since, 999)\n    ) AS metadata\n   FROM
child_subject_details csd\n   JOIN parent_children pc ON pc.child_id = csd.child_id\n   LEFT
JOIN LATERAL (\n    SELECT \n      (CURRENT_DATE - MAX(ps.session_date))::int AS
days_since\n    FROM planned_sessions ps\n    WHERE ps.child_id = csd.child_id\n
AND ps.subject_id = csd.subject_id\n      AND ps.status = 'completed'\n   ) last_session ON
true\n   WHERE COALESCE(last_session.days_since, 999) > 10\n ),\n\n all_reminders AS
(\n   SELECT * FROM reminders_mocks\n   UNION ALL\n   SELECT * FROM
reminders_revisit\n   UNION ALL\n   SELECT * FROM reminders_momentum\n   UNION
ALL\n   SELECT * FROM reminders_neglected\n ),\n\n gentle_reminders_data AS (\n
SELECT jsonb_agg(\n    jsonb_build_object(\n      'type', r.type,\n      'priority', r.priority,\n
'child_id', r.child_id,\n      'child_name', r.child_name,\n      'message', r.message,\n
'action_label', r.action_label,\n      'action_route', r.action_route,\n      'metadata',
r.metadata\n    )\n    ORDER BY r.priority, r.child_name\n   ) AS data\n   FROM (\n    SELECT
* FROM all_reminders\n    ORDER BY priority, child_name\n    LIMIT 5\n   ) r\n ),\n\n
coming_up_next_data AS (\n   SELECT jsonb_agg(\n    jsonb_build_object(\n
'planned_session_id', ps.id,\n      'child_id', ps.child_id,\n      'child_name', pc.child_name,\n
'subject_id', ps.subject_id,\n      'subject_name', s.subject_name,\n      'subject_color',
COALESCE(s.color, '#6B7280'),\n      'topic_name', t.topic_name,\n      'session_date',
ps.session_date,\n      'session_duration_minutes', ps.session_duration_minutes,\n
'status', ps.status\n    )\n    ORDER BY ps.session_date, ps.session_index\n   ) AS data\n
FROM (\n    SELECT *\n    FROM planned_sessions\n    WHERE child_id IN (SELECT
```

```
child_id FROM parent_children)\n      AND status = 'planned'\n      AND session_date >=
CURRENT_DATE\n    ORDER BY session_date, session_index\n    LIMIT 5\n   ) ps\n   JOIN
parent_children pc ON pc.child_id = ps.child_id\n   JOIN subjects s ON s.id = ps.subject_id\n
LEFT JOIN topics t ON t.id = ps.topic_ids[1]\n ),\n\n subject_coverage_data AS (\n   SELECT
jsonb_agg(\n    jsonb_build_object(\n      'child_id', coverage.child_id,\n      'child_name',
coverage.child_name,\n      'subject_id', coverage.subject_id,\n      'subject_name',
coverage.subject_name,\n      'subject_color', coverage.color,\n      'subject_icon',
coverage.icon,\n      'sessions_completed', coverage.sessions_completed,\n
'topics_covered', coverage.topics_covered\n    )\n    ORDER BY coverage.child_name,
coverage.subject_name\n   ) AS data\n   FROM (\n    SELECT \n      pc.child_id,\n
pc.child_name,\n      csd.subject_id,\n      csd.subject_name,\n      csd.color,\n
csd.icon,\n      COUNT(DISTINCT ps.id) FILTER (WHERE ps.status = 'completed') AS
sessions_completed,\n      COUNT(DISTINCT topic_id) AS topics_covered\n    FROM
parent_children pc\n    JOIN child_subject_details csd ON csd.child_id = pc.child_id\n
LEFT JOIN planned_sessions ps ON ps.child_id = pc.child_id \n      AND ps.subject_id =
csd.subject_id\n      AND ps.session_date BETWEEN v_week_start AND v_week_end\n
AND ps.status = 'completed'\n    LEFT JOIN LATERAL unnest(ps.topic_ids) AS topic_id ON
true\n    GROUP BY pc.child_id, pc.child_name, csd.subject_id, csd.subject_name,
csd.color, csd.icon\n   ) coverage\n   WHERE coverage.sessions_completed > 0 OR
coverage.topics_covered > 0\n )\n\n SELECT jsonb_build_object(\n   'children',
COALESCE((SELECT data FROM children_data), '[]'::jsonb),\n   'week_summary',
COALESCE((SELECT data FROM week_summary_data), '{}'::jsonb),\n   'daily_pattern',
COALESCE((SELECT data FROM daily_pattern_data), '[]'::jsonb),\n   'gentle_reminders',
COALESCE((SELECT data FROM gentle_reminders_data), '[]'::jsonb),\n   'coming_up_next',
COALESCE((SELECT data FROM coming_up_next_data), '[]'::jsonb),\n   'subject_coverage',
COALESCE((SELECT data FROM subject_coverage_data), '[]'::jsonb)\n ) INTO v_result;\n\n
RETURN v_result;\nEND;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_get_plan_coverage_overview",
  "arguments": "p_child_id uuid",
  "definition": "CREATE OR REPLACE FUNCTION
public.rpc_get_plan_coverage_overview(p_child_id uuid)\n RETURNS jsonb\n LANGUAGE
plpgsql\n STABLE SECURITY DEFINER\n SET search_path TO 'public'\nAS
$function$\nDECLARE\n v_revision_period_end date;\n v_days_remaining integer;\n
v_weeks_remaining numeric;\n v_total_planned integer;\n v_total_completed integer;\n
v_total_remaining integer;\n v_total_minutes integer;\n v_subjects jsonb;\nBEGIN\n -- Get
revision period end date\n SELECT rp.end_date\n INTO v_revision_period_end\n FROM
revision_periods rp\n WHERE rp.child_id = p_child_id AND rp.is_active = true\n LIMIT 1;\n\n
-- Fallback to revision_plans if no revision_period\n IF v_revision_period_end IS NULL
THEN\n   SELECT rp.end_date\n   INTO v_revision_period_end\n   FROM revision_plans rp\n
WHERE rp.child_id = p_child_id AND rp.status = 'active'\n   LIMIT 1;\n END IF;\n\n --
Calculate time remaining\n IF v_revision_period_end IS NOT NULL THEN\n
v_days_remaining := GREATEST(0, v_revision_period_end - CURRENT_DATE);\n
v_weeks_remaining := GREATEST(0, v_days_remaining / 7.0);\n END IF;\n\n -- Get totals
from actual planned_sessions\n SELECT \n   COALESCE(COUNT(*), 0),\n
```

COALESCE(COUNT(*) FILTER (WHERE status = 'completed'), 0),\n COALESCE(COUNT(*) FILTER (WHERE status IN ('planned', 'started')), 0),\n COALESCE(SUM(session_duration_minutes), 0)\n INTO v_total_planned, v_total_completed, v_total_remaining, v_total_minutes\n FROM planned_sessions\n WHERE child_id = p_child_id\n AND status != 'skipped';\n\n -- Build subjects array using CTE\n WITH subject_stats AS (\n SELECT \n   ps.subject_id,\n   s.subject_name,\n COALESCE(s.color, '#6B7280') as color,\n   COALESCE(s.icon, 'book') as icon,\n COUNT(*) as planned_sessions,\n   COUNT(*) FILTER (WHERE ps.status = 'completed') as completed_sessions,\n   COUNT(*) FILTER (WHERE ps.status IN ('planned', 'started')) as remaining_sessions,\n   SUM(ps.session_duration_minutes) as total_minutes\n FROM planned_sessions ps\n JOIN subjects s ON s.id = ps.subject_id\n WHERE ps.child_id = p_child_id\n   AND ps.status != 'skipped'\n GROUP BY ps.subject_id, s.subject_name, s.color, s.icon\n)\n SELECT COALESCE(jsonb_agg(\n jsonb_build_object(\n 'subject_id', ss.subject_id,\n   'subject_name', ss.subject_name,\n   'color', ss.color,\n 'icon', ss.icon,\n   'planned_sessions', ss.planned_sessions,\n   'completed_sessions', ss.completed_sessions,\n   'remaining_sessions', ss.remaining_sessions,\n 'total_minutes', ss.total_minutes,\n   'completion_percent', CASE \n   WHEN ss.planned_sessions > 0 \n   THEN ROUND((ss.completed_sessions::numeric / ss.planned_sessions) * 100)\n   ELSE 0 \n   END\n)\n ORDER BY ss.subject_name\n), '[]'::jsonb)\n INTO v_subjects\n FROM subject_stats ss;\n\n -- Return result\n RETURN jsonb_build_object(\n 'child_id', p_child_id,\n \n 'revision_period', jsonb_build_object(\n 'end_date', v_revision_period_end,\n   'days_remaining', v_days_remaining,\n 'weeks_remaining', ROUND(v_weeks_remaining, 1)\n),\n \n 'totals', jsonb_build_object(\n   'planned_sessions', v_total_planned,\n   'completed_sessions', v_total_completed,\n   'remaining_sessions', v_total_remaining,\n   'total_minutes', v_total_minutes,\n   'total_hours', ROUND(v_total_minutes / 60.0, 1),\n 'completion_percent', CASE \n   WHEN v_total_planned > 0 \n   THEN ROUND((v_total_completed::numeric / v_total_planned) * 100)\n   ELSE 0 \n   END\n),\n \n 'subjects', v_subjects,\n \n 'status', CASE\n   WHEN v_total_planned = 0 THEN 'no_plan'\n   WHEN v_total_completed >= v_total_planned THEN 'complete'\n   WHEN v_weeks_remaining IS NOT NULL AND v_weeks_remaining > 0 AND \n (v_total_remaining / v_weeks_remaining) <= 15 THEN 'on_track'\n   WHEN v_weeks_remaining IS NOT NULL AND v_weeks_remaining > 0 AND \n (v_total_remaining / v_weeks_remaining) <= 25 THEN 'manageable'\n   ELSE 'intensive'\n END,\n \n 'pace', CASE\n   WHEN v_weeks_remaining IS NOT NULL AND v_weeks_remaining > 0 \n   THEN jsonb_build_object(\n   'sessions_per_week_needed', CEIL(v_total_remaining / v_weeks_remaining),\n   'hours_per_week_needed', ROUND((v_total_minutes - (v_total_completed * 30)) / v_weeks_remaining / 60.0, 1)\n   )\n ELSE NULL\n END\n);\nEND;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_get_planned_session_overview",
  "arguments": "p_planned_session_id uuid",
  "definition": "CREATE OR REPLACE FUNCTION public.rpc_get_planned_session_overview(p_planned_session_id uuid)\n RETURNS TABLE(planned_session_id uuid, child_id uuid, plan_id uuid, session_date date, day_of_week

text, session_pattern session_pattern, session_duration_minutes integer, session_index integer, status planned_session_status, started_at timestamp with time zone, completed_at timestamp with time zone, subject_id uuid, subject_name text, topic_ids uuid[], primary_topic_id uuid, topic_name text, theme_id uuid, theme_name text, component_id uuid, component_name text)\n LANGUAGE sql\n STABLE\nAS $function$\n select\n   ps.id as planned_session_id,\n   ps.child_id,\n   ps.plan_id,\n\n   ps.session_date,\n   ps.day_of_week,\n   ps.session_pattern,\n   ps.session_duration_minutes,\n   ps.session_index,\n\n   ps.status,\n   ps.started_at,\n   ps.completed_at,\n\n   ps.subject_id,\n   s.subject_name,\n\n   ps.topic_ids,\n   ps.topic_ids[1] as primary_topic_id,\n   t.topic_name,\n\n   th.id as theme_id,\n   th.theme_name,\n\n   c.id as component_id,\n   c.component_name\n\n from public.planned_sessions ps\n join public.subjects s on s.id = ps.subject_id\n\n left join public.topics t\n   on t.id = ps.topic_ids[1]\n\n left join public.themes th\n   on th.id = t.theme_id\n\n left join public.components c\n   on c.id = th.component_id\n\n where ps.id = p_planned_session_id;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_get_subject_pathways",
  "arguments": "p_subject_ids uuid[]",
  "definition": "CREATE OR REPLACE FUNCTION public.rpc_get_subject_pathways(p_subject_ids uuid[])\n RETURNS jsonb\n LANGUAGE plpgsql\n SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\nDECLARE\n v_result jsonb;\nBEGIN\n SELECT jsonb_agg(subject_pathways)\n INTO v_result\n FROM (\n   SELECT jsonb_build_object(\n   'subject_id', s.id,\n   'subject_name', s.subject_name,\n   'requires_pathway_selection', s.requires_pathway_selection,\n   'pathways', COALESCE(\n   (\n   SELECT jsonb_agg(\n   jsonb_build_object(\n   'id', ep.id,\n   'pathway_code', ep.pathway_code,\n   'pathway_name', ep.pathway_name,\n   'parent_pathway_id', ep.parent_pathway_id,\n   'is_required_choice', ep.is_required_choice,\n   'display_order', ep.display_order\n   ) ORDER BY ep.display_order\n   )\n   FROM exam_pathways ep\n   WHERE ep.subject_id = s.id\n   ),\n   '[]'::jsonb\n   )\n   ) as subject_pathways\n   FROM subjects s\n   WHERE s.id = ANY(p_subject_ids)\n   AND s.requires_pathway_selection = true\n) sub;\n\n RETURN COALESCE(v_result, '[]'::jsonb);\nEND;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_get_subject_progress",
  "arguments": "p_parent_id uuid, p_child_id uuid",
  "definition": "CREATE OR REPLACE FUNCTION public.rpc_get_subject_progress(p_parent_id uuid, p_child_id uuid DEFAULT NULL::uuid)\n RETURNS jsonb\n LANGUAGE plpgsql\n STABLE SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\nDECLARE\n v_child_id uuid;\n v_result jsonb;\nBEGIN\n -- Determine which child to show\n -- If p_child_id provided, verify it belongs to parent\n -- Otherwise, get first child\n IF p_child_id IS NOT NULL THEN\n   SELECT c.id INTO v_child_id\n   FROM children c\n   WHERE c.id = p_child_id AND c.parent_id = p_parent_id;\n ELSE\n   SELECT c.id INTO v_child_id\n   FROM children c\n   WHERE c.parent_id =

