

Designing a database with ChatGPT

Author: Dr. René Veldwijk, December 2025

The following is a redacted session between the author and ChatGPT 5.1. We want ChatGPT to help design a database that covers part of an HR application under development. The case calls for modeling at a higher level of abstraction than the input superficially suggests, which is essential to designing a robust data(base) model.

TL;DR: Conclusions from testing ChatGPT

All conclusions are tentative. This is merely a superficial analysis. Proper prompt engineering may make a difference. And there are already custom GPTs available created by users.

This is what ChatGPT itself advises:

- **For serious modeling, it is often better to stick with:**
 - Standard ChatGPT + strict instructions
 - Optionally, your own Custom GPT with your fixed rules (normalization level, naming conventions, Oracle focus, etc.)

With these caveats in mind I would suggest the following:

1. ChatGPT does not do a good job as a data modeler in the sense that it does not seem to be able to capture more abstract patterns that may correspond to better data models and database designs.
2. ChatGPT vacillates between different levels of abstraction (i.e., different patterns) resulting in worst of all worlds database designs.
3. ChatGPT does give the data modeler relevant information about important fundamental aspects that should often be taken account in deciding about a data model's fundament. This is useful for novice modelers.
4. As so often, ChatGPT almost slavishly follows the user's train of thought. In the hands of novice modelers this is really dangerous.
5. The lack of accepted data modeling concepts and the overflow of labels marketed as concepts are demonstrated both in suboptimal designs and strange explanations.
6. In the hand of competent professionals and well-developed prompts ChatGPT seems somewhat useful as a design tool but very useful as an implementation tool. Complex constraints expressed in SQL statements are easily translated into database triggers. Even better, ChatGPT recognizes the constellation of constraints to some extent. Example: In this test (not in the document) ChatGPT knew and explained that a SQL-constraint expressing the rule that an organizational units budget must exceed the total of the salaries of the units employees does not need to be checked when an employee is deleted but should be checked if the constraint that salaries are positive is dropped. If this would work for more complex situations it would mean a significant step forward for database design and implementation and data integrity management.

Case description:

The testcase for ChatGPT as a data model designer is taken from our ‘Advanced Data Modeling’ training course. The student is given a description of an organization hierarchy for a specific organization that is part of an HR application under development. The crux of the case is that translating the description directly into a database design is a very bad idea as it leads to a database that is expensive to build and probably subject to extensive and expensive maintenance or database abuse. Fortunately, the higher-level pattern that leads to a robust and low-maintenance database model is abundantly clear.

The case description:

Case I: ‘Warm-up’

- Your analysis for a new Personnel IS teaches you the following:
“The company is subdivided into four divisions, each consisting of a number of product groups, which are in turn subdivided into departments. Divisions, product groups, and departments each have a unique code, a name, and a budget.”
- Assignment: Model this situation using a Data Structure Diagram (or Bachman schema or GSD) and table descriptions

Almost always students come up with models reflecting the abstraction level of the case description. So we see data classes like ‘Division’, ‘Product Group’ and ‘Department’ which leads to a really bad database design. We primarily want to see if ChatGPT falls into the same trap or comes up with the optimal solution. As we shall see, the answer is way more complicated than just a ‘yes’ or ‘no’.

For practical purposes the ChatGPT session (starts at next page) has been redacted, but the unredacted version is of course available.

We are not trying to trick ChatGPT 5.1 but try to follow a not unrealistic process a designer using ChatGPT might follow. Essential to the case is that it is limited to just a part of the database: the organization structure that is the backbone of any HR system’s master data. If Employee data had been included in the case the right design choice would probably have been obvious.

Anyway. Judge for yourself!

The ChatGPT session

Notes on my part will be in this style.

Significant parts of ChatGPT will be designated like this.

You said:

We are designing a tailor-made HR application for a large company.

The part we want to design is a organization structure. From an HR point of view this is master data. We want to get this implemented and working in our first sprint.

We are providing context to ChatGPT. The case is about and HR application, so we may expect data classes like 'Employee', but also information about the structure of the organization. And the application serves only one specific organization. All this is relevant input for the design process.

ChatGPT said:

Good, clear scope.

To avoid running ahead: a few key choices first. Please answer briefly.

1. Nodes
 - Do you only need organizational units (e.g. Division → Department → Team), or also Positions and/or Jobs as first-class entities?
2. Link to people
 - Do employees link directly to an org unit, or do they occupy a Position which belongs to an org unit?
3. History
 - Must org structure and assignments be time-based (valid-from / valid-to), or is “current state only” enough for sprint 1?
4. Multiple assignments
 - Can an employee have multiple concurrent org assignments (e.g. 70% Dept A, 30% Project B)?
5. Multiple hierarchies
 - Do you need more than one hierarchy type (e.g. Line org vs. Cost center vs. Project), or just one primary hierarchy for sprint 1?

Once you've fixed these, I'll design a minimal but solid relational model for this part.

This is a really good response from ChatGPT, albeit maybe a bit premature, but ChatGPT is always trying to generate interaction – also for paid customers like the author. For a novice designer this is fine, however. And it shows that ChatGPT really makes use of background information.

You said:

#1: correct. #2: directly linked but for a later sprint. #3: current only for now #4: single assignment (current). #5: I'll tell you.

