ChatGPT

Cross-Theme Schema Overlaps and On-the-Fly Data Conflation in Overture Maps

Shared Schema Attributes Across Overture Themes

All Overture data themes share a set of core columns with identical names and purposes. These core fields include `id` (a unique feature identifier, often a GERS ID if applicable), `geometry` (GeoParquet WKB geometry), `bbox` (bounding box coordinates), `version` (an incrementing version number), `sources` (provenance of each attribute), and metadata fields `filename`, `theme`, and `type` [1] [2]. For example, both the Addresses and Places themes have columns `id`, `geometry`, `bbox`, `version`, `sources`, etc., serving the same roles [3] [4]. These common fields ensure a consistent baseline schema across themes and simplify multi-theme data handling.

Beyond those core columns, several thematic attributes appear (sometimes with slight variations) in multiple themes:

- **Name fields** – Many themes use a `names` **struct** to store feature names and translations. For instance, the Places, Buildings, and Divisions schemas all include a `names` container for primary and localized names [5]. (The Addresses theme is an exception, as address points typically have no proper name.) Consistent use of the `names` struct enables similar handling of feature names across places, buildings, and administrative divisions.

- **Subtype and Class** – The fields `subtype` and `class` are used in multiple themes to categorize features, though their meaning depends on context. For example, Buildings have a `subtype` indicating broad use (residential, commercial, etc.) and a `class` for specific building type [6]. The Transportation segments use `subtype` to distinguish road vs. rail vs. water, and `class` to denote the road's category in the network hierarchy [7]. Likewise, Divisions use `subtype` for administrative level (country, region, locality, etc.) and a `class` field to mark attributes like land vs. maritime or city size [8]. Even though the value domains differ (a road's class is not the same as a building's class), the reuse of these column names shows a schema pattern for "type and subtype" across themes.

- **Level and Hierarchy** – Some themes carry a `level` concept. In Buildings, `level` is a vertical stacking order (e.g. base level for parts) [9], and in the Base theme's infrastructure data, a similar `level` column exists (possibly reused for stacked infrastructure or layering) [10]. Divisions, instead of a simple level number, have hierarchical pointers like `parent_division_id` and a `hierarchies` struct to represent containment relationships [11] [12]. While not a direct name overlap, these all relate to *hierarchical structure* of features. Aligning these concepts (e.g. ensuring addresses or places can reference division hierarchies) could enhance conflation.

- **Geographic codes** – The `country` code column appears in themes that represent region-bound data. The Addresses theme includes a `country` ISO 3166-1 code for each address [3], and Divisions similarly include a `country` code for each admin unit (or its parent if the unit itself lacks one) [13]. By contrast, Places and Transportation do *not* have a top-level `country` field – a place's country is obtainable via its address sub-object, and a road segment's country can be inferred from its location or division references. The Divisions schema also has a `region` code (ISO 3166-2 for principal subdivisions) [14]. These fields are not present in all themes, but where they exist (Addresses, Divisions), they carry the same meaning. Consistent naming (`country`, `region`) allows straightforward joins or filters by country across those datasets.

- **Wikidata IDs** – Several themes include a `wikidata` field to link features to Wikidata entries. For example, Divisions have an optional `wikidata` ID column for political entities [15], and the Places theme's `brand` object also carries a `wikidata` ID for known brands [16]. This overlap, while not universal, is semantically similar where present (linking to external knowledge) and could be unified as a common enrichment across themes.

