Comparison of Database Table Counts in Popular Open-Source ERPs

1. **Odoo:**

- **Core (base install):** Roughly *~600 tables* in a minimal Odoo database. Even a small Odoo instance (only core modules) already contains “hundreds of tables,” including many internal metadata tables for UI, security, etc[1][2].
- **With Common Modules:** Enabling typical business modules (e.g. Sales, CRM, Inventory, Accounting) adds dozens of tables. For example, OpenERP 7 (old name for Odoo) had **about 600 tables** with standard modules installed[2]. Newer Odoo versions include even more apps, so an all-modules install can push well beyond 600 tables (approaching the high hundreds).
- **Total Estimated:** *~600–800 tables* (core + standard apps). Large Odoo deployments can approach this range, though exact count depends on installed modules.
- **Schema Notes:** Odoo uses a model-driven ORM (each model = one SQL table). The schema is highly modular – modules define new models (tables) or extend existing ones. New fields are added as real columns (Odoo doesn’t use generic EAV tables for custom fields). There are numerous meta-tables (`ir_model`, `ir_field`, `ir_ui_*` etc.) storing module and UI definitions. Transient models use temporary tables. Overall, the design is metadata-rich but still relational (no entity-attribute-value for business data)[3].

2. **ERPNext (Frappe):**

- **Core (base install):** The base Frappe framework plus ERPNext’s minimal set of modules create on the order of *several hundred tables*. (In Frappe, each *DocType* corresponds to a table, plus a few system tables for things like singletons and settings.)
- **With Common Modules:** A standard ERPNext deployment (which typically includes CRM, Sales, Buying, Stock, Accounting, HR, etc.) results in **~873 tables** in the database[4]. For instance, an out-of-the-box ERPNext v14 with the HR module enabled had 873 tables (about 852 of those correspond to DocType tables, plus some system tables)[4].
- **Total Estimated:** *~850–900+ tables* for a full ERPNext install with all official modules. (If fewer modules are used, the count would be lower – each module adds its own DocType tables.)
- **Schema Notes:** ERPNext is built on the Frappe framework, which is heavily metadata-driven. **DocTypes** (defined in JSON/YAML or via the UI) generate SQL tables (prefixed with `tab`). Each DocType = one table, and child tables (for one-to-many fields) are separate tables. “Single” DocTypes don’t get their own table – their fields are stored in a common `tabSingles` table[5]. Aside from a few system tables (`__Auth`, `__UserSettings`, etc. for internal use[6]), data is stored in these DocType tables. The design avoids runtime schema changes – new fields/DocTypes are added via migration patches, which create new tables/columns. No use of EAV; all fields are actual columns in tables.

3. **Dolibarr:**

- **Core (fresh install):** Dolibarr's database starts with on the order of *~150+ tables* when only core modules are enabled. (The official developer wiki indexes about 153 core tables[7], which likely correspond to the base modules.)
- **With Common Modules:** As modules for various domains are activated, the schema grows. By 2020, a Dolibarr instance with many common modules enabled had around **290 tables** in total[8]. Each functional module (CRM, Finance, HR, etc.) adds a set of tables; for example, product-related features alone use 20+ tables[9].
- **Total Estimated:** *~250–300 tables* with a full set of typical Dolibarr modules. (Dolibarr modules are optional, so smaller deployments can have fewer tables, while enabling all modules brings the count near 300.)
- **Schema Notes:** Dolibarr uses a classic relational schema with fixed tables per module. It is not ORM-driven – modules explicitly create tables (usually with prefix `llx_`). The schema is modular in that each module's tables are somewhat separate. Dolibarr also provides an **"extrafields"** mechanism: there is an `llx_extrafields` table and related structures[10] that allow adding custom fields to certain records without altering core table structures (an approach akin to a limited EAV for custom extensions). In general, however, most data is stored in dedicated tables for each business entity. The design is relatively straightforward, and adding a module involves creating new tables or adding columns via migration scripts.