ChatGPT said:

Given your choices, here is a minimal org-structure schema for sprint 1 (single primary hierarchy, current state only).

Entities:

- OrgUnitType – categorizes units (Company, Division, Department, Team, etc.).
- OrgUnit – the actual units in a tree (parent-child).
- EmployeeOrgAssignment – optional stub for later, linking employees to org units (current only, single assignment).

Logical model:

- One hierarchy: OrgUnit.parent_org_unit_id → OrgUnit.org_unit_id (root has NULL parent).
- Every OrgUnit has exactly one OrgUnitType.
- An employee can have at most one current org-unit assignment (enforced via a unique constraint when you add employees).

Example DDL (generic SQL, easy to adapt to Oracle):

```
CREATE TABLE OrgUnitType (  
  org_unit_type_id BIGINT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  
  code VARCHAR(30) NOT NULL UNIQUE, -- e.g. COMPANY, DIVISION, DEPT, TEAM  
  name VARCHAR(100) NOT NULL  
);  
  
CREATE TABLE OrgUnit (  
  org_unit_id BIGINT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  
  org_unit_type_id BIGINT NOT NULL  
    REFERENCES OrgUnitType(org_unit_type_id),  
  parent_org_unit_id BIGINT NULL  
    REFERENCES OrgUnit(org_unit_id),  
  code VARCHAR(50) NOT NULL UNIQUE, -- business key, used in integrations  
  name VARCHAR(200) NOT NULL,  
  is_active BOOLEAN NOT NULL DEFAULT TRUE  
  -- later: add valid_from/valid_to for history  
);  
  
-- For a later sprint (Linking people directly to OrgUnit)  
-- Employee table assumed to exist then.  
CREATE TABLE EmployeeOrgAssignment (  
  employee_id BIGINT NOT NULL, -- FK → Employee(employee_id)  
  org_unit_id BIGINT NOT NULL  
    REFERENCES OrgUnit(org_unit_id),  
  PRIMARY KEY (employee_id),  
  CONSTRAINT uq_emp_org UNIQUE (employee_id) -- single current assignment  
  -- later: add dates and move PK to (employee_id, valid_from)  
);
```

If you want, next step we can:

- Check this against the queries you must support in sprint 1 (e.g. “get full tree”, “who reports under X”, etc.).
- Adapt syntax specifically to Oracle (identity vs sequences, boolean emulation).

ChatGPT has this tendency to come up with premature answers. Also, this is not a complete data model / database specification. It is just relational data structures plus standard relational constraints. That said, ChatGPT comes up with a more abstract model than the case description, which still has to be provided, suggests.

You said:

Again: Please don't come up with all kinds of premature texts. This is a generic requirement on my part.

ChatGPT said:

Helder.

Ik zal geen voorstellen of uitwerkingen meer geven voordat jij daar expliciet om vraagt.

ChatGPT reverts to Dutch.

You said:

The organization (Survival Chemicals Inc.) is made up of 4 divisions: Explosives, Catalysts, Coatings and Vitamins. Each division has multiple product groups (currently between 2 and 7), 21 in total. Product groups are divided into departments (currently between 3 and 12), 98 in total. It is a straightforward hierarchical government model.

Here's the main part of the treacherously simple case description. Note that the case also contains information about the model's/database's content, which is irrelevant to the model itself. But this is totally realistic, as the designer might even upload an organization chart.

ChatGPT said:

Understood: single line hierarchy

Division → Product group → Department, fixed 3 levels, current-state only.

Next two points I need before designing tables:

1. Hierarchies
 - Is this the only hierarchy you need in sprint 1 (no separate cost-center, project, legal-entity trees)?
2. Required queries in sprint 1

Very concretely: name 3–5 questions the system must answer, e.g.

 - “List all departments in the Coatings division”
 - “Given a department, show its division and product group”
 - “Show the full structure as a tree”

Once you list those, I will propose a minimal relational design that supports exactly that.

You said:

#1 It is that simple indeed. Also, as far as columns are concerned we only need to record ID, Name and yearly budget (in Euro x1000). #2: Questions like that, indeed. But it is a database, so no optimizations for certain types of queries please.

It is NOT a three level hierarchy because there will also be a board, but we will not correct that for now.

ChatGPT said:

Proposal, exactly matching your constraints.

Logical model

- Division(division_id, name, yearly_budget_k_eur)
- ProductGroup(product_group_id, division_id, name, yearly_budget_k_eur)
- Department(department_id, product_group_id, name, yearly_budget_k_eur)

Example DDL (generic / Oracle-ish):

```
sql

CREATE TABLE Division (
  division_id      NUMBER(10)      PRIMARY KEY,
  name             VARCHAR2(100)   NOT NULL,
  yearly_budget_k_eur  NUMBER(12,0) NOT NULL
);

CREATE TABLE ProductGroup (
  product_group_id  NUMBER(10)      PRIMARY KEY,
  division_id       NUMBER(10)      NOT NULL
    REFERENCES Division(division_id),
  name              VARCHAR2(100)   NOT NULL,
  yearly_budget_k_eur  NUMBER(12,0) NOT NULL
);

CREATE TABLE Department (
  department_id     NUMBER(10)      PRIMARY KEY,
  product_group_id  NUMBER(10)      NOT NULL
    REFERENCES ProductGroup(product_group_id),
  name              VARCHAR2(100)   NOT NULL,
  yearly_budget_k_eur  NUMBER(12,0) NOT NULL
);
```

If you want, next step can be: sample rows for the 4 divisions / 21 product groups / 98 departments.