Table: Examples of Common Columns in Overture Themes

Column	Themes & Usage
`id`	**All themes:** Stable feature identifier (often GERS) [3] [4].
`geometry`	**All themes:** Geometry in WKB (Point for addresses/places [3] [4], Polygon/ LineString as appropriate in others).
`bbox`	**All themes:** Bounding box of the geometry [3] [17].
`version`	**All themes:** Integer version counter for updates [18] (tracks changes per release).
`sources`	**All themes:** Provenance of attributes (source dataset pointers) [18].
`name` / `names`	**Places, Buildings, Divisions, etc.:** Name struct with primary and alternate names [19]. *(Not in Addresses or basic Transport connector.)*
`subtype`	**Buildings:** broad building category [6]; **Divisions:** admin level type (country, locality, etc.) [20]; **Transport:** segment category (road/rail/water) [7]; **Base:** feature category (e.g., land vs water).
`class`	**Buildings:** specific building type [21]; **Divisions:** land vs maritime or settlement size class [8]; **Transport (roads):** road class/hierarchy (e.g., highway, local street) [7]; **Base:** subtype refinement (e.g., detailed land use class).
`country`	**Addresses:** ISO country code of address [3]; **Divisions:** ISO country code of the entity (or its root parent) [13]. *(Not in Places/Buildings natively.)*
`theme`, `type`, `filename`	**All themes:** Metadata columns indicating the dataset theme, feature type, and source file [2].

As seen above, many column names are literally duplicated across themes (especially the core schema), and several others are *conceptually* similar even if not in every dataset. This consistency is by design – for

example, every GeoParquet across Overture includes the `theme`, `type`, and `filename` fields to self-identify its origin [2]. Such overlaps can be leveraged to align and integrate data from different themes.

Semantically Equivalent Fields in Different Themes

In addition to exact name overlaps, there are cases of **semantic overlap** – fields that represent the same real-world information but are named or structured differently per theme. Identifying these can highlight opportunities for schema unification or on-the-fly conflation:

- **City/Locality Names:** An address's city is not stored in a single explicit `"city_name"` field in the Addresses theme. Instead, Overture addresses use an array `address_levels` for administrative divisions, with country-specific ordering. For example, in the U.S., `address_levels` has two entries: *[State, Municipality]* [22] (so the city name is the second element). In contrast, the Places theme (for points of interest) includes an `addresses` array of structs, each of which can have a `locality` field for city name, along with `region` (state/province code) and `country` [23]. The Divisions theme, on the other hand, explicitly models cities as features of subtype `locality` with their official name in the `names` struct [24] [25]. Semantically, *address city*, *place locality*, and *division locality* refer to the same concept – the city or town in which a point lies – but the data is organized differently. This suggests a conflation approach: if an address point is missing a city name, one could look up the containing `locality` division (by spatial join or parent ID) to retrieve that name. Likewise, if a place's address only gives a postal code and no city, the division hierarchy could supply the city. Recognizing that **Address.address_levels[municipality] ≈ Place.addresses.locality ≈ Division.name (subtype=locality)** is key to cross-theme enrichment.

- **Administrative Hierarchy:** The Addresses theme's `address_levels` array (up to 5 entries) captures multi-level admin units for an address [26], but it does so without naming each level (to remain flexible globally). In practice, these correspond to divisions like state, county, city, etc. The Divisions theme provides a rich `hierarchies` structure and `parent_division_id` to navigate those levels for any place [12]. A semantic match exists between *address_levels* and the division hierarchy: for example, an address in France might have `["Île-de-France", "Paris"] in address_levels, which maps to a Division of subtype region named "Île-de-France" and its child locality "Paris." **Bridging these schemas** (even if the column names differ) could allow automatic filling of missing admin levels. One could imagine enhancing the address schema with explicit fields or foreign keys (like` city_id `,` state_id` referencing Division IDs) rather than a free-form array – this would raise consistency, though it adds complexity. In the interim, tools can be built to interpret address_levels in context or use the postal code to query Divisions for a matching locality.

- **Postal Codes and Regions:** The Addresses theme has a `postcode` field [3] for ZIP or postal codes, whereas the Places theme does not break out postal code separately in the top-level schema (it might be embedded in the freeform address string). However, a Place's `addresses` entry could include a postal code if we extended the schema – the example didn't show one [23], but semantically it's expected. Similarly, **state/province** appears as an ISO code in Place addresses (`region`: e.g., `"US-NY"` for New York State [27]) and as the first element of `address_levels` in US addresses. These are the same piece of data (state/region of the address). Unifying naming (e.g., if addresses had a `region` column or places used address_levels) would make cross-theme usage easier. At the

very least, documenting the mapping (e.g., *address_levels[0]* = state = *places.address.region* for the US) allows users to script conflation.

