BEYOND GRAPHS: LEARNING WITH RELATIONAL DBS

Anonymous authors
Paper under double-blind review

ABSTRACT

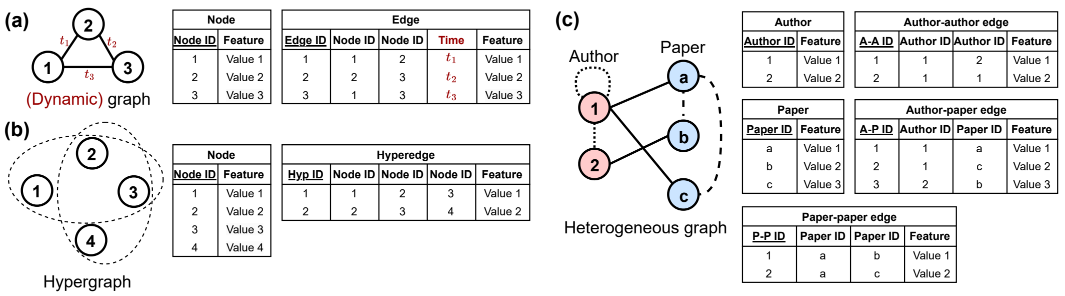
Despite recent advancements in representation learning on graphs, there still lacks a unified framework that addresses the challenges of learning from real-world relational data, which can involve heterogeneous, dynamic, and multi-ary relationships. Existing efforts focus on extending graph learning to alternative graph representations, including heterogeneous graphs and hypergraphs; however, these approaches are highly specific to particular use cases, therefore introducing complexity into their application and deployment. We propose to unify and extend existing graph learning research with relational databases (RDBs). RDBs, characterized by their simplicity and versatility, consist of multiple tables linked by shared key columns. We show that diverse types of graphs can be unified as RDBs and different graph learning tasks can be formulated as predicting column values in RDB tables. Furthermore, we introduce Relational Database Neural Networks (RDNNs), the first family of deep learning models that can holistically learn from multi-table information inside a relational DB, without the need for converting it to graphs. RDNNs provide a more flexible and comprehensive deep learning design space for modeling relational data, capable of solving problems beyond the scope of graph learning. Through extensive experimentation on a range of graph and multi-table datasets, we demonstrate that the RDNNs offer competitive or superior performance in comparison to Graph Neural Networks (GNNs) on graph learning tasks and tabular machine learning methods on RDB prediction tasks.

1 INTRODUCTION

The field of graph representation learning has greatly advanced in recent years, showing promising applications in industrial and scientific applications (Zitnik & Leskovec, 2017; Jin et al., 2018; Dauparas et al., 2022). Due to the complex relationships in real-world data, researchers have explored machine learning beyond classic graphs[1] (Kipf & Welling, 2017), including multigraphs (Butler et al., 2023), dynamic graphs (Pareja et al., 2020), heterogeneous graphs (Schlichtkrull et al., 2018), and hypergraphs (Feng et al., 2019). However, these graph representations still carry distinct assumptions about the type of relational data that they can model, and therefore, fail to fully capture the diverse relational information that pervades real-world data. Notably, this limitation of graph representation not only prevents researchers from generalizing their method to other types of relational data but also leaves practitioners perplexed in selecting the optimal representation for their data.

Stepping back, we realize that graphs are not the sole representation of relational data; instead, relational databases (RDBs), consisting of multiple tables, have been the gold standard for representing relational data (Codd, 1970). In fact, the abovementioned graph representations can all be considered special cases of RDBs. Despite its generality, RDBs have received much less attention from the machine learning community and have not yet fully benefited from the recent advancement of deep learning research. A possible reason is that RDBs create strict constraints on the underlying data, *e.g.*, entity and referential integrity constraints (Grefen & Apers, 1993), although essential for relational database management systems (RDBMS)[2], making the representation less flexible for deep learning models. The complexity of RDBMS interface, which requires SQL programming, further makes it complex to integrate with modern deep learning systems.

[1]Also known as simple graph, undirected graph, or homogeneous graph.
[2]RDB is a data representation. Not to be confused with RDBMS, which is a system that manages RDB.