```
p_parent_id\n    ORDER BY c.created_at ASC\n    LIMIT 1;\n  END IF;\n\n  -- If no child found, return empty structure\n  IF v_child_id IS NULL THEN\n    RETURN jsonb_build_object(\n      'child', NULL,\n      'overview', NULL,\n      'subjects', '[]'::jsonb,\n      'timeline', '[]'::jsonb,\n      'focus_areas', '[]'::jsonb,\n      'suggestions', '[]'::jsonb\n    );\n  END IF;\n\n  WITH\n  -- ============================================================================\n  -- CHILD INFO\n  -- ============================================================================\n  child_info AS (\n    SELECT\n      c.id AS child_id,\n      COALESCE(c.preferred_name, c.first_name) AS child_name,\n      c.year_group,\n      COALESCE(\n        (SELECT et.name FROM child_subjects cs\n         JOIN subjects s ON s.id = cs.subject_id\n         LEFT JOIN exam_types et ON et.id = s.exam_type_id\n         WHERE cs.child_id = c.id LIMIT 1),\n        'GCSE'\n      ) AS exam_type,\n      (SELECT COUNT(DISTINCT cs.subject_id) FROM child_subjects cs WHERE cs.child_id = c.id) AS active_subjects_count,\n      (SELECT COUNT(*) FROM planned_sessions ps \n       WHERE ps.child_id = c.id \n       AND ps.session_date BETWEEN date_trunc('week', CURRENT_DATE)::date AND (date_trunc('week', CURRENT_DATE) + INTERVAL '6 days')::date\n       AND ps.status = 'completed') AS sessions_this_week,\n      (SELECT COUNT(DISTINCT topic_id) FROM planned_sessions ps\n       CROSS JOIN LATERAL unnest(ps.topic_ids) AS topic_id\n       WHERE ps.child_id = c.id \n       AND ps.session_date BETWEEN date_trunc('week', CURRENT_DATE)::date AND (date_trunc('week', CURRENT_DATE) + INTERVAL '6 days')::date\n       AND ps.status = 'completed') AS topics_covered_this_week\n    FROM children c\n    WHERE c.id = v_child_id\n  ),\n\n  -- ============================================================================\n  -- SUBJECT DETAILS with topics covered/remaining\n  -- ============================================================================\n  subject_stats AS (\n    SELECT\n      cs.subject_id,\n      s.subject_name,\n      COALESCE(s.color, '#7C3AED') AS subject_color,\n      COALESCE(s.icon, 'book') AS subject_icon,\n      eb.name AS exam_board_name,  -- FIXED: was eb.board_name\n      et.name AS exam_type,\n      -- Count total topics for this subject\n      (SELECT COUNT(DISTINCT t.id) \n       FROM topics t \n       JOIN themes th ON th.id = t.theme_id\n       JOIN components comp ON comp.id = th.component_id\n       WHERE comp.subject_id = cs.subject_id) AS total_topics,\n      -- Count distinct topics covered (ever)\n      (SELECT COUNT(DISTINCT topic_id)\n       FROM planned_sessions ps\n       CROSS JOIN LATERAL unnest(ps.topic_ids) AS topic_id\n       WHERE ps.child_id = v_child_id\n       AND ps.subject_id = cs.subject_id\n       AND ps.status = 'completed') AS topics_covered_total,\n      -- Last session date for this subject\n      (SELECT MAX(ps.session_date)\n       FROM planned_sessions ps\n       WHERE ps.child_id = v_child_id\n       AND ps.subject_id = cs.subject_id\n       AND ps.status = 'completed') AS last_session_date\n    FROM child_subjects cs\n    JOIN subjects s ON s.id = cs.subject_id\n    LEFT JOIN exam_boards eb ON eb.id = s.exam_board_id\n    LEFT JOIN exam_types et ON et.id = s.exam_type_id\n    WHERE cs.child_id = v_child_id\n  ),\n\n  -- ============================================================================\n  -- RECENTLY COVERED TOPICS (last 14 days)\n  -- ============================================================================\n  recently_covered AS (\n    SELECT\n      ps.subject_id,\n      topic_id,\n      t.topic_name,\n      th.theme_name,\n      COUNT(DISTINCT ps.id) AS session_count,\n      MAX(ps.session_date) AS last_covered_date,\n      (CURRENT_DATE -
```