4. **Apache OFBiz:**

- **Core (base framework):** OFBiz comes as a suite of applications; the default installation with all "out-of-the-box" components creates a very large schema – on the order of *several hundred tables*. In fact, an AWS team benchmarking OFBiz noted it has a **"complex schema: 837 tables"** (plus over 4,100 indexes) in the standard distribution[11]. This count includes all the base applications (accounting, order management, manufacturing, etc.) that ship with OFBiz.
- **With Common Modules:** OFBiz typically includes all its modules in one install (you don't separately add modules at runtime; they're part of the compiled system). So the *837 tables* can be seen as the total for the full system[11]. If some components are omitted, the count would be slightly lower, but generally an OFBiz deployment uses most components.
- **Total Estimated:** *~800+ tables* for a full OFBiz installation (the number can vary by version; e.g. OFBiz 18.12 had ~837 tables[11]).
- **Schema Notes:** OFBiz uses an **entity-engine** approach with XML entity model definitions. The schema is highly **metadata-driven** but *not* dynamic at runtime – the entity models (tables and relationships) are defined in XML files and translated into SQL tables on startup. Because OFBiz covers many domains, it has a very broad schema (e.g. `Order*` tables, `Product*` tables, `Party*` tables, etc. for each domain). The design is modular: each component has its set of entity definitions, but all are loaded into one unified database. OFBiz does not use an EAV model; rather, it defines concrete tables for each entity. There are some generic entities (e.g. `GenericValue` APIs and "extendable" entity patterns) but those still rely on real columns. The schema is fully relational with a large number of foreign keys linking the tables. The presence of an **application dictionary** (in XML) is similar in spirit to Compiere/ADempiere, but there's no separate in-DB schema registry – the XML is the source of truth.

5. **iDempiere:**

- **Core (base):** iDempiere (a fork/continuation of Compiere/ADempiere) has a very extensive core schema. A fresh iDempiere installation (which includes all base modules like financials, CRM, manufacturing, etc.) contains on the order of **800–900 tables**. For example, an iDempiere 9.0 schema report shows **879 tables** in the database[12] (not counting views). These encompass the Application Dictionary tables (prefix `AD_`), core business tables (`C_` for common, `M_` for material, etc.), and all default functionalities.
- **With Extensions:** iDempiere supports plugins that can add additional tables, but a typical "expanded" setup might include a few industry-specific plugins. The common practice, however, is that the base already includes most modules. So **~879 tables[12]** is already the total with core modules. Additional community plugins (if installed) could add a handful of tables each, pushing the count slightly higher (into the 900+ range).
- **Total Estimated:** *~880+ tables* for a comprehensive iDempiere system (core plus a few popular plugins). Without any extra plugins, high-800s is normal.
- **Schema Notes:** iDempiere's schema is **heavily metadata-driven via the Application Dictionary (AD)**. The AD_ *tables (like `AD_Table`, `AD_Column`, etc.) describe the data model and drive the UI. However, unlike Odoo/ERPNext, this metadata does not create new tables on the fly – it's used by the application to know about fields, windows, etc. The actual tables for business entities exist in the database and are defined by the core (e.g. `C_Invoice`, `M_Product`, etc.). iDempiere inherits Compiere's modular prefix convention (AD for dictionary, C for common, M for material, etc.[13]). The schema is modular in design (many tables, each focused on a specific entity), but all modules are part of one coherent data model. Customization is often done via metadata (adding columns through the dictionary, which then physically alters the table). There is minimal use of EAV – instead, new columns are added to the relevant table (the dictionary can automate this). One can also create custom tables via the dictionary. In summary, iDempiere/ADempiere have large,* fixed schemas* defined by metadata, with the flexibility to extend by adding columns/tables through that same mechanism.