Now breaks my wooden shoe! This modeling case fits perfectly in ChatGPT's more abstract premature model. But it has somehow decided to lower the level of abstraction to the more concrete input. In practice, this would be a lethal error.

And as we noted, ChatGPT has overlooked that the organization will also have a board, probably manned by employees. So this model is not just low-abstract but also woefully incomplete.

You said:

This model conforms to my input. But it does not feel very robust. Even when I disregard the effect of adding temporal aspects - you were right to mention this... What do you think?

Note that we try to use the Socratic method here, hopefully forcing ChatGPT to explain itself.

ChatGPT said:

Short answer: functionally OK for sprint 1, but structurally fragile.

Main robustness issues:

1. Hard-coded levels
 - Division / ProductGroup / Department as 3 separate tables bakes in exactly three levels.
 - Any future need for e.g. "Business Unit" between Division and ProductGroup, or "Team" under Department, forces schema changes, migrations, and code changes.
2. No generic org-unit concept
 - You cannot treat "an org unit" uniformly in queries or constraints.
 - Common attributes you might want later (manager, cost center, location, cost allocation key, etc.) must be repeated on three tables.
3. Poor support for future hierarchies
 - If they ever want a second structure (reporting vs. cost vs. legal entity), you have no place to model that without duplicating logic.
4. Budgets are very simplistic
 - Only one budget per unit, no versioning (original vs. revised), no validity period, no distinction-n between planned vs. actual, no currency metadata.
 - Aggregation is left entirely to queries; you cannot define any constraint that e.g. department budgets must sum to the product group budget.
5. No temporal structure (even if we ignore it now)
 - Any change to the tree overwrites history; when you later add time, you'll have to refactor more heavily because of the level-specific tables.

So: what you have is a thin "just enough" 3-table design, but not a robust organizational master. If you want, I can sketch a more future-proof variant (single OrgUnit table + hierarchy) that still meets your current simplicity constraints.

ChatGPT comes up with the relevant insights (most notably #2), except that the low-abstraction model isn't really easier to implement than the original version. And at no point the dialogue have I suggested that I expect this lower abstract model – expressed as an incomplete database design, by the way..

You said:

You are right. But the more acute problem is that we sketched a very similar model, but the team that is currently preparing the next sprint is really unhappy about this design. This sprint mainly covers the employee life cycle. Unfortunately, all are on maternity leave now. Do you have any idea what could be so concerning to them?

Here we play the clueless designer role by referring to a separate team that is tasked with implementing the Employee part of the model. Compared to ChatGPT's original model, this low-abstract model is a dramatically bad design, as a data class containing employees has to reference all tables containing organizational units. (The reference to maternity leave based absence of the team is not a sign of good taste, but we stick to the original conversation, only correcting typing errors.)

ChatGPT said:

I can't know their exact reasoning, but for an HR life-cycle sprint there are a few very plausible "red flags" in this design:

1. No positions/jobs
 - HR thinks in terms of positions (FTE slots) and jobs (roles), not just departments.
 - Without positions, they can't model vacancies, headcount vs FTE, backfills, or transfers properly.
2. Org tree too hard-coded
 - Three separate tables (Division / ProductGroup / Department) make reorganizations painful: adding a level (e.g. Business Unit) or splitting/merging departments becomes a technical project, not just data maintenance.
 - HR people know reorgs happen all the time.
3. No temporal validity
 - Employee life cycle processes (hire, transfer, promotion, manager change, reorg) are time-based.
 - With "current only" org and no dates, you can't answer basic questions like "who was their manager on 2025-03-01?" or "which org did they belong to when they got that promotion?".
4. Weak link to "manager" and reporting lines
 - The model has no place to record who leads a division/product group/department.
 - Manager chains (for approvals, performance reviews, salary workflows) depend heavily on a clear, time-aware org and position structure.
5. Only one simple hierarchy
 - HR often needs at least: line org, cost center / financial org, sometimes project/org assignments.

- A rigid, single hierarchy means they foresee issues with payroll, finance, cost allocation, and reporting.
6. Budgets modeled too simplistically
 - One static yearly budget per unit, no versions, no currency metadata, no history.
 - For planning headcount and salaries, they usually expect budget structures that can change over time.
 7. Hard to integrate with other HR/finance systems
 - External systems typically use stable org-unit or cost-center codes, often with their own hierarchies and validity dates.
 - The simplistic three-table model may not map cleanly to what they know they must interface with.

In short: they probably feel this org model will become a constraint the moment they start modeling positions, managers, transfers, approvals, and history—i.e. exactly what an “employee life cycle” sprint needs.

ChatGPT comes up with some reasonable explanations, but not with the most obvious one: Adding an EMPLOYEE table results in references to all three (actually four) tables containing organizational units. Every data class in the HR system – most notably the EMPLOYEE class – will have to reference all specializations of the more abstract concept of ‘organizational unit’

You said:

OK. We're all guessing here. Let's assume that we add the EMPLOYEE table to the model you suggested. And let's keep it simple: just ID, name, salary, gender, birthdate and organizational position. Forget about job titles and so on. Expand the model unless you have any questions you want answered.