Node

Node ID	Feature
1	Value 1
2	Value 2
3	Value 3

Edge

Edge ID	Node ID	Node ID	Time	Feature
1	1	2	t_1	Value 1
2	2	3	t_2	Value 2
3	1	3	t_3	Value 3

Node

Node ID	Feature
1	Value 1
2	Value 2
3	Value 3
4	Value 4

Hyperedge

Hyp ID	Node ID	Node ID	Node ID	Feature
1	1	2	3	Value 1
2	2	3	4	Value 2

Author

Author ID	Feature
1	Value 1
2	Value 2

Author-author edge

A-A ID	Author ID	Author ID	Feature
1	1	2	Value 1
2	1	1	Value 2

Paper

Paper ID	Feature
a	Value 1
b	Value 2
c	Value 3

Author-paper edge

A-P ID	Author ID	Paper ID	Feature
1	1	a	Value 1
2	1	c	Value 2
3	2	b	Value 3

Paper-paper edge

P-P ID	Paper ID	Paper ID	Feature
1	a	b	Value 1
2	a	c	Value 2

Figure 1: **Unifying diverse graph representations into RDBs**. We show that **(a)** (dynamic) graphs, **(b)** hypergraphs, and **(c)** heterogeneous graphs can be succinctly represented as RDBs. Columns with underscores in the "header" indicate that itis a primary key column in the table. For clarity of presentation, we omit the curves for indicating primary-foreign key linkages which are commonly presented in relational DB diagrams. The linkage can be easily inferred based on matching the key column names.

Consequently, existing works either focus on learning from a single table (Chen & Guestrin, 2016; Ke et al., 2017), potentially with additional merges from multiple tables into one table using SQL join (Chaudhuri et al., 1999), or covert relational DB to graph representations and apply graph machine learning (Cvitkovic, 2020; Bai et al., 2021) at the cost of loss of information and computational overhead. Evidently, these approaches are not ideal due to the additional human efforts from skilled data engineers and computational overhead. Developing deep learning approaches native to relational databases not only circumvents the limitations of graph machine learning but also revolutionizes our capability to organize, learn, and predict with databases.

In this paper, we propose to unify and extend existing graph learning research by representing graphs as relational databases (RDBs). As is shown in Figure 1, our key observation is that diverse types of graphs, *e.g.*, simple graphs, heterogeneous graphs, and hypergraphs, can be represented as multiple tables with shared key columns. Notably, our proposed RDB representation of graphs abstracts away the concept of a node, an edge, and a hyperedge; instead, they all correspond to a row in a given table, where the table has 1, 2, or multiple key columns respectively. Consequently, researchers and practitioners no longer need to worry about properly defining nodes, edges, and hyperedges from their data; instead, they only need to prepare their data into multiple tables and declare columns that refer to the same semantic meaning, *i.e.*, keys. Moreover, our proposed RDB representation further unifies and simplifies different graph learning tasks, *e.g.*, node/edge/graph-level tasks, into column value prediction tasks in (different) tables.

Based on this perspective, we propose Relational Database Neural Networks (RDNNs), the first family of deep learning models that can holistically learn from multi-table information inside a relational DB, without the need of converting them to graphs. Notably, RDNNs can support general multi-table data, which do not need to follow the constraints in conventional RDB (Section 4.1 and Figure 6). Each layer in RDNNs defines a 2-stage computation: cross-table communication and inner-table transformation. Overall, compared to designing specialized GNNs such as heterogeneous GNNs and hypergraph GNNs, RDNN provides a more flexible and general neural architecture design space for relational data.

We implement a simple instantiation of RDNN and validate its effectiveness on a variety of datasets and tasks, ranging from diverse types of graphs to real-world relational DB datasets. We demonstrate that RDNNs are comparable with GNNs on various graph datasets and node/edge/graph-level graph learning tasks, and out-performs GNNs when learning on sophisticated heterogeneous graphs. We then show RDNNs demonstrate superior performance compared to existing machine learning methods for relational DBs. In summary, by proposing RDNNs that can be conveniently applied to generic relational data, we hope to circumvent the limitations in graph machine learning and pave the path for a new machine learning frontier of deep learning for databases.