6. **ADempiere:**

- **Core:** ADempiere (the predecessor of iDempiere) similarly has *hundreds of tables* in its default schema. While an exact count for a given version of ADempiere might be a bit lower than iDempiere's 879, it is of the same order of magnitude (likely **~600–800+ tables** in a full install). It includes the core Compiere tables plus enhancements. (For context, iDempiere 9.0 – which evolved from ADempiere – has ~879 tables[12], so ADempiere's count is in that ballpark.)
- **With Extensions:** ADempiere could be extended with industry-specific modules, but in practice most functionality was in the core. Thus, the total tables remain in the high hundreds. Any additional module (packed as an ADempiere customization) would add its tables via the application dictionary. The typical total would still be under ~900 tables.
- **Total Estimated:** *~700–800 tables* (estimate for ADempiere 3.x core plus common extensions). This is an approximation; the schema size is very close to iDempiere's, since iDempiere was built on ADempiere's schema.
- **Schema Notes:** ADempiere's database design is **identical in philosophy to iDempiere's** described above. It uses the **Application Dictionary** to define and manage the schema. All tables and columns are registered in metadata (in AD_Table, AD_Column, etc.), and the application UI is generated from this metadata. The schema itself is **modular by prefix** (e.g. tables for business partner, product, orders, etc., each prefixed by functional areas like `C_`, `M_`,

`HR_`, etc.[13]). The system is designed to allow adding new application functionality by inserting new metadata (which then corresponds to new physical tables or columns created in the DB). ADempiere does not use an EAV model; data is stored in structured tables. Custom fields are added by actually altering the table (often through the dictionary interface which then issues an `ALTER TABLE`). Because of this design, the schema is large but structured, and it remains **consistent and strongly typed** (at the cost of having so many tables).

7. **Openbravo:**

- **Core:** Openbravo ERP, like ADempiere, descended from Compiere, and it maintains a large schema defined by an application dictionary. A fresh Openbravo installation (with all base modules like Finance, Procurement, Sales, Production, etc.) has on the order of *several hundred tables (hundreds)*. Though an exact figure isn't published, it's in the same range as ADempiere's. (For example, Openbravo's schema has a similar breadth – including AD_ *metadata tables, and numerous business tables for its modules. It wouldn't be surprising if it's 600+ tables** in a typical version.)
- **With Modules:** Openbravo's modularity often comes in the form of additional industry modules or localization packs which can add tables. A fully "expanded" Openbravo (with popular add-ons like retail or vertical solutions) would likely push the count higher, perhaps into the **700+** range. The base distribution itself already includes most modules by default.
- **Total Estimated:** *~600–800 tables* in a comprehensive Openbravo setup. (In essence, Openbravo's scale is comparable to ADempiere/iDempiere – all are Compiere-lineage ERPs.)
- **Schema Notes:** Openbravo uses a **metadata-driven, modular schema** virtually identical in concept to ADempiere's. It has an Application Dictionary (in fact, many table names and concepts like `AD_Table`, `AD_Column` exist in Openbravo as well) for defining windows, fields, etc. The physical tables are created accordingly and prefixed similarly (Openbravo uses prefixes like `C_` for common entities, `M_` for material, etc., consistent with Compiere conventions). The schema is **fully relational** and quite normalized. Customizations or module additions in Openbravo are done via its module development pack – which involves defining new tables/columns in XML and then they get integrated (much like how core does it). There is no use of a generic EAV store – new data elements produce new tables or columns. Because Openbravo emphasizes modular extensions, it has a mechanism to safely add columns to core tables via modules (and these are tracked in the metadata). Overall, expect an **ADempiere-like schema architecture**: many tables, strict schema definitions, and heavy use of metadata to drive the application (with Hibernate as the ORM under the hood in later versions).

8. **Tryton:**

- **Core (minimal):** Tryton (which forked from OpenERP/TinyERP) starts with a very slim core. A fresh Tryton database with only the obligatory base modules (`ir` and `res` modules, which handle core config and user/party data) has only the fundamental tables (users, companies, etc.) – on the order of *tens of tables*, not hundreds. (Tryton's base is essentially just the framework; it requires installing modules for domain logic[14].)
- **With Common Modules:** Tryton's design encourages picking and choosing modules. Installing a typical set (Party/Contacts, Sales, Purchases, Invoicing, Accounting, Stock, etc.) will create a new table for each model in those modules. Each module might add a handful of tables. In a full-feature Tryton deployment (including most official modules), you can expect on the order of **a couple hundred tables**. (E.g. if ~40 modules are installed and each has ~3–5

tables on average, one might estimate ~120–200 tables total.) This is an estimate since Tryton's module count and coverage have grown over time. It is significantly *less* table-heavy than Odoo/ERPNext because it doesn't install everything by default – only what you choose.