ChatGPT said:

Extended model with Employee added to the existing 3-level org.

Logical model

- Division(division_id, name, yearly_budget_k_eur)
- ProductGroup(product_group_id, division_id, name, yearly_budget_k_eur)
- Department(department_id, product_group_id, name, yearly_budget_k_eur)
- Employee(employee_id, name, salary_eur, gender, birthdate, department_id)

Example DDL (Oracle style):

Create Table statements not interesting, so not shown here.

If you want to adjust “organizational position” to point to another level (ProductGroup/Division) or to a future Position table, we can change only the department_id column / FK.

ChatGPT really goes off the rails here, as it is obvious that employees can be assigned to all levels of the organizational hierarchy. It would be interesting to know what makes this design the most reasonable choice for the LLM algorithm.

You said:

Obviously, employees exist not only at the department level. Each product group has a management team, for example.

ChatGPT said:

Right, so we must allow assignments at any of the 3 levels.

Change:

1. Remove department_id from Employee.
2. Introduce an assignment table that can point to Division, ProductGroup or Department, with a constraint that exactly one is filled.

Model (Oracle style, compact):

Create Table statements only shown for Employee-related tables as the other tables remain unchanged.

```
CREATE TABLE Employee (  
  employee_id      NUMBER(10)      PRIMARY KEY,  
  name             VARCHAR2(200)   NOT NULL,  
  salary_eur       NUMBER(12,2)    NOT NULL CHECK (salary_eur >= 0),  
  gender           CHAR(1)         NOT NULL CHECK (gender IN ('M','F','X')),  
  birthdate        DATE            NOT NULL CHECK (birthdate <= TRUNC(SYSDATE))  
);  
  
CREATE TABLE EmployeeOrgAssignment (  
  employee_id      NUMBER(10)      PRIMARY KEY  
    REFERENCES Employee(employee_id),  
  division_id      NUMBER(10)      NULL  
    REFERENCES Division(division_id),  
  product_group_id NUMBER(10)      NULL  
    REFERENCES ProductGroup(product_group_id),  
  department_id    NUMBER(10)      NULL  
    REFERENCES Department(department_id),  
  -- exactly one level must be filled  
  CONSTRAINT chk_emp_org_one_level  
    CHECK (  
      (CASE WHEN division_id IS NULL THEN 0 ELSE 1 END +  
       CASE WHEN product_group_id IS NULL THEN 0 ELSE 1 END +  
       CASE WHEN department_id IS NULL THEN 0 ELSE 1 END) = 1  
    )  
);
```

This lets you put employees at:

- division level (management teams),
- product-group level,
- department level,

without changing the rest of the org model.

Now ChatGPT is forced to add references to three tables (should be four) to make it possible to staff the entire organization (except the board). But this is not done by adding references to the EMPLOYEE table but by creating a separate assignment table that has the same primary key as the EMPLOYEE table. This choice unnecessarily complicates an already unsatisfying database design.

You said:

Why not add these references to the EMPLOYEE table?

ChatGPT said:

Short version: you can, but it makes future change harder.

More detail, compact:

1. Conceptual separation

- EMPLOYEE = intrinsic person attributes (name, birthdate, gender, salary).
- Assignment to the org structure is a relationship, not an intrinsic property. That belongs in its own table.
- This is classic normalization: “who they are” vs “where they sit”.

2. Future: multiple or historical assignments

- As soon as you need:
 - more than one concurrent role (e.g. dept + project), or
 - history (old and new departments),
the “three FKs on EMPLOYEE” breaks: you can’t store more than one row per employee.
- With a separate EMPLOYEE_ORG_ASSIGNMENT, you can evolve to multiple rows per employee by changing the PK, without touching EMPLOYEE.

3. Future: changing the org model

- Today you have 3 levels. If later you:
 - insert a level (BusinessUnit), or
 - collapse to a single generic ORG_UNIT table,
you’d have to alter EMPLOYEE directly if the FKs live there.
- If the link is in a separate table, you refactor that one table and keep EMPLOYEE stable.

4. Technical equivalence for “now”

- For your current, strict requirements (1 current assignment, no history), these designs are functionally equivalent:
 - FKs on EMPLOYEE, or
 - FKs in a 1–1 EMPLOYEE_ORG_ASSIGNMENT.

- The separate table is mainly a hedge against very predictable future requirements in HR systems.

If you are absolutely sure you will never support history or multiple assignments and are willing to refactor later, putting the references directly on EMPLOYEE is acceptable and slightly simpler right now.

We find this explanation really interesting. The spurious assignment table just adds complexity as constraints have now to be added to the model that would otherwise be enforced by the EMPLOYEE table structure. This probable reflects a design practice fed into the model as training data.

#1 is an almost mystical argument that seems to go back to the Entity-Relation modeling era. If you train a model with outdated data paradigms you can get explanations like these. And it has nothing to do with 'normalisation'.

The other arguments refer to hypothetical future changes to the model that would be no less cumbersome to implement if all info would be in the EMPLOYEE table itself.

You said:

This really helps me putting my finger on the lack of robustness. I suddenly realize that you made a serious design error!

Time to trigger ChatGPT to show more about its inner working. We can always count on ChatGPT's talkative nature.