```
MAX(ps.session_date))::int AS days_since,\n     COUNT(DISTINCT ps.id) > 1 AS
was_revisited,\n     -- Get most recent confidence level if available\n     (SELECT
rs.confidence_level \n      FROM revision_sessions rs \n      WHERE rs.child_id = v_child_id
AND rs.topic_id = topic_id\n      ORDER BY rs.session_date DESC LIMIT 1) AS
confidence_level\n    FROM planned_sessions ps\n    CROSS JOIN LATERAL
unnest(ps.topic_ids) AS topic_id\n    JOIN topics t ON t.id = topic_id\n    JOIN themes th ON
th.id = t.theme_id\n    WHERE ps.child_id = v_child_id\n     AND ps.status = 'completed'\n
AND ps.session_date >= CURRENT_DATE - INTERVAL '14 days'\n    GROUP BY
ps.subject_id, topic_id, t.topic_name, th.theme_name\n  ),\n\n  --
==========================================================================
===\n  -- COMING UP TOPICS (next 14 days)\n  --
==========================================================================
===\n  coming_up AS (\n    SELECT\n     ps.subject_id,\n     topic_id,\n     t.topic_name,\n
th.theme_name,\n     MIN(ps.session_date) AS session_date,\n     (MIN(ps.session_date) -
CURRENT_DATE)::int AS days_until,\n     MIN(ps.session_date) = CURRENT_DATE + 1 AS
is_tomorrow\n    FROM planned_sessions ps\n    CROSS JOIN LATERAL unnest(ps.topic_ids)
AS topic_id\n    JOIN topics t ON t.id = topic_id\n    JOIN themes th ON th.id = t.theme_id\n
WHERE ps.child_id = v_child_id\n     AND ps.status = 'planned'\n     AND ps.session_date >=
CURRENT_DATE\n     AND ps.session_date <= CURRENT_DATE + INTERVAL '14 days'\n
GROUP BY ps.subject_id, topic_id, t.topic_name, th.theme_name\n  ),\n\n  --
==========================================================================
===\n  -- BUILD SUBJECTS ARRAY with nested recently_covered and coming_up\n  --
==========================================================================
===\n  subjects_data AS (\n    SELECT jsonb_agg(\n     jsonb_build_object(\n      'subject_id',
ss.subject_id,\n      'subject_name', ss.subject_name,\n      'subject_color',
ss.subject_color,\n      'subject_icon', ss.subject_icon,\n      'exam_board_name',
ss.exam_board_name,\n      'exam_type', ss.exam_type,\n      'status', CASE \n       WHEN
ss.last_session_date IS NULL THEN 'not_started'\n       WHEN ss.last_session_date <
CURRENT_DATE - INTERVAL '14 days' THEN 'needs_attention'\n       ELSE 'in_progress'\n
END,\n      'topics_covered_total', COALESCE(ss.topics_covered_total, 0),\n
'topics_remaining', GREATEST(0, COALESCE(ss.total_topics, 0) -
COALESCE(ss.topics_covered_total, 0)),\n      'completion_percentage', CASE \n       WHEN
COALESCE(ss.total_topics, 0) = 0 THEN 0\n       ELSE LEAST(100,
ROUND((COALESCE(ss.topics_covered_total, 0)::numeric / ss.total_topics::numeric) *
100))\n      END,\n      'recently_covered', COALESCE(\n       (SELECT jsonb_agg(\n
jsonb_build_object(\n          'topic_id', rc.topic_id,\n          'topic_name', rc.topic_name,\n
'theme_name', rc.theme_name,\n          'session_count', rc.session_count,\n
'last_covered_date', rc.last_covered_date,\n          'days_since', rc.days_since,\n
'was_revisited', rc.was_revisited,\n          'confidence_level',
COALESCE(rc.confidence_level, 'on_track')\n        ) ORDER BY rc.last_covered_date
DESC\n       ) FROM recently_covered rc WHERE rc.subject_id = ss.subject_id),\n
'[]'::jsonb\n      ),\n      'coming_up', COALESCE(\n       (SELECT jsonb_agg(\n
jsonb_build_object(\n          'topic_id', cu.topic_id,\n          'topic_name', cu.topic_name,\n
'theme_name', cu.theme_name,\n          'session_date', cu.session_date,\n
'days_until', cu.days_until,\n          'is_tomorrow', cu.is_tomorrow\n        ) ORDER BY
cu.session_date ASC\n       ) FROM coming_up cu WHERE cu.subject_id = ss.subject_id),\n
'[]'::jsonb\n      )\n    ) ORDER BY ss.subject_name\n  ) AS data\n  FROM subject_stats ss\n
```

```
),\n\n  --
==========================================================================
===\n  -- OVERVIEW STATS\n  --
==========================================================================
===\n  overview_data AS (\n    SELECT jsonb_build_object(\n      'coverage_status', CASE\n
WHEN EXISTS (\n        SELECT 1 FROM subject_stats ss \n        WHERE ss.last_session_date
IS NULL OR ss.last_session_date < CURRENT_DATE - INTERVAL '14 days'\n      ) THEN
'needs_attention'\n      ELSE 'on_track'\n    END,\n    'coverage_message', CASE\n
WHEN EXISTS (\n        SELECT 1 FROM subject_stats ss \n        WHERE ss.last_session_date
IS NULL OR ss.last_session_date < CURRENT_DATE - INTERVAL '14 days'\n      ) THEN
'Some subjects need attention'\n      ELSE 'All subjects progressing as planned'\n    END,\n
'topics_revisited_count', (SELECT COUNT(*) FROM recently_covered WHERE was_revisited
= true),\n    'next_week_topics_count', (\n      SELECT COUNT(DISTINCT topic_id) \n
FROM coming_up \n      WHERE session_date <= CURRENT_DATE + INTERVAL '7 days'\n
),\n    'next_week_subjects_count', (\n      SELECT COUNT(DISTINCT subject_id) \n
FROM coming_up \n      WHERE session_date <= CURRENT_DATE + INTERVAL '7 days'\n
)\n    ) AS data\n  ),\n\n  --
==========================================================================
===\n  -- TIMELINE (grouped by relative date)\n  --
==========================================================================
===\n  timeline_sessions AS (\n    SELECT\n      ps.id AS planned_session_id,\n
ps.session_date,\n      ps.subject_id,\n      s.subject_name,\n      COALESCE(s.color,
'#7C3AED') AS subject_color,\n      t.topic_name,\n      CASE\n        WHEN ps.session_date =
CURRENT_DATE THEN 'Today'\n        WHEN ps.session_date = CURRENT_DATE + 1 THEN
'Tomorrow'\n        WHEN ps.session_date <= CURRENT_DATE + INTERVAL '3 days' THEN 'In '
|| (ps.session_date - CURRENT_DATE)::int || ' days'\n        WHEN ps.session_date <=
CURRENT_DATE + INTERVAL '7 days' THEN 'This week'\n        ELSE 'Next week'\n      END AS
group_label,\n      (ps.session_date - CURRENT_DATE)::int AS days_until\n    FROM
planned_sessions ps\n    JOIN subjects s ON s.id = ps.subject_id\n    LEFT JOIN topics t ON
t.id = ps.topic_ids[1]\n    WHERE ps.child_id = v_child_id\n      AND ps.status = 'planned'\n
AND ps.session_date >= CURRENT_DATE\n      AND ps.session_date <= CURRENT_DATE +
INTERVAL '14 days'\n    ORDER BY ps.session_date, ps.session_index\n  ),\n\n  timeline_data
AS (\n    SELECT jsonb_agg(\n      jsonb_build_object(\n        'group_label', tg.group_label,\n
'group_date', tg.min_date,\n        'days_until', tg.min_days,\n        'sessions', tg.sessions\n      )
ORDER BY tg.min_days\n    ) AS data\n    FROM (\n      SELECT\n        ts.group_label,\n
MIN(ts.session_date) AS min_date,\n        MIN(ts.days_until) AS min_days,\n        jsonb_agg(\n
jsonb_build_object(\n          'planned_session_id', ts.planned_session_id,\n          'subject_id',
ts.subject_id,\n          'subject_name', ts.subject_name,\n          'subject_color',
ts.subject_color,\n          'topic_name', ts.topic_name\n        ) ORDER BY ts.session_date\n
) AS sessions\n      FROM timeline_sessions ts\n      GROUP BY ts.group_label\n    ) tg\n  ),\n\n
--
==========================================================================
===\n  -- FOCUS AREAS (current topics being worked on)\n  --
==========================================================================
===\n  focus_areas_data AS (\n    SELECT jsonb_agg(\n      jsonb_build_object(\n
'subject_id', fa.subject_id,\n        'subject_name', fa.subject_name,\n        'subject_color',
fa.subject_color,\n        'subject_icon', fa.subject_icon,\n        'focus_topics', fa.focus_topics\n
```

)\n) AS data\n FROM (\n   SELECT\n   ss.subject_id,\n   ss.subject_name,\n   ss.subject_color,\n   ss.subject_icon,\n   (SELECT string_agg(cu.topic_name, ' and ' ORDER BY cu.session_date)\n   FROM (SELECT topic_name, session_date FROM coming_up WHERE subject_id = ss.subject_id LIMIT 2) cu\n   ) AS focus_topics\n   FROM subject_stats ss\n   WHERE EXISTS (SELECT 1 FROM coming_up cu WHERE cu.subject_id = ss.subject_id)\n) fa\n),\n\n -- ==\n -- SUGGESTIONS (topics that may need review)\n -- ==\n suggestions_data AS (\n   SELECT jsonb_agg(\n   jsonb_build_object(\n   'type', sg.type,\n   'priority', sg.priority,\n   'title', sg.title,\n   'message', sg.message,\n   'action_label', sg.action_label,\n   'subject_id', sg.subject_id,\n   'topic_id', sg.topic_id\n   ) ORDER BY sg.priority\n   ) AS data\n   FROM (\n   -- Topics covered 21+ days ago, not scheduled again\n   SELECT\n   'review_recommended'::text AS type,\n   1 AS priority,\n   'Consider an extra review session' AS title,\n   rc.topic_name || ' was covered ' || rc.days_since || ' days ago and might benefit from another look.' AS message,\n   'Add Review Session' AS action_label,\n   rc.subject_id,\n   rc.topic_id\n   FROM (\n   SELECT\n   ps.subject_id,\n   topic_id,\n   t.topic_name,\n   (CURRENT_DATE - MAX(ps.session_date))::int AS days_since\n   FROM planned_sessions ps\n   CROSS JOIN LATERAL unnest(ps.topic_ids) AS topic_id\n   JOIN topics t ON t.id = topic_id\n   WHERE ps.child_id = v_child_id\n   AND ps.status = 'completed'\n   GROUP BY ps.subject_id, topic_id, t.topic_name\n   HAVING (CURRENT_DATE - MAX(ps.session_date))::int >= 21\n   ) rc\n   WHERE NOT EXISTS (\n   SELECT 1 FROM coming_up cu WHERE cu.topic_id = rc.topic_id\n   )\n   LIMIT 3\n   ) sg\n)\n\n -- ==\n -- FINAL RESULT\n -- ==\n SELECT jsonb_build_object(\n   'child', (\n   SELECT jsonb_build_object(\n   'child_id', ci.child_id,\n   'child_name', ci.child_name,\n   'year_group', ci.year_group,\n   'exam_type', ci.exam_type,\n   'active_subjects_count', ci.active_subjects_count,\n   'sessions_this_week', ci.sessions_this_week,\n   'topics_covered_this_week', ci.topics_covered_this_week\n   ) FROM child_info ci\n   ),\n   'overview', (SELECT data FROM overview_data),\n   'subjects', COALESCE((SELECT data FROM subjects_data), '[]'::jsonb),\n   'timeline', COALESCE((SELECT data FROM timeline_data), '[]'::jsonb),\n   'focus_areas', COALESCE((SELECT data FROM focus_areas_data), '[]'::jsonb),\n   'suggestions', COALESCE((SELECT data FROM suggestions_data), '[]'::jsonb)\n) INTO v_result;\n\n RETURN v_result;\nEND;\n$function$\n"
 },
 {
   "schema_name": "public",
   "function_name": "rpc_get_todays_planned_sessions",
   "arguments": "p_child_id uuid, p_date date",
   "definition": "CREATE OR REPLACE FUNCTION public.rpc_get_todays_planned_sessions(p_child_id uuid, p_date date DEFAULT CURRENT_DATE)\n RETURNS TABLE(planned_session_id uuid, child_id uuid, session_date date, session_index integer, status planned_session_status, subject_id uuid, subject_name text, topic_ids uuid[], primary_topic_id uuid, topic_name text, component_name text,