2 Related Work

Deep learning with generalized graphs. GNNs Scarselli et al. (2008); Kipf & Welling (2017) are originally defined over simple/homogenous graphs, where a pair of nodes are connected with edges, without node/edge type information. Many research works have extended GNNs beyond simple graph representations, such as heterogeneous graphs Schlichtkrull et al. (2018); Wang et al. (2019); Hu et al. (2020b), hypergraphs Feng et al. (2019); Huang & Yang (2021), and dynamic graphs Pareja et al. (2020); You et al. (2022). Although these GNNs have achieved amazing results under their corresponding graph representations, they do not apply to other types of graphs or generic relational data such as RDBs. Our proposed RDNNs apply to diverse graph representations and generic relational data as well.

Machine learning for RDBs. Existing works focus on machine learning from a single table, such as gradient boosting decision trees Chen & Guestrin (2016); Ke et al. (2017) and neural networks Arik & Pfister (2021). However, generalizing these methods to multiple tables requires manually joining different tables and is bounded by the cost and complexity of table joins. There are recent works on learning from multiple tables, but they require first converting databases to graphs Cvitkovic (2020), heterogeneous hypergraphs Bai et al. (2021), or text data Arora et al. (2021), then applying existing deep learning methods such as GNNs and BERT. The additional conversion from RDBs to other representations creates additional complexity, loss of information, and computation overhead. In contrast, RDNN is a novel neural network architecture that natively builds on multi-table representations, capable of automatically learning from multi-table information in an RDB.

Deep learning for RDB management systems. Existing works have also explored using neural network's predictive power to optimize for RDB management systems, such as buffer tuning Tan et al. (2019), view selection Yuan et al. (2020); Lan et al. (2020), plan enumerator Marcus & Papaemmanouil (2018), and cardinality estimation Kipf et al. (2018); Park et al. (2020). Orthogonal to this line of work where deep learning is developed for optimizing RDB/RDBMS, our work extends the capability of deep learning by enabling it to take RDBs as a new type of input data modality.

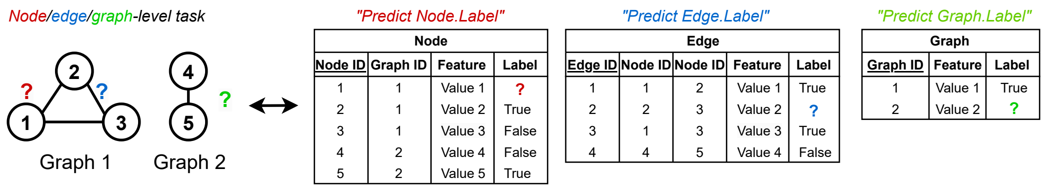
3 Preliminaries and Perspectives

Relational databases. An RDB consists of a set of tables $\mathcal{T} = \{T_t\}$, each table $T_t \in \mathbb{R}^{n \times c}$ has n rows and c columns, where $T_t[r, :]$ represents the r-th row and $T_t[:, c]$ represents the c-th column. Given a table T_t, some columns are defined as keys, where a primary key $T_t[:, c_{\text{pkey}}]$ has unique values in each row, and foreign keys $T_t[:, c_{\text{fkey}}]$ can be linked to primary keys in other tables within the RDB. Next, we show how different graph representations and prediction tasks can be unified with this succinct representation. Figure 1 and 2 provide concrete visualized examples.

Graphs as RDBs. A graph is commonly represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_i\}$ is the node set and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the edge set. Each node can have d_{node}-dim feature, and each edge can have d_{edge}-dim feature. From an RDB perspective, a graph $\mathcal{G}$ can be represented with two tables, T_{node} and T_{edge}. T_{node} has $|\mathcal{V}|$ rows, 1 primary key column c_{node}, and d_{node} number of feature columns. T_{edge} has $|\mathcal{E}|$ rows, 1 primary key column c_{edge}, 2 foreign key columns that link to c_{node}, and d_{edge} number of feature columns.

Heterogeneous graphs as RDBs. Heterogeneous graphs introduce node types and edge types to the graph definitions. Specifically, each node v is mapped to a node type with $\phi(v) : \mathcal{V} \to \mathcal{A}$, where there are $|\mathcal{A}|$ node types in total, and each edge e that connects node v_i and v_j is mapped to an edge type $\phi(v_i, e, v_j) : \mathcal{E} \to \mathcal{R}$, where there are $|\mathcal{R}|$ edge types in total. Although it is nontrivial to manage a heterogeneous graph, it can be succinctly described by an RDB with $|\mathcal{A}|$ node tables for each node type and $\mathcal{R}$ edge types for each edge type. Similar to simple graphs, each node table represents each node as a row, with 1 primary key column and additional feature columns; each edge table represents each edge as a row, with 1 primary key column, 2 foreign key columns, and additional feature columns; note that different node/edge tables often have different shapes.