ChatGPT said:

You're right that there is a real design problem here.

The core structural error is this:

- We modeled the organization as 3 different tables (DIVISION / PRODUCT_GROUP / DEPARTMENT).
- Then we modeled the employee-org link with 3 separate optional FKs + a CHECK to ensure exactly one is filled.

That "three-table + three-FK" pattern is inherently brittle: every time the org structure changes (extra level, rename, merge levels), both the org tables and the assignment table must be rebuilt.

A more robust correction (still simple, still current-state only):

```

CREATE TABLE OrgUnitType (
  org_unit_type_id NUMBER(10) PRIMARY KEY,
  code VARCHAR2(30) NOT NULL UNIQUE, -- DIV, PGRP, DEPT
  name VARCHAR2(100) NOT NULL
);

CREATE TABLE OrgUnit (
  org_unit_id NUMBER(10) PRIMARY KEY,
  org_unit_type_id NUMBER(10) NOT NULL
  REFERENCES OrgUnitType(org_unit_type_id),
  parent_org_unit_id NUMBER(10) NULL
  REFERENCES OrgUnit(org_unit_id),
  name VARCHAR2(200) NOT NULL,
  yearly_budget_k_eur NUMBER(12,0) NOT NULL
);

CREATE TABLE Employee (
  employee_id NUMBER(10) PRIMARY KEY,
  name VARCHAR2(200) NOT NULL,
  salary_eur NUMBER(12,2) NOT NULL CHECK (salary_eur >= 0),
  gender CHAR(1) NOT NULL CHECK (gender IN ('M','F','X')),
  birthdate DATE NOT NULL CHECK (birthdate <= TRUNC(SYSDATE))
);

CREATE TABLE EmployeeOrgAssignment (
  employee_id NUMBER(10) PRIMARY KEY
  REFERENCES Employee(employee_id),
  org_unit_id NUMBER(10) NOT NULL
  REFERENCES OrgUnit(org_unit_id)
);

```

Now:

- Divisions, product groups, departments are just OrgUnit rows with different OrgUnitType.
- Employees can sit at any level with one FK (org_unit_id), and future levels don't change EMPLOYEE or EMPLOYEE_ORG_ASSIGNMENT.

We see ChatGPT reverting to it's original, more abstract design but not discarding the unnecessary assignment table. Time to confront ChatGPT with the fact that the organization has a board.

You said:

Nice. I'll get back to this model in a moment. The serious design error I was referring to is that the first model made it impossible to register the company board and their supporting personnel.

ChatGPT said:

Exactly: that's a very concrete symptom of what's wrong with the first model.

In the original design:

- The world starts at Division.
- Every employee must sit in Division, ProductGroup or Department.
- The board + central staff sit "above" or "across" the four divisions, so they don't fit anywhere.

To make them fit, you would have had to:

- Fake a "fifth division" like Corporate, or
- Leave them unassigned (bad), or
- Start adding special cases in code.

That's precisely what the unified OrgUnit model fixes:

- Add an OrgUnitType like COMPANY (Survival Chemicals Inc.) and maybe CORP_SUPPORT.
- Create a root OrgUnit row for the company, children for the 4 divisions, then product groups, then departments.
- Board members and shared staff get EmployeeOrgAssignment.org_unit_id pointing to the company (or a corporate support unit).

So your observation about the board is a very clear indicator that the 3-table design didn't cover the full organizational reality.

The fixes to include the board are quite horrifying. We want to stay within the low abstraction framework but not go this way. So we force its hand by explicitly asking for a COMPANY table.

You said:

You are right that your second model is much more robust (although this table EmployeeOrgAssignment that is one-on-one with EMPLOYEE seems even stranger to me now). But I really think YOU made a serious design error as every normal company has a top management level. Why not have a table COMPANY in the first version of the model?

ChatGPT said:

Good question, and you're right to call it out.

1. Why no COMPANY table in the first model?
Because your initial description started the world at "4 divisions," I implicitly treated "company" as an external fact and modeled the tree from that level down. That is a mistake from an HR / master-data perspective: the top organizational node (the legal entity / company) almost always needs to be explicit so you can place the board and corporate staff somewhere real instead of faking a division.

So yes: a more correct "first" version in that style would have been:

- Company
- Division (FK → Company)
- ProductGroup (FK → Division)
- Department (FK → ProductGroup)

That would at least have given you a proper place for the board and corporate staff.

2. Why this still doesn't really fix robustness
Even with a Company table, the deeper problem remains:

- You are still hard-coding each level into its own table.
- Any new level ("Business Unit", "Team") or structural change still requires schema changes.
- You still cannot treat "org unit" generically in queries or constraints.

So: adding Company would have fixed the “no room for the board” bug, but not the structural fragility.

3. About the 1-to-1 EmployeeOrgAssignment

You’re right that, in the here-and-now, a 1-to-1 link table looks odd. Its purpose is:

- Keep Employee = “who this person is” (intrinsic attributes).
- Keep “where they sit in the org” in a separate relation that can later become 1-to-N or time-based without touching the person table.

If that feels like over-engineering for sprint 1, you can absolutely merge org_unit_id into Employee now and split it out later when you introduce history or multiple assignments. The generic OrgUnit model is the big robustness win; the 1-to-1 assignment table is a smaller, optional refinement.