- **Total Estimated:** *~100–200 tables* in a practical Tryton ERP database with a broad set of modules. Smaller deployments with just a few modules could be well under 100 tables, whereas an exhaustive deployment might exceed 200 if every possible official module is added.
- **Schema Notes:** Tryton uses an **ORM (Python SQLAlchemy)** and each model class corresponds 1-to-1 with a database table (just like Odoo) – *"all Tryton model classes are mapped one-to-one to a DB table"* (i.e. no table inheritance or polymorphic storage)[15]. The schema is **modular**: each module's Python code defines new Model classes, which the Tryton server translates into SQL tables upon installation. There is no central metadata registry in the database (models are defined in code, not via in-DB dictionary tables). Also, Tryton avoids using any generic key-value store for custom fields – if you need a new field, you typically write a module to add it (which will alter the table schema accordingly). Because of this philosophy, Tryton doesn't accumulate unused tables – you only get tables for modules you actually install. The schema itself is clean and normalized, similar to Odoo's (since they share an ancestor). No EAV patterns are used by default; data integrity is enforced via standard relational constraints. In summary, Tryton's schema grows with installed functionality but remains relatively lightweight and is entirely defined by the installed modules' code.

9. **webERP:**

- **Core (out-of-the-box):** webERP is a lightweight PHP-based ERP, and its database schema is relatively small. A fresh webERP installation creates on the order of *~50–80 tables* (much smaller than the others above). For example, an anecdotal discussion in the community mentioned around 60–70 tables making up the whole schema (webERP's focus is primarily accounting, inventory, and sales).
- **With Extensions:** webERP is not typically extended via plugins in the same way; it comes with a fixed set of features and corresponding tables. (There is a fork called FrontAccounting that's similar.) So, the "expanded" setup is essentially the same as the base – all standard webERP modules are usually part of the core package. The total number of tables thus remains in the few-dozen range.
- **Total Estimated:** *~70 tables* (approximately). This covers financials, AR/AP, inventory, sales orders, purchase orders, etc., which are the domains webERP supports. (For instance, tables like `chartmaster`, `gltrans` for GL; `stockmaster`, `stockmoves` for inventory; `debtortrans` for AR invoices, etc., each correspond to key records in the system.)
- **Schema Notes:** webERP's schema is **straightforward and fixed**. It's a collection of MySQL tables defined in SQL scripts (there is a `weberp-new.sql` with all CREATE TABLE statements[16]). The design is *not* metadata-driven – the tables are defined manually by the developers, and the PHP code directly queries them (no ORM). The schema is fully relational but simple: for example, fields are often denormalized for simplicity/performance, and a few master tables (like `config`) hold global settings. There is no concept of dynamic model extension or custom fields beyond editing the database directly. This simplicity makes webERP easy to modify by PHP developers (as noted in reviews: "*really simple implementation, meant for anybody to be able to modify it*"[17]). However, it's less flexible in-app: if you need a new field or table, you typically add it through a code/DB change. In summary, webERP has a **compact, static schema** that covers core ERP functions without a lot of extra metadata or fluff – which is by design to keep it lean.