```
theme_name text)\n LANGUAGE sql\n STABLE\nAS $function$\n  select\n    ps.id as planned_session_id,\n    ps.child_id,\n\n    ps.session_date,\n    ps.session_index,\n ps.status,\n\n    ps.subject_id,\n    s.subject_name,\n\n    ps.topic_ids,\n    ps.topic_ids[1] as primary_topic_id,\n    t.topic_name,\n\n    c.component_name,\n    th.theme_name\n\n  from public.planned_sessions ps\n  join public.subjects s on s.id = ps.subject_id\n\n  left join public.topics t\n    on t.id = ps.topic_ids[1]\n\n  left join public.themes th\n    on th.id = t.theme_id\n\n  left join public.components c\n    on c.id = th.component_id\n\n  where ps.child_id = p_child_id\n    and ps.session_date = p_date\n\n  order by ps.session_index asc;\n$function$\n"
 },
 {
   "schema_name": "public",
   "function_name": "rpc_get_todays_sessions",
   "arguments": "p_child_id uuid, p_session_date date",
   "definition": "CREATE OR REPLACE FUNCTION public.rpc_get_todays_sessions(p_child_id uuid, p_session_date date DEFAULT CURRENT_DATE)\n RETURNS TABLE(planned_session_id uuid, session_date date, session_index integer, session_pattern text, session_duration_minutes integer, status text, subject_id uuid, subject_name text, icon text, color text, topic_count integer, topics_preview jsonb)\n LANGUAGE plpgsql\n SECURITY DEFINER\nAS $function$\nBEGIN\n  RETURN QUERY\n  SELECT\n    ps.id AS planned_session_id,\n    ps.session_date,\n    ROW_NUMBER() OVER (\n      PARTITION BY ps.session_date\n      ORDER BY ps.created_at\n    )::integer AS session_index,\n    ps.session_pattern::text AS session_pattern,\n    ps.session_duration_minutes,\n    ps.status::text AS status,\n    ps.subject_id,\n    COALESCE(s.subject_name, 'General Revision') AS subject_name,\n    COALESCE(s.icon, 'book') AS icon,\n    COALESCE(s.color, '#6B7280') AS color,\n    COALESCE(array_length(ps.topic_ids, 1), 0) AS topic_count,\n    COALESCE(\n      (\n        SELECT jsonb_agg(\n          jsonb_build_object(\n            'id', t.id,\n            'topic_name', t.topic_name,\n            'order_index', idx.ord\n          )\n          ORDER BY idx.ord\n        )\n        FROM unnest(ps.topic_ids)\n          WITH ORDINALITY AS idx(topic_id, ord)\n        JOIN topics t\n          ON t.id = idx.topic_id\n        WHERE idx.ord <= 3\n      ),\n      '[]'::jsonb\n    ) AS topics_preview\n  FROM planned_sessions ps\n  LEFT JOIN subjects s\n    ON s.id = ps.subject_id\n  WHERE ps.child_id = p_child_id\n    AND ps.session_date = p_session_date\n  ORDER BY ps.created_at;\nEND;\n$function$\n"
 },
 {
   "schema_name": "public",
   "function_name": "rpc_get_week_plan",
   "arguments": "p_child_id uuid, p_week_start_date date",
   "definition": "CREATE OR REPLACE FUNCTION public.rpc_get_week_plan(p_child_id uuid, p_week_start_date date)\n RETURNS TABLE(day_date date, sessions jsonb)\n LANGUAGE plpgsql\nAS $function$\nbegin\n  return query\n  with days as (\n    select generate_series(\n      p_week_start_date,\n      p_week_start_date + interval '6 days',\n      interval '1 day'\n    )::date as day_date\n  ),\n\n  numbered_sessions as (\n    select\n      ps.id,\n      ps.child_id,\n      ps.plan_id,\n      ps.session_date,\n      ps.session_pattern,\n      ps.session_duration_minutes,\n      ps.status,\n      ps.subject_id,\n      ps.topic_ids,\n      ps.created_at,\n      row_number() over (\n        partition by ps.session_date::date\n        order by ps.created_at\n      ) as computed_session_index\n    from public.planned_sessions ps\n
```

```
where ps.child_id = p_child_id\n    and ps.session_date::date between p_week_start_date\n and (p_week_start_date + interval '6 days')::date\n  ),\n\n  per_day as (\n    select\n ns.session_date::date as day_date,\n    jsonb_agg(\n      jsonb_build_object(\n 'planned_session_id', ns.id,\n        'session_date', ns.session_date,\n        'session_index', ns.computed_session_index,\n        'session_pattern', ns.session_pattern::text,\n 'session_duration_minutes', ns.session_duration_minutes,\n        'status', ns.status::text,\n 'subject_id', ns.subject_id,\n        'subject_name', coalesce(s.subject_name, 'General Revision'),\n        'icon', coalesce(s.icon, 'book'),\n        'color', coalesce(s.color, '#6B7280'),\n        'topic_count', coalesce(array_length(ns.topic_ids, 1), 0),\n 'topics_preview',\n          coalesce(\n            (\n              select jsonb_agg(\n jsonb_build_object(\n                  'id', t.id,\n                  'topic_name', t.topic_name,\n 'order_index', idx.ord\n                )\n              order by idx.ord\n            )\n            from unnest(ns.topic_ids)\n              with ordinality as idx(topic_id, ord)\n              join public.topics t\n                on t.id = idx.topic_id\n              where idx.ord <= 3\n            ),\n            '[]'::jsonb\n          )\n      )\n      order by ns.computed_session_index\n    ) as sessions\n  from numbered_sessions ns\n  left join public.subjects s\n    on s.id = ns.subject_id\n  group by ns.session_date::date\n  )\n\n  select\n  d.day_date,\n  coalesce(p.sessions, '[]'::jsonb) as sessions\n  from days d\n  left join per_day p\n    on p.day_date = d.day_date\n  order by d.day_date;\n\nend;\n$function$\n"
 },
 {
   "schema_name": "public",
   "function_name": "rpc_list_exam_types",
   "arguments": "",
   "definition": "CREATE OR REPLACE FUNCTION public.rpc_list_exam_types()\n RETURNS TABLE(id uuid, name text, code text, sort_order integer)\n LANGUAGE sql\n STABLE SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\n  select\n  et.id,\n  et.name,\n  et.code,\n  et.sort_order\n  from public.exam_types et\n  order by et.sort_order asc, et.name asc;\n$function$\n"
 },
 {
   "schema_name": "public",
   "function_name": "rpc_list_goals",
   "arguments": "",
   "definition": "CREATE OR REPLACE FUNCTION public.rpc_list_goals()\n RETURNS TABLE(id uuid, code text, name text, description text, sort_order integer)\n LANGUAGE sql\n STABLE SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\n  select\n  g.id,\n  g.code,\n  g.name,\n  g.description,\n  g.sort_order\n  from public.goals g\n  order by g.sort_order asc, g.name asc;\n$function$\n"
 },
 {
   "schema_name": "public",
   "function_name": "rpc_list_need_areas",
   "arguments": "",
   "definition": "CREATE OR REPLACE FUNCTION public.rpc_list_need_areas()\n RETURNS TABLE(code jcq_area, name text, description text, helper_text text, is_jcq_recognised boolean, sort_order integer)\n LANGUAGE sql\n STABLE SECURITY DEFINER\n SET
```

search_path TO 'public'\nAS $function$\n SELECT \n code,\n name,\n description,\n helper_text,\n is_jcq_recognised,\n sort_order\n FROM public.need_areas\n ORDER BY sort_order ASC;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_list_need_clusters",
  "arguments": "",
  "definition": "CREATE OR REPLACE FUNCTION public.rpc_list_need_clusters()\n RETURNS TABLE(code text, name text, description text, jcq_area jcq_area, jcq_area_name text, condition_name text, parent_friendly_name text, typical_behaviours text[], example_signs text[], typically_has_accommodations boolean, common_arrangements text[], sort_order integer)\n LANGUAGE sql\n STABLE SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\n SELECT \n nc.code,\n nc.name,\n nc.description,\n nc.jcq_area,\n na.name AS jcq_area_name,\n nc.condition_name,\n nc.parent_friendly_name,\n nc.typical_behaviours,\n nc.example_signs,\n COALESCE(nc.typically_has_accommodations, false),\n nc.common_arrangements,\n COALESCE(nc.sort_order, 9999) AS sort_order\n FROM public.need_clusters nc\n LEFT JOIN public.need_areas na ON na.code = nc.jcq_area\n WHERE nc.is_active = true\n ORDER BY \n COALESCE(na.sort_order, 99),\n COALESCE(nc.sort_order, 9999) ASC,\n nc.name ASC;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_list_subject_groups_for_exam_types",
  "arguments": "p_exam_type_ids uuid[]",
  "definition": "CREATE OR REPLACE FUNCTION public.rpc_list_subject_groups_for_exam_types(p_exam_type_ids uuid[])\n RETURNS TABLE(exam_type_id uuid, subject_name text, icon text, color text, boards jsonb)\n LANGUAGE sql\n STABLE\nAS $function$\n with base as (\n select\n s.exam_type_id,\n s.subject_name,\n s.exam_board_id,\n eb.name as exam_board_name,\n\n -- pick one subject row deterministically for this (subject_name + board)\n (array_agg(s.id order by s.code asc, s.created_at asc, s.id asc))[1] as subject_id,\n\n -- consistent display fields\n (array_agg(s.icon order by s.code asc, s.created_at asc, s.id asc))[1] as icon,\n (array_agg(s.color order by s.code asc, s.created_at asc, s.id asc))[1] as color\n from public.subjects s\n join public.exam_boards eb on eb.id = s.exam_board_id\n where s.exam_type_id = any(p_exam_type_ids)\n group by\n s.exam_type_id,\n s.subject_name,\n s.exam_board_id,\n eb.name\n)\n select\n b.exam_type_id,\n b.subject_name,\n\n -- keep icon/color stable for the subject card\n (array_agg(b.icon order by b.exam_board_name asc, b.exam_board_id asc))[1] as icon,\n (array_agg(b.color order by b.exam_board_name asc, b.exam_board_id asc))[1] as color,\n\n jsonb_agg(\n jsonb_build_object(\n 'exam_board_id', b.exam_board_id,\n 'exam_board_name', b.exam_board_name,\n 'subject_id', b.subject_id\n)\n order by b.exam_board_name asc, b.exam_board_id asc\n) as boards\n from base b\n group by\n b.exam_type_id,\n b.subject_name\n order by\n b.subject_name asc;\n$function$\n"
 },
 {