- **Free-form vs Structured Address:** Places provide a `freeform` address text in each address object [23], whereas the Addresses theme splits `street`, `number`, and `unit` into separate fields [28] [29]. If a Place's address is missing some structured part (say it has a freeform "123 Main St" but no separate `number`), one could search the Addresses dataset for that same location to retrieve structured components. Conversely, an address point lacking a street name might be cross-verified against a place or division record with a known name. This highlights a semantic relationship: **Address.street + number ≈ Place.address.freeform**. A conflation tool could attempt to parse freeform addresses into structured fields or join on coordinates to combine info from both representations.

- **Feature Names vs Addresses:** The name of a division (city, county, etc.) is effectively the "official" value that might populate an address's city field. For example, the division with subtype `locality` named "New York" is what an address in New York City should list as its city. While one dataset stores it under `names.primary` [30] and another stores it in an address list, they are referring to the same entity. Similarly, a Place (POI) has its own `names` for the place's name, but might also carry an address that includes a city name – that city name should match a Division name. Ensuring consistency here (perhaps via **GERS IDs or cross-references**) would improve data quality. Today, Overture does not yet link address records or place addresses directly to division IDs, but doing so is a logical step.

- **Surface Type:** A more technical semantic overlap exists with things like road surface vs land cover. The Transportation schema has a `road_surface` property (under a struct) to denote pavement or dirt, etc. [31]. The Base theme for land cover has a `surface` field for material, and also a `land_cover` type classification. While these appear in different themes, they represent similar real-world attributes (ground or road material). Aligning their naming or values (e.g., using the same vocabulary for surface types) could allow a unified treatment of "surface" information for both roads and general land cover. This is a smaller semantic point, but in a graph model or combined schema, one could consider `surface` as a property that certain features have (roads, land, infrastructure) regardless of theme.

In summary, **semantic duplicates** like *city vs locality vs admin level 3*, *postal code vs postcode*, or *road_surface vs surface* indicate places where the schema could be standardized. For the user or data engineer, recognizing these equivalences means they can craft queries or conflation scripts to fill gaps. For example, if an Address is missing a locality name but has a postal code, one could use the postal code to find the corresponding Division locality name (or lookup in an external postal database) – thereby enriching the address data on the fly. Likewise, if a Places record lacks an explicit city field, the coordinate can be used to do a point-in-polygon query against Division boundaries to retrieve the city and state. These are feasible because the information *exists* in Overture, just in another theme or under a different label.

Graph Model for Cross-Theme Data Conflation (Using GERS and Relationships)

Considering the above overlaps, a **graph-based model** could be a powerful way to achieve real-time conflation across Overture's themes. In a graph, nodes could represent individual entities (addresses, places, buildings, roads, divisions, etc.), and edges could represent relationships or shared identifiers. Overture's **Global Entity Reference System (GERS)** is key here – GERS assigns stable IDs to real-world entities and is explicitly intended to facilitate cross-dataset linking [32] [33]. Each theme feature carries an `id` which, if the feature is part of GERS, includes a GERS UID (e.g., `"overture:places:place:12345"`). By using these as graph nodes or keys, one can link data that refer to the same entity:

- **Same-Entity Links:** If the same real-world entity appears in multiple themes, GERS is meant to give them a shared ID. For example, a building footprint in Buildings theme might get a GERS ID that also appears on a corresponding entry in an external dataset or potentially a Places feature if one represented the same building. (Currently an address and a building are considered different entities, so they wouldn't share an ID, but a building and a building part do have a relationship via `building_id`.) As GERS coverage grows, we expect more such links. In fact, the Overture team has mentioned plans to assign GERS IDs across themes like Transportation and Places, enabling use cases like "associating businesses with the buildings in which they operate" [34]. In a graph, this could mean an edge connecting a Place node (business) to a Building node if they share an address or location – possibly via a common GERS building ID or a spatial join.

- **Hierarchical Links:** Graph edges can naturally model hierarchy and containment. The Divisions theme already contains parent-child relationships (e.g., locality -> region -> country) via `parent_division_id` [12]. These could become edges in a graph (e.g., "Paris *is in* Île-de-France"). Similarly, one could create relationships like *Address -> Division (locality)* to denote that an address lies within a certain city. This link could be established by matching the address's locality name to a division name, or by a spatial containment test (point-in-polygon of division_area). Once established (even dynamically), it lets you traverse from an address node to its city node and retrieve city attributes (name, population, etc.) instantly. The graph could also link a division to its division_area polygon and division_boundary edges for more context.