Hypergraphs as RDBs. A hypergraph further generalizes the definition of edge set $\mathcal{E}$ from connecting pairs of nodes to any number of nodes. In practice, researchers often focus on k-uniform hypergraph, where all the hyperedges connect to k nodes. In this case, a k-uniform hypergraph can be represented as 2 tables, T_{node} and $T_{\text{hyperedge}}$. Compared to converting a simple graph to RDB, T_{node} has an identical definition, while table $T_{\text{hyperedge}}$ has k foreign key columns that link to c_{node}. A general hypergraph, where a hyperedge can connect an arbitrary number of nodes, can be equivalently represented as



Node			
Node ID	Graph ID	Feature	Label
1	1	Value 1	?
2	1	Value 2	True
3	1	Value 3	False
4	2	Value 4	False
5	2	Value 5	True

Edge				
Edge ID	Node ID	Node ID	Feature	Label
1	1	2	Value 1	True
2	2	3	Value 2	?
3	1	3	Value 3	True
4	4	5	Value 4	False

Graph		
Graph ID	Feature	Label
1	Value 1	True
2	Value 2	?

Figure 2: **Unifying prediction tasks on graphs into RDB column prediction task**. Node, edge, and graph level prediction tasks on graphs can be equivalently converted to predicting missing column values in node, edge, and graph tables in an RDB.

a heterogeneous graph with 2 node types (node and hyperedge) and 1 edge type (describing the incidence between hyperedges and the nodes), therefore, it can be succinctly represented as 3 tables.

Dynamic graphs as RDBs. A common approach to representing a dynamic graph is to associate a time stamp with each node/edge/hyperedge. For RDBs, it corresponds to simply adding a time column for each node/edge/hyperedge table. In the case where nodes and edges may appear and disappear in a graph, we could use 2 time columns to record the time period when a node/edge exists.

Graph learning tasks as predicting RDB column values. In addition to unifying diverse graph representations into RDBs, we further show that prediction tasks on graphs, either at node, edge, or graph level, can be equivalently converted into predicting column values in an RDB, illustrated in Figure 2. Concretely, a node-level prediction task associates each node v with node label y_v, an edge-level task associates each edge e with label y_e, and a graph-level task associates each graph $\mathcal{G}$ with label $y_{\mathcal{G}}$; these labels y_v, y_e, and $y_{\mathcal{G}}$ can then be conveniently represented as columns in node, edge, and graph tables, respectively. Note that when multiple graphs are present in a dataset, we can introduce a graph table T_{graph}, whose primary key column c_{node} is referred to as an additional foreign key column in node table T_{node} (and/or T_{edge}).

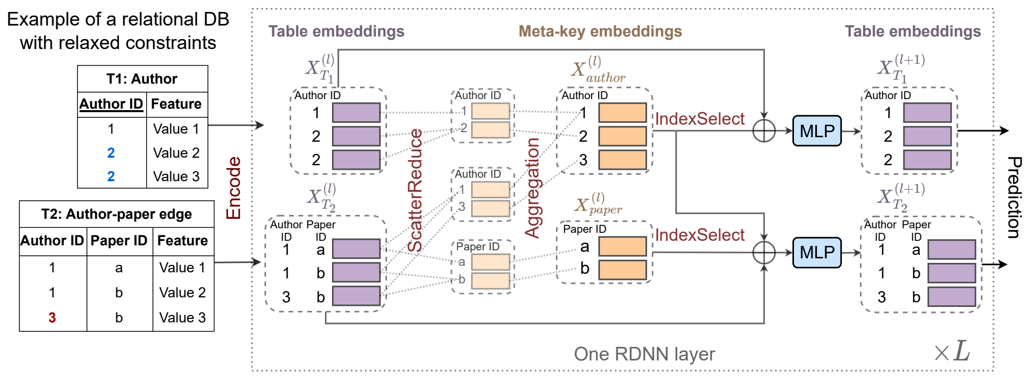
Perspective: from analytical to predictive queries on RDBs. Demonstrating the expressiveness of column prediction tasks on RDBs has a profound impact beyond just unifying graph prediction tasks. Currently, the majority of user interactions with RDBs rely on SQL queries, which focus on processing and analyzing the existing information in a given RDB. As is shown in Figure 2, users could write generic declarative predictive queries, *e.g.*, "`Predict Node.Label`", to define any column prediction tasks on RDBs. Such predictive queries provide users with an intuitive yet powerful interface to predict unknown/future information based on an RDB, beyond just analyzing existing information from RDB with SQL queries. In sum, predictive queries have the potential to *revolutionize the interaction with RDBs* and the *application/deployment of machine learning algorithms*. Properly instantiating these predictive queries requires powerful machine models that can holistically leverage information in RDBs; our solution, RDNNs, will be discussed in Section 4.