```
  "schema_name": "public",
  "function_name": "rpc_list_subject_names",
  "arguments": "p_exam_type_id uuid",
  "definition": "CREATE OR REPLACE FUNCTION
public.rpc_list_subject_names(p_exam_type_id uuid)\n RETURNS TABLE(subject_name
text)\n LANGUAGE sql\n STABLE\nAS $function$\n  select distinct s.subject_name\n  from
public.subjects s\n  where s.exam_type_id = p_exam_type_id\n  order by s.subject_name
asc;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_list_subject_variants",
  "arguments": "p_exam_type_id uuid, p_subject_name text",
  "definition": "CREATE OR REPLACE FUNCTION
public.rpc_list_subject_variants(p_exam_type_id uuid, p_subject_name text)\n RETURNS
TABLE(subject_id uuid, subject_code text, subject_name text, exam_type_id uuid,
exam_type_name text, exam_board_id uuid, exam_board_name text, breadcrumb_label
text)\n LANGUAGE sql\n STABLE\nAS $function$\n  select\n    s.id as subject_id,\n    s.code
as subject_code,\n    s.subject_name,\n    s.exam_type_id,\n    et.name as
exam_type_name,\n    s.exam_board_id,\n    eb.name as exam_board_name,\n
(s.subject_name || ' • ' || et.name || ' • ' || eb.name) as breadcrumb_label\n  from
public.subjects s\n  join public.exam_types et on et.id = s.exam_type_id\n  join
public.exam_boards eb on eb.id = s.exam_board_id\n  where s.exam_type_id =
p_exam_type_id\n    and s.subject_name = p_subject_name\n  order by eb.name asc, s.code
asc;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_list_subjects_for_exam_types",
  "arguments": "p_exam_type_ids uuid[]",
  "definition": "CREATE OR REPLACE FUNCTION
public.rpc_list_subjects_for_exam_types(p_exam_type_ids uuid[])\n RETURNS
TABLE(subject_id uuid, subject_name text, exam_type_id uuid, exam_board_id uuid,
exam_board_name text, subject_code text, icon text, color text)\n LANGUAGE sql\n STABLE
SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\n  select\n    s.id as
subject_id,\n    s.subject_name,\n    s.exam_type_id,\n    s.exam_board_id,\n    eb.name as
exam_board_name,\n    s.code as subject_code,\n    s.icon,\n    s.color\n  from
public.subjects s\n  join public.exam_boards eb on eb.id = s.exam_board_id\n  where
s.exam_type_id = any(p_exam_type_ids)\n  order by s.subject_name asc, eb.name
asc;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_mark_achievements_notified",
  "arguments": "p_child_id uuid, p_achievement_ids uuid[]",
  "definition": "CREATE OR REPLACE FUNCTION
public.rpc_mark_achievements_notified(p_child_id uuid, p_achievement_ids uuid[] DEFAULT
```

NULL::uuid[])\n RETURNS integer\n LANGUAGE plpgsql\n SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\nDECLARE\n v_count integer;\nBEGIN\n IF p_achievement_ids IS NULL THEN\n -- Mark all unnotified as notified\n UPDATE public.child_achievements\n SET notified = true\n WHERE child_id = p_child_id AND notified = false;\n ELSE\n -- Mark specific achievements as notified\n UPDATE public.child_achievements\n SET notified = true\n WHERE child_id = p_child_id \n AND id = ANY(p_achievement_ids)\n AND notified = false;\n END IF;\n\n GET DIAGNOSTICS v_count = ROW_COUNT;\n RETURN v_count;\nEND;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_parent_create_child_and_plan",
  "arguments": "p_payload jsonb",
  "definition": "CREATE OR REPLACE FUNCTION public.rpc_parent_create_child_and_plan(p_payload jsonb)\n RETURNS jsonb\n LANGUAGE plpgsql\n SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\nDECLARE\n v_parent_id uuid := auth.uid();\n v_child_id uuid;\n v_plan_id uuid;\n v_goal_id uuid;\n v_revision_period_id uuid;\n \n -- Child fields\n v_first_name text;\n v_last_name text;\n v_preferred_name text;\n v_country text;\n v_year_group integer;\n \n -- Validation counts\n v_subject_count int;\n v_cluster_count int := 0;\n v_has_availability boolean := false;\n \n -- Revision period fields\n v_start_date date;\n v_end_date date;\n v_contingency_percent integer;\n v_feeling_code text;\n v_history_code text;\n \n -- Legacy support\n v_legacy_availability jsonb := null;\n v_legacy_exam_timeline text := null;\n \n -- Loop variables\n v_subject record;\n v_day record;\n v_slot record;\n v_override record;\n v_pathway record;\n v_cluster record;\n v_template_id uuid;\n v_override_id uuid;\n v_child_subject_id uuid;\n v_cluster_id uuid;\nBEGIN\n -- ==\n -- Authentication check\n -- ==\n IF v_parent_id IS NULL THEN\n RAISE EXCEPTION 'Not authenticated';\n END IF;\n\n -- ==\n -- HARD GUARD: refuse old needs model\n -- ==\n IF p_payload ? 'needs' THEN\n RAISE EXCEPTION 'Payload contains deprecated key \"needs\". Use \"need_clusters\" instead.';\n END IF;\n\n -- ==\n -- Extract and validate child fields\n -- ==\n v_first_name := nullif(p_payload#>>'{child,first_name}', '');\n v_last_name := nullif(p_payload#>>'{child,last_name}', '');\n v_preferred_name := nullif(p_payload#>>'{child,preferred_name}', '');\n v_country := nullif(p_payload#>>'{child,country}', '');\n v_year_group := nullif(p_payload#>>'{child,year_group}', '')::int;\n\n IF v_first_name IS NULL THEN\n RAISE EXCEPTION 'Missing child.first_name';\n END IF;\n\n IF v_year_group IS NOT NULL AND (v_year_group < 7 OR v_year_group > 13) THEN\n RAISE EXCEPTION 'Invalid child.year_group (must be 7-13)';\n END IF;\n\n --

```
==========
=\n  -- Validate goal\n  --
==========
=\n  SELECT g.id INTO v_goal_id\n  FROM public.goals g\n  WHERE g.code = (p_payload-
>>'goal_code');\n\n  IF v_goal_id IS NULL THEN\n    RAISE EXCEPTION 'Invalid goal_code';\n
END IF;\n\n  --
==========
=\n  -- Validate subjects (support both new and legacy format)\n  --
==========
=\n  \n  -- Check for new format: \"subjects\" array with rich data\n  IF
jsonb_typeof(p_payload->'subjects') = 'array' THEN\n    SELECT count(*) INTO
v_subject_count\n    FROM jsonb_array_elements(p_payload->'subjects') AS s\n    WHERE s-
>>'subject_id' IS NOT NULL;\n  -- Legacy format: \"subject_ids\" flat array\n  ELSIF
jsonb_typeof(p_payload->'subject_ids') = 'array' THEN\n    SELECT count(*) INTO
v_subject_count\n    FROM jsonb_array_elements_text(p_payload->'subject_ids');\n  ELSE\n
v_subject_count := 0;\n  END IF;\n\n  IF v_subject_count = 0 THEN\n    RAISE EXCEPTION 'No
subjects provided (use \"subjects\" array or legacy \"subject_ids\")';\n  END IF;\n\n  --
==========
=\n  -- Check availability (support both new and legacy format)\n  --
==========
=\n  \n  -- New format: \"weekly_availability\"\n  IF jsonb_typeof(p_payload-
>'weekly_availability') = 'object' THEN\n    SELECT EXISTS (\n      SELECT 1\n      FROM
jsonb_each(p_payload->'weekly_availability') AS d(day_key, day_obj)\n      WHERE (day_obj-
>>'enabled')::boolean = true\n      AND jsonb_array_length(coalesce(day_obj->'slots',
'[]'::jsonb)) > 0\n    ) INTO v_has_availability;\n  -- Legacy format: \"settings.availability\"\n
ELSIF jsonb_typeof(p_payload#>'{settings,availability}') = 'object' THEN\n
v_legacy_availability := p_payload#>'{settings,availability}';\n    SELECT EXISTS (\n      SELECT
1\n      FROM jsonb_each(v_legacy_availability) AS d(day_name, day_obj)\n      WHERE
coalesce((day_obj->>'sessions')::int, 0) > 0\n    ) INTO v_has_availability;\n  END IF;\n\n  IF
NOT v_has_availability THEN\n    RAISE EXCEPTION 'No study capacity: must include at least
one day with sessions';\n  END IF;\n\n  --
==========
=\n  -- CREATE CHILD\n  --
==========
=\n  INSERT INTO public.children (parent_id, first_name, last_name, preferred_name,
country, year_group)\n  VALUES (v_parent_id, v_first_name, v_last_name, v_preferred_name,
v_country, v_year_group)\n  RETURNING id INTO v_child_id;\n\n  --
==========
=\n  -- CREATE CHILD GOAL\n  --
==========
=\n  INSERT INTO public.child_goals (child_id, goal_id)\n  VALUES (v_child_id, v_goal_id);\n\n
--
==========
=\n  -- CREATE CHILD SUBJECTS (with grades and priority)\n  -- Note: subject_name and
exam_board_name in payload are ignored (display only)\n  --
==========
=\n  \n  -- New format: \"subjects\" array with rich data\n  IF jsonb_typeof(p_payload-
```