So that’s settled...

You said:

OK. Before we move on to your second, more robust, version of the model let's first give the first version again, but with the EMPLOYEE table and with the improvements we agreed on.

ChatGPT said:

Here is the “first version” again, now with:

- COMPANY at the top level.
- EMPLOYEE assigned to exactly one level: company, division, product group, or department.

Create Table statement only shown for the new table COMPANY and the EMPLOYEE table that just got another reference to a new table COMPANY that can have one record.

```
CREATE TABLE Company (  
    company_id      NUMBER(10)      PRIMARY KEY,  
    name            VARCHAR2(200)    NOT NULL,  
    yearly_budget_k_eur NUMBER(12,0) NOT NULL  
);
```

```

CREATE TABLE Employee (
  employee_id      NUMBER(10)      PRIMARY KEY,
  name             VARCHAR2(200)    NOT NULL,
  salary_eur       NUMBER(12,2)     NOT NULL CHECK (salary_eur >= 0),
  gender           CHAR(1)          NOT NULL CHECK (gender IN ('M','F','X')),
  birthdate        DATE             NOT NULL CHECK (birthdate <= TRUNC(SYSDATE)),
  company_id       NUMBER(10)       NULL
    REFERENCES Company(company_id),
  division_id      NUMBER(10)       NULL
    REFERENCES Division(division_id),
  product_group_id NUMBER(10)       NULL
    REFERENCES ProductGroup(product_group_id),
  department_id    NUMBER(10)       NULL
    REFERENCES Department(department_id),
  CONSTRAINT chk_emp_one_org_level
    CHECK (
      (CASE WHEN company_id      IS NULL THEN 0 ELSE 1 END +
       CASE WHEN division_id     IS NULL THEN 0 ELSE 1 END +
       CASE WHEN product_group_id IS NULL THEN 0 ELSE 1 END +
       CASE WHEN department_id   IS NULL THEN 0 ELSE 1 END) = 1
    )
);

```

This is the “patched” first design: fixed levels (Company → Division → ProductGroup → Department) plus employees placed at exactly one of those levels.

You said:

OK. I accept this model, acknowledging the lack of robustness. But I want a COMPLETE model and this version still lacks a number of constraints. By this I mean that the model accepts certain data configurations that cannot exist in reality, given y description of the real world situation we intend to model.

If mankind would believe that the world is flat, Generative AI models like ChatGPT would use this erroneous idea. Likewise, the common misconception that a data model or a database design is just a set of data structures plus some limited set of constraints (like keys or data column formats) apparently leads ChatGPT to presume that it has generated a complete model. It has not! A model is not complete if it accepts data that cannot occur in the data configurations that the model must support. For example, we have stated that the model represents the organizational structure for a single company, which implies that the Company table must not have more than one record. This constraint on the model content is an integral part of the data model, no less so than the rule that each department must be subordinate to one product group.

ChatGPT said:

Good point. Let’s tighten this first model as far as we reasonably can in pure SQL, given your narrative.

Extra constraints we can (and should) encode

From your description we can safely enforce:

1. There is exactly one company and it is Survival Chemicals Inc.
→ At most 1 row, fixed ID and name.
2. There are at most 4 divisions and they must be Explosives, Catalysts, Coatings, Vitamins.
→ Division IDs restricted to 1–4, names restricted to that set, unique.
3. Names are unique in their natural scope.
 - Company name unique.
 - Division name unique in the company.
 - Product-group name unique within a division.
 - Department name unique within a product group.
4. Money values non-negative.
5. Exactly one org level per employee (already there).

Counts like “exactly 4 divisions exist”, “exactly 21 product groups”, “exactly 98 departments”, or “each division has ≥ 2 product groups” are not expressible cleanly in standard DDL; that would require triggers or data-governance outside the DB.

Now ChatGPT has really lost it!

Constraint #1 is correct. The model describes the structure of one specific company. Accepting more companies does not conform to the case and would certainly cause problems.

But the constraints listed under #2 are insane. Of course the organization structure can change in the sense that the number and relationships of divisions, product groups and departments can change over time. As long as the structure remains a hierarchy of no more than 4 levels there is no problem with the model.

Constraints #3 are interesting. It makes sense that names are not confusing but as far as the model is concerned, confusion cannot occur because the table and the ID column within that table identify an organizational unit. Normally such constraints are kept out of the model but ChatGPT is right.

Constraint #4 is OK and #5 indeed does not need to be specified.

Tightened DDL

Create Table statements lengthy and not especially interesting as we are interested in the more complex constraints that cannot be expressed by means of DDL.

If you want to go one step further (with triggers) to enforce “exactly 4 divisions” or “exactly 21 product groups,” we can do that next, but this is about as “complete” as you can get in pure relational DDL for the world you described.

You said:

Interesting, but this is really a case of over-engineering. The added 1 record only constraint on COMPANY is obviously necessary. The added constraints enforcing unique names are OK but will hardly ever be enforced in practice, because violation will not endanger database integrity (thanks to primary keys). The added constraints on minimal numbers of departments etc. are awkward, not

only for (again) the reason that "violation" will not affect database integrity, but also because changes can and will occur. We already noted that there are a lot of potential changes that cannot be supported by this admittedly non-robust model, but limiting changes supported by the model seems way over the top. Why not forbid all data modification after initial population. To conclude: the model needed only one additional constraint that. Agree?