10. **Metasfresh:**

- Core: Metasfresh is an open-source ERP that originated as a fork of ADempiere. As such, its base schema is very large and comparable to ADempiere/iDempiere. A fresh Metasfresh database will have on the order of several hundred tables (likely 800+), since it includes all the Adempiere-based dictionary tables and business entity tables. (While an exact count isn't published, being a direct descendant of ADempiere, it presumably has a similar count of tables – on the order of 800–900.)
- With Add-ons: Metasfresh includes all its major modules in the core distribution. It's not common to have "plug-ins" outside of what comes with it, aside from custom development. Thus, the expanded setup is essentially the default setup. If anything, Metasfresh has been actively developed and might have added some tables beyond the original ADempiere schema for new features, but also may have cleaned up others. The net count remains in the same high range.
- Total Estimated: ~800–900 tables in a Metasfresh database. It is one of the larger schemas, consistent with the Compiere-line ERPs. For practical comparison, if iDempiere is ~879 tables[12], Metasfresh will be of that order (give or take, depending on version).
- Schema Notes: Metasfresh sticks to the Application Dictionary architecture inherited from ADempiere. It has the same table prefixes (e.g. AD_ for metadata, C_ for common, M_ for material management, etc.), and the data model is defined in-DB in the AD tables as well as in XML migration scripts. Metasfresh's approach to schema is to use migration scripts (called "smart migrations") to modify the schema in a controlled way, always in line with its dictionary. There is no use of EAV; all fields are real columns. Metasfresh, being modern, has also introduced features like data partitioning and performance tweaks in the DB layer[18][19] (e.g., to cope with very large tables, since clients can have high transaction volumes). The modularity in metasfresh is mainly handled via the dictionary and modular code – you can add new tables by creating new dictionary entries and writing the corresponding code, similarly to how one would in ADempiere. Essentially, metasfresh's schema = ADempiere's schema (+ evolutions): a very large, metadata-driven, but static-at-runtime schema that covers a broad range of ERP functionality.

Sources: Primary sources include official documentation, community forum Q&As, and schema inspection reports for each project. For example, Odoo/OpenERP forum remarks on table counts[2], Frappe/ERPNext developer forum statistics[4], Dolibarr wiki and forums[8][7], an AWS blog for OFBiz[11], and SchemaSpy analysis for iDempiere[12] have been used to substantiate the numbers and architectural notes above. Each system's design philosophy (whether using an application dictionary, ORM, or simple SQL schema) is noted in the schema notes.

---

[1] [3] Visual database design / schema for Odoo? | Odoo

https://www.odoo.com/forum/help-1/visual-database-design-schema-for-odoo-168392

[2] I want to know which are the tables of the database of OpenERP 7 ? | Odoo

https://www.odoo.com/forum/help-1/i-want-to-know-which-are-the-tables-of-the-database-of-openerp-7-5463

[4] [5] [6] How many tables in db?! - Documentation - Frappe Forum

https://discuss.frappe.io/t/how-many-tables-in-db/134421

[7] [10] Category:Table SQL - Dolibarr ERP CRM Wiki

https://wiki.dolibarr.org/index.php/Category:Table_SQL

[8] Dolibarr db erp digrame - Feedback - Dolibarr international forum

https://www.dolibarr.org/forum/t/dolibarr-db-erp-digrame/18514

[9] Product with serial numbers - Developing for Dolibarr

https://www.dolibarr.org/forum/t/product-with-serial-numbers/20249

[11] Up and running with Apache OFBiz and Amazon Aurora DSQL | AWS Database Blog

https://aws.amazon.com/blogs/database/up-and-running-with-apache-ofbiz-and-amazon-aurora-dsql/

[12] SchemaSpy - idempiere9.adempiere

https://globalqss.com/idempiere/9_20211224/schemaspy/complete/

[13] Table Prefix - ADempiere

https://www.adempierebr.com/Table_Prefix

[14] Tryton 6.2.5 Installing All modules in windows 10 - Developer - Tryton Discussion

https://discuss.tryton.org/t/tryton-6-2-5-installing-all-modules-in-windows-10/5133

[15] Model for View table - Developer - Tryton Discussion

https://discuss.tryton.org/t/model-for-view-table/8583

[16] [17] Open Source ERP Review (as of May 2013) - Stack Overflow

https://stackoverflow.com/questions/16746660/open-source-erp-review-as-of-may-2013

[18] Data lifecycle management concept - metasfresh documentation

https://docs.metasfresh.org/pages/concepts/data_lifecycle_management.html

[19] Managing very large tables - System Administrator - Tryton Discussion

https://discuss.tryton.org/t/managing-very-large-tables/4407