```
>'subjects') = 'array' THEN\n    FOR v_subject IN\n     SELECT \n       (s->>'subject_id')::uuid AS subject_id,\n       coalesce((s->>'sort_order')::int, row_number() OVER ()) AS sort_order,\n nullif(s->>'current_grade', '')::int AS current_grade,\n       nullif(s->>'target_grade', '')::int AS target_grade,\n       coalesce(s->>'grade_confidence', 'confirmed') AS grade_confidence\n FROM jsonb_array_elements(p_payload->'subjects') AS s\n     WHERE s->>'subject_id' IS NOT NULL\n    LOOP\n     INSERT INTO public.child_subjects (\n       child_id, \n subject_id, \n       sort_order, \n       current_grade, \n       target_grade, \n grade_confidence\n     )\n     VALUES (\n       v_child_id,\n       v_subject.subject_id,\n v_subject.sort_order,\n       v_subject.current_grade,\n       v_subject.target_grade,\n v_subject.grade_confidence\n     );\n    END LOOP;\n  -- Legacy format: \"subject_ids\" flat array\n  ELSE\n    INSERT INTO public.child_subjects (child_id, subject_id, sort_order)\n SELECT \n     v_child_id, \n     (x.value)::uuid,\n     row_number() OVER () AS sort_order\n FROM jsonb_array_elements_text(coalesce(p_payload->'subject_ids', '[]'::jsonb)) AS x(value);\n  END IF;\n\n  --
==========
=\n  -- CREATE CHILD PATHWAYS (for tier/option selections)\n  --
==========
=\n  IF jsonb_typeof(p_payload->'pathway_selections') = 'array' THEN\n    FOR v_pathway IN\n     SELECT \n       (ps->>'subject_id')::uuid AS subject_id,\n       (ps->>'pathway_id')::uuid AS pathway_id\n     FROM jsonb_array_elements(p_payload->'pathway_selections') AS ps\n WHERE ps->>'pathway_id' IS NOT NULL\n    LOOP\n     -- Get the child_subject_id for this subject\n     SELECT id INTO v_child_subject_id\n     FROM public.child_subjects\n WHERE child_id = v_child_id\n     AND subject_id = v_pathway.subject_id;\n     \n     -- Insert pathway selection if we found the child_subject\n     IF v_child_subject_id IS NOT NULL THEN\n       INSERT INTO public.child_pathways (child_id, child_subject_id, pathway_id)\n VALUES (v_child_id, v_child_subject_id, v_pathway.pathway_id)\n       ON CONFLICT (child_id, pathway_id) DO NOTHING;\n     END IF;\n   END LOOP;\n  END IF;\n\n  --
==========
=\n  -- CREATE CHILD NEED CLUSTERS\n  --
==========
=\n  IF jsonb_typeof(p_payload->'need_clusters') = 'array' THEN\n    FOR v_cluster IN\n SELECT \n       nc->>'cluster_code' AS cluster_code\n     FROM jsonb_array_elements(p_payload->'need_clusters') AS nc\n     WHERE nullif(nc->>'cluster_code', '') IS NOT NULL\n    LOOP\n     -- Look up cluster_id from code\n     SELECT id INTO v_cluster_id\n     FROM public.need_clusters\n     WHERE code = v_cluster.cluster_code\n     AND is_active = true;\n     \n     IF v_cluster_id IS NOT NULL THEN\n       INSERT INTO public.child_need_clusters (child_id, cluster_id, source)\n VALUES (v_child_id, v_cluster_id, 'observed')\n       ON CONFLICT (child_id, cluster_id) DO NOTHING;\n     END IF;\n   END LOOP;\n  END IF;\n\n  --
==========
=\n  -- CREATE REVISION PERIOD (if provided)\n  --
==========
=\n  v_start_date := nullif(p_payload#>>'{revision_period,start_date}', '')::date;\n  v_end_date := nullif(p_payload#>>'{revision_period,end_date}', '')::date;\n  v_contingency_percent := coalesce(nullif(p_payload#>>'{revision_period,contingency_percent}', '')::int, 10);\n v_feeling_code := nullif(p_payload#>>'{revision_period,feeling_code}', '');\n  v_history_code := nullif(p_payload#>>'{revision_period,history_code}', '');\n\n  IF v_start_date IS NOT NULL
```

```
AND v_end_date IS NOT NULL THEN\n    INSERT INTO public.revision_periods (\n
child_id,\n    start_date,\n    end_date,\n    contingency_percent,\n    feeling_code,\n
history_code\n    )\n    VALUES (\n    v_child_id,\n    v_start_date,\n    v_end_date,\n
v_contingency_percent,\n    v_feeling_code,\n    v_history_code\n    )\n    RETURNING id
INTO v_revision_period_id;\n\n    -- Link to child\n    UPDATE public.children\n    SET
active_revision_period_id = v_revision_period_id\n    WHERE id = v_child_id;\n\n    --
Calculate legacy exam_timeline from dates for plan generator\n    v_legacy_exam_timeline :=
CASE\n    WHEN v_end_date - v_start_date <= 14 THEN '2_weeks'\n    WHEN v_end_date -
v_start_date <= 42 THEN '6_weeks'\n    WHEN v_end_date - v_start_date <= 84 THEN
'3_months'\n    ELSE '6_months'\n    END;\n  ELSE\n    -- Legacy: use exam_timeline
directly\n    v_legacy_exam_timeline := coalesce(p_payload->>'exam_timeline', '6_weeks');\n
END IF;\n\n  --
=========================================================================
=\n  -- CREATE WEEKLY AVAILABILITY TEMPLATE (new format)\n  -- CHANGED: Removed
ON CONFLICT - now allows multiple slots per time_of_day\n  --
=========================================================================
=\n  IF jsonb_typeof(p_payload->'weekly_availability') = 'object' THEN\n    FOR v_day IN\n
SELECT \n      (d.day_key)::int AS day_of_week,\n      coalesce((d.day_obj-
>>'enabled')::boolean, true) AS is_enabled,\n      d.day_obj->'slots' AS slots\n    FROM
jsonb_each(p_payload->'weekly_availability') AS d(day_key, day_obj)\n    LOOP\n    -- Create
template row for this day\n    INSERT INTO public.weekly_availability_template (\n
child_id,\n      day_of_week,\n      is_enabled\n    )\n    VALUES (\n      v_child_id,\n
v_day.day_of_week,\n      v_day.is_enabled\n    )\n    RETURNING id INTO v_template_id;\n
\n    -- Create slots for this day (no ON CONFLICT - multiple slots allowed)\n    IF
v_day.is_enabled AND jsonb_typeof(v_day.slots) = 'array' THEN\n      FOR v_slot IN\n
SELECT \n        s->>'time_of_day' AS time_of_day,\n        s->>'session_pattern' AS
session_pattern\n      FROM jsonb_array_elements(v_day.slots) AS s\n      WHERE s-
>>'time_of_day' IS NOT NULL\n      AND s->>'session_pattern' IS NOT NULL\n      LOOP\n
INSERT INTO public.weekly_availability_slots (\n        template_id,\n        time_of_day,\n
session_pattern\n      )\n      VALUES (\n        v_template_id,\n        v_slot.time_of_day,\n
v_slot.session_pattern\n      );\n      END LOOP;\n    END IF;\n  END LOOP;\n  \n  -- Also
populate legacy revision_schedules for compatibility with plan generator\n  PERFORM
public.rpc_set_revision_schedules_from_weekly_template(v_child_id);\n  \n  -- Legacy
format: use settings.availability\n  ELSIF v_legacy_availability IS NOT NULL THEN\n
PERFORM public.rpc_set_revision_schedules_from_availability(v_child_id,
v_legacy_availability);\n  END IF;\n\n  --
=========================================================================
=\n  -- CREATE DATE OVERRIDES (if provided)\n  -- CHANGED: Removed ON CONFLICT for
override slots too\n  --
=========================================================================
=\n  IF jsonb_typeof(p_payload->'date_overrides') = 'array' THEN\n    FOR v_override IN\n
SELECT \n      (o->>'date')::date AS override_date,\n      o->>'type' AS override_type,\n      o-
>>'reason' AS reason,\n      o->'slots' AS slots\n    FROM jsonb_array_elements(p_payload-
>'date_overrides') AS o\n    WHERE o->>'date' IS NOT NULL\n    AND o->>'type' IS NOT
NULL\n  LOOP\n    INSERT INTO public.availability_date_overrides (\n      child_id,\n
override_date,\n      override_type,\n      reason\n    )\n    VALUES (\n      v_child_id,\n
v_override.override_date,\n      v_override.override_type,\n      v_override.reason\n    )\n
```

```
RETURNING id INTO v_override_id;\n     \n     -- Create slots for extra days (no ON CONFLICT)\n     IF v_override.override_type = 'extra' AND jsonb_typeof(v_override.slots) = 'array' THEN\n       FOR v_slot IN\n         SELECT \n           s->>'time_of_day' AS time_of_day,\n           s->>'session_pattern' AS session_pattern\n         FROM jsonb_array_elements(v_override.slots) AS s\n         WHERE s->>'time_of_day' IS NOT NULL\n         AND s->>'session_pattern' IS NOT NULL\n       LOOP\n         INSERT INTO public.availability_override_slots (\n           override_id,\n           time_of_day,\n           session_pattern\n         )\n         VALUES (\n           v_override_id,\n           v_slot.time_of_day,\n           v_slot.session_pattern\n         );\n       END LOOP;\n     END IF;\n   END LOOP;\n  END IF;\n\n  --
==========
=\n  -- STORE SETTINGS SNAPSHOT (for reference)\n  --
==========
=\n  INSERT INTO public.child_settings (child_id, settings, created_by)\n  VALUES (v_child_id, coalesce(p_payload->'settings', '{}'::jsonb), v_parent_id);\n\n  --
==========
=\n  -- ENSURE GAMIFICATION ROWS EXIST\n  --
==========
=\n  PERFORM public.ensure_child_gamification_rows(v_child_id);\n\n  --
==========
=\n  -- GENERATE PLAN (uses revision_schedules)\n  --
==========
=\n  SELECT public.generate_revision_plan_14_days(v_child_id, v_legacy_exam_timeline)\n  INTO v_plan_id;\n\n  --
==========
=\n  -- RETURN RESULT\n  --
==========
=\n  RETURN jsonb_build_object(\n    'child_id', v_child_id,\n    'plan_id', v_plan_id,\n    'revision_period_id', v_revision_period_id\n  );\nEND;\n$function$\n"
 },
 {
   "schema_name": "public",
   "function_name": "rpc_patch_planned_session_payload",
   "arguments": "p_planned_session_id uuid, p_patch jsonb",
   "definition": "CREATE OR REPLACE FUNCTION public.rpc_patch_planned_session_payload(p_planned_session_id uuid, p_patch jsonb)\n RETURNS jsonb\n LANGUAGE plpgsql\n SECURITY DEFINER\nAS $function$\nDECLARE\nv_result jsonb;\nBEGIN\nUPDATE planned_sessions\nSET \ngenerated_payload = generated_payload || p_patch,\nupdated_at = now()\nWHERE id = p_planned_session_id\nRETURNING generated_payload INTO v_result;\n\nRETURN v_result;\nEND;\n$function$\n"
 },
 {
   "schema_name": "public",
   "function_name": "rpc_save_child_pathways",
   "arguments": "p_child_id uuid, p_pathway_selections jsonb",
```