ChatGPT said:

Thinking during 10s

Largely yes, with one nuance.

- The truly structural omission in the first model was:
→ "Exactly one company row" (or at least "at most one") so that the top node actually exists and is unique.
- The other things I added (fixed division names, fixed counts, etc.) are indeed over-engineering as integrity constraints; they are business rules that (a) can change and (b) don't threaten key-based integrity.
- I would still classify simple domain checks like salary ≥ 0 , budget ≥ 0 , birthdate \leq today as genuine integrity constraints, not over-engineering, because they prevent values that are impossible in your universe, independent of process.

So:

- For structure: yes, the only essential extra constraint was the "single company row".
- For data domains: I would still add basic CHECKs for non-negative money and valid dates.

You said:

I agree with your domain checks. The other remarks are noted.

A new Division or a department moves to another Product group really is NOT a 'business rule' violation. But then again, the term 'business rule' is so often misused that it must have entered the training data.

ChatGPT said:

Understood. If you want to proceed to the robust version (generic OrgUnit), or refine the employee-assignment strategy, let me know.

You said:

This is indeed what I want. But I want a more robust model that is fully equivalent with this final version of the non-robust model. By this I mean that your robust model must not accept data configurations that can obviously not occur in the real world when changes occur in the organizational structure. (If the model allows for changes that may realistically occur in the future that is of course fine. And, yes, I know that many of your initial observations about limited robustness (like storing the evolution of the organizational hierarchy) still stand. *And another thing: I am aware that some constraints still have to be implemented by means of database triggers and stored procedures. I am OK if you just express these as SQL statements on the database and disregard the extra complexities caused by lack of RDBMS modernization.*

ChatGPT said:

Here is a more robust model, equivalent in behavior to your final “non-robust” model, but with a generic org structure and explicit constraints so obviously impossible orgs are rejected.

1. Logical idea

- One generic table: OrgUnit (company, divisions, product groups, departments).
- A small lookup: OrgUnitType with 4 codes: COMPANY, DIVISION, PRODUCT_GROUP, DEPARTMENT.
- One company only, at the root.
- Fixed parent–child type pattern enforced by trigger:
 - COMPANY → children are DIVISION
 - DIVISION → children are PRODUCT_GROUP
 - PRODUCT_GROUP → children are DEPARTMENT
 - DEPARTMENT → no children
- Employees are assigned to exactly one OrgUnit (as before, but now via a single FK), with domain checks.

This allows future additions (e.g. add a type TEAM under DEPARTMENT later) without breaking the basic model.

2. Core tables (Oracle-style DDL)

<pre>sql -- 1) Lookup for org unit types CREATE TABLE OrgUnitType (code VARCHAR2(30) CONSTRAINT pk_org_unit_type PRIMARY KEY, description VARCHAR2(100) NOT NULL); INSERT INTO OrgUnitType (code, description) VALUES ('COMPANY', 'Top-level legal entity'); INSERT INTO OrgUnitType (code, description) VALUES ('DIVISION', 'Division'); INSERT INTO OrgUnitType (code, description) VALUES ('PRODUCT_GROUP', 'Product group'); INSERT INTO OrgUnitType (code, description) VALUES ('DEPARTMENT', 'Department'); -- 2) Generic org unit table CREATE TABLE OrgUnit (org_unit_id NUMBER(10) CONSTRAINT pk_org_unit PRIMARY KEY, org_unit_type_code VARCHAR2(30) NOT NULL CONSTRAINT fk_orgunit_type REFERENCES OrgUnitType(code), parent_org_unit_id NUMBER(10) NULL CONSTRAINT fk_orgunit_parent REFERENCES OrgUnit(org_unit_id), name VARCHAR2(200) NOT NULL, yearly_budget_k_eur NUMBER(12,0) NOT NULL CONSTRAINT chk_orgunit_budget CHECK (yearly_budget_k_eur >= 0));</pre>	<pre>-- 3) Employee with single current assignment CREATE TABLE Employee (employee_id NUMBER(10) CONSTRAINT pk_employee PRIMARY KEY, name VARCHAR2(200) NOT NULL, salary_eur NUMBER(12,2) NOT NULL CONSTRAINT chk_salary CHECK (salary_eur >= 0), gender CHAR(1) NOT NULL CONSTRAINT chk_gender CHECK (gender IN ('M','F','X')), birthdate DATE NOT NULL CONSTRAINT chk_birthdate CHECK (birthdate <= TRUNC(SYSDATE)), org_unit_id NUMBER(10) NOT NULL CONSTRAINT fk_employee_orgunit REFERENCES OrgUnit(org_unit_id));</pre>
---	---

So far: same semantics as your last model (exactly one org assignment per employee), but via a generic OrgUnit.

We have arrived at a more abstract model. What is nice is that ChatGPT has added a table OrgUnitType and also provided this table's content. This is really commendable because we aim for an equivalent model and this limits the possibilities to the four unit types in the previous design, while at the same time offering

flexibility for later changes. But unless other structures and/or constraints are added this model is still too permissive. For instance, it would be OK to have a division or the company itself report to a department, which would obviously be a no go in reality. That said, this setup does permit a flatter organization, for instance have a division flatten its structure by doing away with product groups and have its departments report to it directly. In the low abstract model this would force this division to work with a dummy Product Group, thereby opening a can of worms for software programs.