- **Cross-theme Edges:** Some themes have explicit cross-links that can be graphed. For example, Buildings have `has_parts` and `building_id` fields linking a building to its building parts [35] [36] [37]. Transportation segments link to Connector nodes (intersections) via a list of connector IDs [38] – essentially a many-to-many graph between road segments and intersection points. These could be represented as edges in a network graph. In conflation terms, if you have a road segment and want to find the nearest address, a graph could traverse from a road segment to a connector (intersection) and then to other segments or address points near that node. While this is more relevant for routing, it shows how interconnected the data can be.

Using a graph model, **real-time conflation** becomes a query problem: missing attribute on one node? Traverse the graph to a connected node that has it. For example, consider an address point missing its city. In a graph where addresses are connected to the nearest locality division (or have an edge to a division node via shared GERS or spatial join), one could simply query the graph for the address's linked division and pull the name. This could be much faster than performing an on-the-fly spatial search each time, because

the relationship is pre-indexed in the graph. Another scenario: a Place (POI) might have an edge linking it to an Address entity (if Overture eventually provides a bridge between place addresses and the master address dataset via a common ID or explicit join). Then, any missing detail on the Place (say it doesn't list a postal code) could be obtained from the Address node it's linked to.

Graph traversal is generally efficient for these lookups. The **Global Entity Registry** behind GERS essentially operates like a graph database of all entities and their matches over releases [39] [33]. Overture's own materials highlight that GERS "provides an easy mechanism to conflate data from different data providers based on a specific GERS ID" [40]. In practice, one could construct a graph where GERS IDs are central nodes and theme-specific feature nodes link to that ID. Then conflation is literally a matter of following edges: for example, *address feature A – [GERS node 123] – place feature B* might indicate an address and a place share the same real-world location. A query could then join their properties without ever doing a spatial computation at query time.

It's worth noting that as of now, **not every cross-theme relationship is explicitly encoded**. Addresses and Places don't yet share common IDs (a place's address is given in text, not by referencing an address ID or division ID), so many edges must be inferred. But a graph approach could initially be used internally to **identify semantic overlaps** (e.g. link all fields named "name" across schemas as one concept node, link "locality" to "address_level[municipality]" nodes, etc., to visualize schema connectivity). This is more of a meta-graph of the schema itself, useful for planning schema changes. On the data side, a property graph or knowledge graph integrating all Overture themes would be a powerful tool for both conflation and user-facing queries that span multiple layers (imagine asking "find all restaurants (Places) in cities (Divisions) with population over 1M (Divisions), and include the building height (Buildings) of the building they occupy if available"). This kind of query is complex in a traditional relational setup but could be answered if the relationships are encoded via GERS and graph traversal.

In summary, **using a graph model** (with GERS IDs as the linkage and additional spatial or semantic edges as needed) appears highly feasible and valuable for conflation. It would allow on-the-fly enrichment of one theme's data with another's attributes by traversing connections, rather than having to constantly perform joins or lookups from scratch. This aligns with Overture's long-term vision: as more features across themes get GERS IDs (and thus can be directly joined) and as more relationships (like building-contains-place or address-in-division) are catalogued, the data essentially forms a big graph of the world's map features. Real-time conflation is then simply graph navigation.

Schema Alignment and GeoParquet Considerations

Apart from a graph database approach, there are practical steps in the data format itself that could facilitate easier conflation by end-users. **GeoParquet**, being a columnar format, allows efficient selective reads of columns and is well-suited for distributed storage by spatial partitions. How the schemas align can impact the ease of merging data:

- **Column Name Alignment:** If the same or analogous attribute has the same column name across themes, users can more easily write one query to pull that field from multiple tables. For instance, having a `country` column in both Addresses and Divisions means a user can filter both datasets by `country='US'` consistently [3] [13]. Conversely, where names differ (e.g. `locality` vs an unnamed address level), extra manipulation is needed. One suggestion raised is whether **unifying column names or positions** across themes could help. Using consistent names for semantically

identical fields is definitely recommended – e.g., if a future schema update introduced an explicit `city_name` field in Addresses (populated from address_levels or divisions), and likewise ensured Places addresses use the same key for city, then conflation becomes a straightforward join on `city_name`. This could also enable Overture's pipeline to automatically fill those via lookup. So, aligning schemas by **name** is the clearer path (and could be proposed to the schema team to improve data quality by reducing ambiguity).