```
  "definition": "CREATE OR REPLACE FUNCTION public.rpc_save_child_pathways(p_child_id uuid, p_pathway_selections jsonb)\n RETURNS jsonb\n LANGUAGE plpgsql\n SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\nDECLARE\n  v_selection jsonb;\n  v_inserted int := 0;\nBEGIN\n  -- Validate child exists\n  IF NOT EXISTS (SELECT 1 FROM children WHERE id = p_child_id) THEN\n    RETURN jsonb_build_object('success', false, 'error', 'Child not found');\n  END IF;\n\n  -- Process each selection\n  FOR v_selection IN SELECT * FROM jsonb_array_elements(p_pathway_selections)\n  LOOP\n    INSERT INTO child_pathways (child_id, child_subject_id, pathway_id)\n    VALUES (\n      p_child_id,\n      (v_selection->>'child_subject_id')::uuid,\n      (v_selection->>'pathway_id')::uuid\n    )\n    ON CONFLICT (child_id, pathway_id) DO UPDATE\n    SET selected_at = now();\n    \n    v_inserted := v_inserted + 1;\n  END LOOP;\n\n  RETURN jsonb_build_object(\n    'success', true,\n    'selections_saved', v_inserted\n  );\nEND;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_set_child_need_cluster",
  "arguments": "p_child_id uuid, p_cluster_code text, p_source text, p_access_arrangements jsonb, p_diagnosed_date date, p_notes text",
  "definition": "CREATE OR REPLACE FUNCTION public.rpc_set_child_need_cluster(p_child_id uuid, p_cluster_code text, p_source text DEFAULT 'observed'::text, p_access_arrangements jsonb DEFAULT NULL::jsonb, p_diagnosed_date date DEFAULT NULL::date, p_notes text DEFAULT NULL::text)\n RETURNS jsonb\n LANGUAGE plpgsql\n SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\nDECLARE\n  v_cluster_id uuid;\n  v_result_id uuid;\nBEGIN\n  IF p_source NOT IN ('formal', 'observed') THEN\n    RAISE EXCEPTION 'Invalid source. Must be \"formal\" or \"observed\"';\n  END IF;\n  \n  IF p_source = 'observed' AND p_access_arrangements IS NOT NULL THEN\n    RAISE EXCEPTION 'access_arrangements can only be provided when source is \"formal\"';\n  END IF;\n\n  SELECT id INTO v_cluster_id\n  FROM public.need_clusters\n  WHERE code = p_cluster_code AND is_active = true;\n  \n  IF v_cluster_id IS NULL THEN\n    RAISE EXCEPTION 'Need cluster not found or inactive: %', p_cluster_code;\n  END IF;\n\n  INSERT INTO public.child_need_clusters (\n    child_id,\n    cluster_id,\n    source,\n    access_arrangements,\n    diagnosed_date,\n    notes,\n    updated_at\n  )\n  VALUES (\n    p_child_id,\n    v_cluster_id,\n    p_source::public.need_source,\n    p_access_arrangements,\n    p_diagnosed_date,\n    p_notes,\n    now()\n  )\n  ON CONFLICT (child_id, cluster_id) DO UPDATE SET\n    source = EXCLUDED.source,\n    access_arrangements = EXCLUDED.access_arrangements,\n    diagnosed_date = EXCLUDED.diagnosed_date,\n    notes = EXCLUDED.notes,\n    updated_at = now()\n  RETURNING id INTO v_result_id;\n\n  RETURN jsonb_build_object(\n    'id', v_result_id,\n    'child_id', p_child_id,\n    'cluster_code', p_cluster_code,\n    'source', p_source\n  );\nEND;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_set_revision_schedules_from_availability",
  "arguments": "p_child_id uuid, p_availability jsonb",
```

```
  "definition": "CREATE OR REPLACE FUNCTION public.rpc_set_revision_schedules_from_availability(p_child_id uuid, p_availability jsonb)\n RETURNS void\n LANGUAGE plpgsql\n SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\ndeclare\n  r record;\n  v_sessions int;\n  v_ui_pattern text;\n  v_pattern public.session_pattern;\n  v_minutes int;\nbegin\n  if p_child_id is null then\n    raise exception 'Missing child_id';\n  end if;\n\n  -- wipe existing schedules (simple + safe for early stage)\n  delete from public.revision_schedules\n  where child_id = p_child_id;\n\n  if jsonb_typeof(p_availability) <> 'object' then\n    return;\n  end if;\n\n  for r in\n    select key as day_key, value as day_obj\n    from jsonb_each(p_availability)\n  loop\n    v_sessions := coalesce(nullif(r.day_obj->>'sessions','')::int, 0);\n    v_ui_pattern := coalesce(nullif(r.day_obj->>'session_pattern',''), 'p45');\n\n    v_pattern :=\n      case v_ui_pattern\n        when 'p20' then 'SINGLE_20'::public.session_pattern\n        when 'p45' then 'DOUBLE_45'::public.session_pattern\n        when 'p70' then 'TRIPLE_70'::public.session_pattern\n        else 'DOUBLE_45'::public.session_pattern\n      end;\n\n    v_minutes := public.session_pattern_minutes(v_pattern);\n\n    if v_sessions > 0 then\n      insert into public.revision_schedules (\n        child_id,\n        day_of_week,\n        slot_index,\n        session_pattern,\n        session_duration_minutes,\n        is_active,\n        created_at,\n        updated_at\n      )\n      select\n        p_child_id,\n        lower(r.day_key),\n        gs.slot_index,\n        v_pattern,\n        v_minutes,\n        true,\n        now(),\n        now()\n      from generate_series(1, v_sessions) as gs(slot_index);\n    end if;\n  end loop;\nend;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_set_revision_schedules_from_weekly_template",
  "arguments": "p_child_id uuid",
  "definition": "CREATE OR REPLACE FUNCTION public.rpc_set_revision_schedules_from_weekly_template(p_child_id uuid)\n RETURNS void\n LANGUAGE plpgsql\n SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\nDECLARE\n  v_day_names text[] := ARRAY['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', 'sunday'];\n  v_template RECORD;\n  v_slot RECORD;\n  v_slot_index integer;\n  v_session_pattern public.session_pattern;\n  v_duration integer;\nBEGIN\n  -- Delete existing schedules for this child\n  DELETE FROM revision_schedules WHERE child_id = p_child_id;\n\n  -- Loop through each enabled day in the template\n  FOR v_template IN\n    SELECT t.day_of_week, t.is_enabled\n    FROM weekly_availability_template t\n    WHERE t.child_id = p_child_id AND t.is_enabled = true\n    ORDER BY t.day_of_week\n  LOOP\n    v_slot_index := 1;\n\n    -- Loop through each slot for this day (ordered by time_of_day)\n    FOR v_slot IN\n      SELECT s.time_of_day, s.session_pattern\n      FROM weekly_availability_slots s\n      JOIN weekly_availability_template t ON t.id = s.template_id\n      WHERE t.child_id = p_child_id AND t.day_of_week = v_template.day_of_week\n      ORDER BY \n        CASE s.time_of_day\n          WHEN 'early_morning' THEN 1\n          WHEN 'morning' THEN 2\n          WHEN 'afternoon' THEN 3\n          WHEN 'evening' THEN 4\n          WHEN 'before_school' THEN 1\n          WHEN 'after_school' THEN 3\n          ELSE 5\n        END,\n        s.id\n    LOOP\n      v_session_pattern := CASE v_slot.session_pattern\n        WHEN 'p20' THEN 'SINGLE_20'::public.session_pattern\n        WHEN 'p45' THEN 'DOUBLE_45'::public.session_pattern\n        WHEN 'p70' THEN 'TRIPLE_70'::public.session_pattern\n        ELSE 'DOUBLE_45'::public.session_pattern\n
```

```
END;\n\n    v_duration := CASE v_slot.session_pattern\n      WHEN 'p20' THEN 20\n      WHEN 'p45' THEN 45\n      WHEN 'p70' THEN 70\n      ELSE 45\n    END;\n\n    INSERT INTO revision_schedules (\n      id,\n      child_id,\n      day_of_week,\n      slot_index,\n      session_pattern,\n      session_duration_minutes,\n      created_at,\n      updated_at\n    ) VALUES (\n      gen_random_uuid(),\n      p_child_id,\n      v_day_names[v_template.day_of_week + 1],\n      v_slot_index,\n      v_session_pattern,\n      v_duration,\n      now(),\n      now()\n    );\n\n    v_slot_index := v_slot_index + 1;\n  END LOOP;\n END LOOP;\nEND;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_skip_planned_session",
  "arguments": "p_planned_session_id uuid, p_notes text",
  "definition": "CREATE OR REPLACE FUNCTION public.rpc_skip_planned_session(p_planned_session_id uuid, p_notes text DEFAULT NULL::text)\n RETURNS jsonb\n LANGUAGE plpgsql\n SECURITY DEFINER\nAS $function$\nBEGIN\nUPDATE planned_sessions\nSET \nstatus = 'skipped',\nupdated_at = now(),\ngenerated_payload = generated_payload || jsonb_build_object(\n'skip_reason', p_notes\n)\nWHERE id = p_planned_session_id;\n\nRETURN jsonb_build_object('success', true);\nEND;\n$function$\n"
 },
 {
  "schema_name": "public",
  "function_name": "rpc_start_planned_session",
  "arguments": "p_planned_session_id uuid",
  "definition": "CREATE OR REPLACE FUNCTION public.rpc_start_planned_session(p_planned_session_id uuid)\n RETURNS TABLE(out_planned_session_id uuid, out_status text, out_started_at timestamp with time zone, out_revision_session_id uuid)\n LANGUAGE plpgsql\n SECURITY DEFINER\nAS $function$\ndeclare\n  v_child_id uuid;\n  v_subject_id uuid;\n  v_topic_ids uuid[];\n  v_primary_topic_id uuid;\n  v_total_topics int;\n  v_started_at timestamptz;\n\n  v_payload jsonb;\n\n  v_cards_count int := 0;\n  v_questions_count int := 0;\n  v_has_worked boolean := false;\n\n  v_rs_id uuid;\nbegin\n  -- Lock planned session row and get topic_ids array\n  select ps.child_id, ps.subject_id, ps.topic_ids, ps.topic_ids[1], ps.started_at\n    into v_child_id, v_subject_id, v_topic_ids, v_primary_topic_id, v_started_at\n  from public.planned_sessions ps\n  where ps.id = p_planned_session_id\n  for update;\n\n  if not found then\n    raise exception 'Planned session not found: %', p_planned_session_id;\n  end if;\n\n  -- Calculate total topics from array\n  v_total_topics := coalesce(array_length(v_topic_ids, 1), 1);\n\n  -- Ensure payload exists\n  select ps.generated_payload into v_payload\n  from public.planned_sessions ps\n  where ps.id = p_planned_session_id;\n\n  if v_payload is null or v_payload = '{}'::jsonb then\n    v_payload := public.rpc_generate_planned_session_payload(p_planned_session_id);\n  end if;\n\n  -- Extract counts from payload (FlashcardViewer.cards, PracticeQuestionCard.questions)\n  select coalesce(jsonb_array_length(x.step->'cards'), 0)\n    into v_cards_count\n  from jsonb_array_elements(v_payload->'steps') x(step)\n  where x.step->>'component' = 'FlashcardViewer'\n  limit 1;\n\n  select coalesce(jsonb_array_length(x.step->'questions'), 0)\n    into v_questions_count\n  from jsonb_array_elements(v_payload->'steps') x(step)\n
```