3. Constraints that enforce the real-world org shape

Here things get cumbersome. I told ChatGPT that I wanted single SQL statement to express constraints that cannot be part of the DDL. Instead, ChatGPT generates lengthy database triggers which I have to change back to easily accessible SQL statements. Perhaps I should have been more stringent.

3.1 Exactly one company, and it's the root

<p>Constraint 3.1.a</p> <p>A COMPANY must be the root (no parent)</p> <p>Violations</p> <pre> sql SELECT org_unit_id, org_unit_type_code, parent_org_unit_id FROM OrgUnit WHERE org_unit_type_code = 'COMPANY' AND parent_org_unit_id IS NOT NULL ORDER BY org_unit_id; </pre>	<p>Constraint 3.1.b</p> <p>There must be at most one COMPANY</p> <p>Violations</p> <p>(all COMPANY rows are returned if there is more than one)</p> <pre> sql SELECT org_unit_id, org_unit_type_code FROM OrgUnit WHERE org_unit_type_code = 'COMPANY' AND (SELECT COUNT(*) FROM OrgUnit WHERE org_unit_type_code = 'COMPANY') > 1 ORDER BY org_unit_id; </pre>
---	---

This ensures:

- There can't be two companies.
- The company can't have a parent.
- The company row corresponds to Survival Chemicals Inc., matching your earlier constraint.

Again, this is less than brilliant. It is not just that the constraint on the company name is unnecessary and unwanted (and not shown here). Also, a single SQL statement would be sufficient to enforce that there is one single company root unit and that all non-company units report to a superior unit.

3.2 Parent–child type pattern

ChatGPT generates four very similar constraints of which we show only two.

<p>Constraint 3.2.b</p> <p>Children of COMPANY must be DIVISION</p> <p>Violations</p> <pre>sql SELECT c.org_unit_id, c.org_unit_type_code, p.org_unit_id AS parent_org_unit_id, p.org_unit_type_code AS parent_type FROM OrgUnit c JOIN OrgUnit p ON p.org_unit_id = c.parent_org_unit_id WHERE p.org_unit_type_code = 'COMPANY' AND c.org_unit_type_code <> 'DIVISION' ORDER BY c.org_unit_id;</pre>	<p>Constraint 3.2.c</p> <p>Children of DIVISION must be PRODUCT_GROUP</p> <p>Violations</p> <pre>sql SELECT c.org_unit_id, c.org_unit_type_code, p.org_unit_id AS parent_org_unit_id, p.org_unit_type_code AS parent_type FROM OrgUnit c JOIN OrgUnit p ON p.org_unit_id = c.parent_org_unit_id WHERE p.org_unit_type_code = 'DIVISION' AND c.org_unit_type_code <> 'PRODUCT_GROUP' ORDER BY c.org_unit_id;</pre>
--	--

Effects:

- A DIVISION cannot appear at the top (must have parent of type COMPANY).
- A PRODUCT_GROUP cannot sit directly under COMPANY or under another product group, etc.
- A DEPARTMENT cannot be root or have children.
- No cycles are possible with these four types because types only “move down” the chain: COMPANY → DIVISION → PRODUCT_GROUP → DEPARTMENT.

This is exactly the logical structure you described, but now enforced:

COMPANY (Survival Chemicals Inc.) → multiple DIVISION org units (Explosives, Catalysts, Coatings, Vitamins, or others in the future) → multiple PRODUCT_GROUP units → multiple DEPARTMENT units

Employees can be attached to any of these units via the FK Employee.org_unit_id, just as in your non-robust model they could be attached to company/division/product group/department.

In a more abstract model that aims to be equivalent to a less abstract model information encoded in low-abstract data structures ('Company', 'Division', ...) comes back as table content in a more abstract table (here: OrgUnitType, see above). But referring to the content of these tables in code is a very bad practice as it undermines the added robustness and flexibility of this model. Specifically: it should be possible to add other types of organizational units and other relationships as long as the structure remains a hierarchy with one record of type 'Company' at the top. What ChatGPT does is half-baked. The model needs more abstract tables than just OrgUnitType and these constraints should be defines on the structure – not the content – of these additional structures.

You said:

I think you are right: This model is equivalent to the final model with a hard-coded hierarchy but is more robust, but only with respect to employee registration. As for the organizational structure part of the model (our teams sprint task!) it is still very rigid and actually a strange kind of hybrid. What really bugs me is a table with organizational unit types whose IDs are hard-coded in constraint expressions. Really awkward.

ChatGPT said:

Thinking during 32s

Agreed: that previous “robust” version was logically OK but stylistically ugly and too hybrid.

Below is a cleaner, more robust model that:

- Is equivalent to your final non-robust model.
- Has a generic OrgUnit table.
- Encodes the hierarchy rules in data (OrgUnitType metadata), not hard-coded in trigger logic.
- Still enforces “no impossible orgs” for the world you described.

You can later extend the hierarchy by changing data (rows in OrgUnitType), not code.

•
•
•
•
•

The session goes on from here but we have encountered the essential strong and weak points of ChatGPT 5.1 at this point.