- **Column Order/Index in Parquet:** The user inquired whether having shared fields in the *same column index* across Parquet files would allow faster conflation. Parquet stores data in columns internally, and while the physical column index matters for how data is laid out, most query engines use column *names* to align data from different files. There isn't a direct concept of "same position = same meaning" unless the schemas are identical. However, if one were to create a unified schema superset, and ensure each theme's Parquet files adhere to that ordering (with nulls for fields not used), then theoretically one could treat all themes in a single dataframe or table. For example, one could design a schema with columns: id, geometry, bbox, country, postcode, street, number, unit, locality, region, name, subtype, class, etc., covering every possible field from all themes. Each theme would populate only relevant columns. Then a user could load all theme files together (since column structure matches) and filter by theme = 'addresses' or 'places' as needed. In such a unified scenario, "city name" might always reside in, say, column position 10, making it trivial to select that column across datasets in a columnar read. **However,** this approach can be cumbersome (many nulls for unrelated fields) and is not how Overture currently structures the data (they keep themes separate).

  A more practical interpretation is that *if* schemas are similar, tools like Apache Arrow or DuckDB can do a **union** or **join** without schema reconciliation overhead. We already see that many columns line up one-to-one (id, geometry, bbox, etc.), especially among themes with similar feature types. For example, the Buildings and Building_Parts have very similar schemas, enabling them to be read together easily [41] [42]. If Addresses and Places had a few more columns in common (like an explicit city field), one could imagine a union or join operation where those fields align naturally. In contrast, when one field is buried in an array or nested struct (like address_levels vs locality), it's harder to combine on the fly.

- **Spatial Partitioning:** Overture's data is partitioned by geographic chunks (likely tiles or H3 hex cells) for distribution [43] [44]. This means that for a given region (bounding box), one can retrieve the relevant subset of each theme's data relatively easily (each theme's files are organized by `theme=.../x.parquet` with spatial indices). If a user downloads the same area from Addresses and from Divisions, for instance, the data can be joined by location after the fact. The **GeoParquet spatial indexing** can speed up these queries (some engines can do predicate pushdown on geometry). While this doesn't directly conflate by attribute, it ensures that cross-theme conflation can be done within the same spatial window. For example, one could use a bounding box filter to load all addresses and all places in a city, then perform a join in-memory to link those that are nearest or share coordinates. The partitioning scheme is crucial: if both themes use the same tiling or index system, the "slice" of data you get for a region will naturally contain the pieces to conflate. Overture documentation suggests using bounding boxes and tools (DuckDB, etc.) to pull data for a specific area [45] [46] – once you have that, you have all themes' data for that area ready to cross-reference.

- **On-the-fly Enrichment Tools:** Until the schema itself evolves, one approach to deliver value is building an **enrichment layer or tool**. For instance, a Python or DuckDB script could take an Addresses dataset and add city names by consulting the Divisions dataset (either via spatial join or a prepared lookup table from division points/polygons). This could be packaged for users who need conflated data "on their machines." Because Parquet is columnar, reading just the needed columns (e.g., address id, geom, postcode, plus division id and name) is efficient. The user could choose to run this step to get a more valuable combined dataset. Essentially, we'd be offering a **derived product** that merges themes. If done at query time, it leverages the efficient columnar IO and any available indices. If done as a preprocessing step, it could be provided in a new release.