```
where x.step->>'component' = 'PracticeQuestionCard'\n  limit 1;\n\n  select exists (\n  select 1\n    from jsonb_array_elements(v_payload->'steps') x(step)\n    where x.step->>'component' = 'WorkedExampleCard'\n      and coalesce(x.step->'example','{}'::jsonb) <> '{}'::jsonb\n  ) into v_has_worked;\n\n  -- Create or reuse revision_sessions row (unique planned_session_id)\n  -- Now includes current_topic_index and total_topics\n  insert into public.revision_sessions (\n    child_id, subject_id, topic_id, planned_session_id,\n    status, started_at,\n    current_step, current_step_index, current_item_index,\n    current_topic_index, total_topics\n  )\n  values (\n    v_child_id, v_subject_id, v_primary_topic_id, p_planned_session_id,\n    'in_progress', now(),\n    'recall', 0, 0,\n    0, v_total_topics\n  )\n  on conflict (planned_session_id)\n  do update set\n    status = case when public.revision_sessions.status = 'completed' then 'completed' else 'in_progress' end,\n    started_at = coalesce(public.revision_sessions.started_at, now()),\n    current_step = coalesce(public.revision_sessions.current_step, 'recall'),\n    current_step_index = coalesce(public.revision_sessions.current_step_index, 0),\n    current_item_index = coalesce(public.revision_sessions.current_item_index, 0),\n    -- Preserve topic index if already set, otherwise start at 0\n    current_topic_index = coalesce(public.revision_sessions.current_topic_index, 0),\n    total_topics = v_total_topics\n  returning id into v_rs_id;\n\n  -- Upsert step rows (recall/reinforce/practice/reflection)\n  -- totals:\n  --  recall: 1 prompt\n  --  reinforce: cards + worked example (if present)\n  --  practice: questions\n  --  reflection: 1 toggle\n  insert into public.revision_session_steps (revision_session_id, step_key, status, started_at, total_items, current_item_index, payload, answer_summary)\n  values\n    (v_rs_id, 'recall',    'in_progress', now(), 1, 0, '{}'::jsonb, '{}'::jsonb),\n    (v_rs_id, 'reinforce',  'not_started', null, greatest(v_cards_count + (case when v_has_worked then 1 else 0 end), 1), 0, '{}'::jsonb, '{}'::jsonb),\n    (v_rs_id, 'practice', 'not_started', null, greatest(v_questions_count, 1), 0, '{}'::jsonb, '{}'::jsonb),\n    (v_rs_id, 'reflection', 'not_started', null, 1, 0, '{}'::jsonb, '{}'::jsonb)\n  on conflict (revision_session_id, step_key)\n  do update set\n    total_items = excluded.total_items,\n    current_item_index = 0,\n    payload = public.revision_session_steps.payload,        -- keep any prior answers\n    answer_summary = public.revision_session_steps.answer_summary;\n\n  -- Mark planned session started (keep this conservative: timestamps only)\n  update public.planned_sessions\n  set started_at = coalesce(started_at, now()),\n      status = case when status = 'planned' then 'started' else status end\n  where id = p_planned_session_id;\n\n  out_planned_session_id := p_planned_session_id;\n  out_status := 'started';\n  out_started_at := (select started_at from public.planned_sessions where id = p_planned_session_id);\n  out_revision_session_id := v_rs_id;\n  return next;\nend;\n$function$\n"
 },
 {
   "schema_name": "public",
   "function_name": "rpc_update_child_revision_profile",
   "arguments": "p_child_id uuid, p_exam_date date, p_current_feeling text, p_previous_experience text",
   "definition": "CREATE OR REPLACE FUNCTION public.rpc_update_child_revision_profile(p_child_id uuid, p_exam_date date DEFAULT NULL::date, p_current_feeling text DEFAULT 'on_track'::text, p_previous_experience text DEFAULT 'first_time'::text)\n RETURNS jsonb\n LANGUAGE plpgsql\n SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\nDECLARE\n  v_profile jsonb;\n
```

v_exam_timeline jsonb;\nBEGIN\n IF p_current_feeling NOT IN ('on_track', 'bit_behind', 'not_sure', 'crisis') THEN\n   RAISE EXCEPTION 'Invalid current_feeling. Must be one of: on_track, bit_behind, not_sure, crisis';\n END IF;\n \n IF p_previous_experience NOT IN ('really_well', 'ok', 'struggle', 'first_time') THEN\n   RAISE EXCEPTION 'Invalid previous_experience. Must be one of: really_well, ok, struggle, first_time';\n END IF;\n\n v_profile := public.compute_revision_profile(p_exam_date, p_current_feeling, p_previous_experience);\n \n v_exam_timeline := jsonb_build_object(\n   'exam_date', p_exam_date,\n   'current_feeling', p_current_feeling,\n   'previous_experience', p_previous_experience,\n   'captured_at', now()\n);\n\n UPDATE public.children\n SET \n   revision_profile = v_profile,\n   exam_timeline = v_exam_timeline,\n   revision_profile_updated_at = now(),\n   updated_at = now()\n WHERE id = p_child_id;\n\n IF NOT FOUND THEN\n   RAISE EXCEPTION 'Child not found: %', p_child_id;\n END IF;\n\n RETURN jsonb_build_object(\n   'child_id', p_child_id,\n   'revision_profile', v_profile,\n   'exam_timeline', v_exam_timeline\n);\nEND;\n$function$\n"
  },
  {
    "schema_name": "public",
    "function_name": "rpc_update_child_streak",
    "arguments": "p_child_id uuid",
    "definition": "CREATE OR REPLACE FUNCTION public.rpc_update_child_streak(p_child_id uuid)\n RETURNS jsonb\n LANGUAGE plpgsql\n SECURITY DEFINER\n SET search_path TO 'public'\nAS $function$\nDECLARE\n v_today date := CURRENT_DATE;\n v_last_completed date;\n v_current_streak integer;\n v_longest_streak integer;\n v_last_scheduled_date date;\n v_streak_broken boolean := false;\n v_new_streak integer;\n v_new_longest integer;\nBEGIN\n -- Ensure gamification rows exist\n PERFORM public.ensure_child_gamification_rows(p_child_id);\n\n -- Get current streak state\n SELECT last_completed_date, current_streak, longest_streak\n INTO v_last_completed, v_current_streak, v_longest_streak\n FROM public.child_streaks\n WHERE child_id = p_child_id;\n\n -- Find the most recent scheduled day that was completed (today or before)\n SELECT MAX(ps.session_date::date)\n INTO v_last_scheduled_date\n FROM public.planned_sessions ps\n WHERE ps.child_id = p_child_id\n   AND ps.status = 'completed'\n   AND ps.session_date::date <= v_today;\n\n -- If no completed sessions, streak is 0\n IF v_last_scheduled_date IS NULL THEN\n   UPDATE public.child_streaks\n   SET current_streak = 0, updated_at = now()\n   WHERE child_id = p_child_id;\n\n   RETURN jsonb_build_object(\n   'child_id', p_child_id,\n   'current_streak', 0,\n   'longest_streak', v_longest_streak,\n   'last_completed_date', NULL,\n   'streak_broken', false\n   );\n END IF;\n\n -- Check if any scheduled day between last_completed and today was skipped\n IF v_last_completed IS NOT NULL THEN\n   IF EXISTS (\n   SELECT 1\n   FROM public.planned_sessions ps\n   WHERE ps.child_id = p_child_id\n   AND ps.session_date::date > v_last_completed\n   AND ps.session_date::date < v_last_scheduled_date\n   AND ps.status != 'completed'\n   AND EXISTS (\n   SELECT 1 FROM public.revision_schedules rs \n   WHERE rs.child_id = p_child_id \n   AND rs.day_of_week = lower(to_char(ps.session_date, 'FMDay'))\n   AND rs.is_active = true\n   )\n   ) THEN\n   v_streak_broken := true;\n   END IF;\n END IF;\n\n -- Calculate new streak\n IF v_streak_broken OR v_last_completed IS NULL THEN\n   -- Start fresh streak from today's completion\n   v_new_streak := 1;\n ELSIF v_last_scheduled_date = v_last_completed THEN\n   -- Same day, no change\n   v_new_streak := v_current_streak;\n

ELSE\n -- Continuing streak\n v_new_streak := v_current_streak + 1;\n END IF;\n\n -- Update longest if needed\n v_new_longest := GREATEST(v_longest_streak, v_new_streak);\n\n -- Save streak state\n UPDATE public.child_streaks\n SET \n current_streak = v_new_streak,\n longest_streak = v_new_longest,\n last_completed_date = v_last_scheduled_date,\n updated_at = now()\n WHERE child_id = p_child_id;\n\n RETURN jsonb_build_object(\n 'child_id', p_child_id,\n 'current_streak', v_new_streak,\n 'longest_streak', v_new_longest,\n 'last_completed_date', v_last_scheduled_date,\n 'streak_broken', v_streak_broken\n);\nEND;\n$function$\n"
  }
]