In conclusion, while **column index alignment** in Parquet is a niche consideration (most benefits can be achieved simply by using consistent column *names* across themes, which we should advocate), the columnar nature of GeoParquet is definitely an advantage for conflation. It allows us to fetch slices of data fast and join on either keys or spatial coordinates as needed. By ensuring that overlapping fields either share names or are documented as equivalent, we make it easier for both Overture's engineers and end-users to perform these joins. Given the points above, a reasonable recommendation to the schema team might be:

- Introduce explicit fields for common address attributes (e.g., `locality` (city), `region` (state/province)) in the Address theme, or at least consistently in the Place address object, so that Addresses, Places, and Divisions can speak the same language for those attributes. This semantic alignment would increase data quality and reduce missing info across themes.
- Leverage GERS IDs to directly link features where applicable (for example, if a Place is located in a Building, record the building's GERS id in the Place entry). This would eliminate guesswork in conflation by providing a direct key join. The ArcGIS Overture blog demonstrated enriching buildings with external data via GERS joins, with great success [47] [48], and suggested future expansions of GERS to other themes [34].
- Provide "bridge" files or relationship tables that map, say, address points to their containing division IDs (essentially a precomputed lookup of which city/state each address belongs to). This would be extremely useful for on-the-fly conflation, as the user could just join on the stable IDs rather than doing spatial computations. Overture already releases bridge files for matching source IDs to GERS [49]; similarly, an *admin bridge* (address ID -> division ID) could be derived.
- Continue to maintain the core schema consistency (id, geometry, etc.) and extend that philosophy to any new common fields that might emerge (for example, if adding an `updated_at` timestamp or similar, do it uniformly).

By pursuing these improvements, the Overture dataset's **value can be significantly raised**. Users would gain the ability to dynamically conflate data from different themes with minimal effort, either through well-chosen schema design or through a graph of GERS-linked entities. In the meantime, with the current datasets, we can still achieve conflation by carefully using the overlapping fields and external logic. Whether for internal pipeline enhancements or end-user tools, the ideas discussed here – schema unification, graph relationships, and partition-aligned joins – all aim at the same goal: **delivering a richer, more connected map dataset on the fly**, so that no useful piece of information remains siloed in one theme when it could complement another.

Sources

- Overture Maps Addresses Schema & Guide [3] [1] [22]
- Overture Maps Places Schema & Guide [4] [50] [23]
- Overture Maps Buildings Schema & Guide [6] [21]
- Overture Maps Divisions Schema & Guide [8] [51] [25]
- Overture Maps Transportation Schema & Guide [7] [38]
- Overture Maps Base Theme Schema [52] [53]
- *Understanding Overture's Global Entity Reference System (GERS)* [32] [40]
- *Esri ArcGIS Blog – Enriching Overture Data with GERS* (Use cases for conflation) [47] [34]

---

[1] [3] [22] [26] [43] Addresses | Overture Maps Documentation
https://docs.overturemaps.org/guides/addresses/

[2] [4] [18] [44] [45] [46] [50] Places | Overture Maps Documentation
https://docs.overturemaps.org/guides/places/

[5] [8] [11] [12] [13] [14] [15] [19] [20] [25] [30] [51] Divisions | Overture Maps Documentation
https://docs.overturemaps.org/guides/divisions/

[6] [9] [21] [36] [37] [41] [42] Buildings | Overture Maps Documentation
https://docs.overturemaps.org/guides/buildings/

[7] [17] [31] [38] Transportation | Overture Maps Documentation
https://docs.overturemaps.org/guides/transportation/

[10] [52] [53] Base | Overture Maps Documentation
https://docs.overturemaps.org/guides/base/

[16] [23] [27] place | Overture Maps Documentation
https://docs.overturemaps.org/schema/reference/places/place/

[24] division | Overture Maps Documentation
https://docs.overturemaps.org/schema/reference/divisions/division/

[28] [29] address | Overture Maps Documentation
https://docs.overturemaps.org/schema/reference/addresses/address/

[32] [33] [39] [49] What is GERS? | Overture Maps Documentation
https://docs.overturemaps.org/gers/

[34] [40] [47] [48] Enriching Overture Data with GERS
https://www.esri.com/arcgis-blog/products/arcgis-online/mapping/enriching-overture-data-with-gers

[35] building | Overture Maps Documentation
https://docs.overturemaps.org/schema/reference/buildings/building/