Summary: why viewing graphs as RDBs. The discussions above have clearly demonstrated that diverse graph representations can be unified as RDBs which are succinct and expressive, and prediction tasks on graphs can be further unified as column prediction tasks on RDBs. By developing RDNNs, the first family of neural networks native to RDBs (Section 4), we refrain from developing specialized GNNs for each type of graph and each type of machine learning task; instead, we can learn from all the diverse types of graphs and generic RDBs with the same neural network architecture, with the potential of revolutionizing how users interact with RDBs.

As a concrete example, we could easily define a "dynamic heterogenous hypergraph" from real-world data, but managing it with graph representations would be highly non-trivial, let alone developing a specialized GNN that can learn from the graph. In contrast, by representing it as an RDB, it is merely an RDB with a few more tables than what we have discussed above; in real-world scenarios, it is not uncommon to work with RDBs with hundreds of tables.

4 Relational Database Neural Networks

Next, we introduce Relational Database Neural Networks (RDNN), the first deep learning architecture native to RDBs without the need for converting RDBs to other data representations. An RDNN



T1: Author	
Author ID	Feature
1	Value 1
2	Value 2
2	Value 3

T2: Author-paper edge		
Author ID	Paper ID	Feature
1	a	Value 1
1	b	Value 2
3	b	Value 3

Figure 3: **Illustration of RDNN architecture.** The input tables are transformed to initial table embeddings $X_{T_t}^{(0)}$ through an encoder layer. Each RDNN layer comprises a cross-table communication step followed by an inner-table transformation step using MLP. During cross-table communication, we first compute meta-key embeddings through scatter-reduce and aggregation of table embeddings (table-to-key communication). We then update the table embeddings by combining previous table embeddings with corresponding meta-key embeddings selected based on the index (key-to-table communication). We use an example of a multi-table input where several constraints of standard relational databases are relaxed to demonstrate the flexibility of RDNN in modeling real-world relational data. We do not mandate the inclusion of a typical *primary key* that comprises all possible and unique values of the key in the database.

takes a relational DB, *i.e.*, multiple tables, as its input, and pass it through multiple RDNN layers to generate embeddings for each row in the tables, which we refer to as *table embeddings* $X_{T_t}^{(l)}$. By attaching proper prediction heads over the learned table embeddings, RDNNs can be trained end-to-end to perform classification or regression predictions over one or multiple columns.

4.1 Input: Relational DBs with Relaxed Constraints

A standard relational database (RDB) in conventional RDB management systems (RDBMS) must adhere to various constraints to enforce many-to-one relationships needed for identifying paths along which related tables are joined together. However, our observation reveals that many real-world relational datasets deviate from these constraints. In the example in Figure 6, there lacks a table where author ID (or paper ID) is presented as a typical *primary key* where the values need to be unique across all rows and comprise a super-set of all author IDs. Instead, there could be duplicated author ID in both tables (relaxed key constraints) and there is no one table that contains all the author IDs existing in the other tables (relaxed referential constraints). The author-paper edge table does not contain a unique A-P ID as the *primary key* (relaxed entity integrity constraints).

Rather than necessitating the intervention of experienced RDB experts to manually rectify these violated constraints, our proposed Relational Deep Neural Network (RDNN) can seamlessly accept RDBs with relaxed constraints as input. Notably, when the naming conventions for key columns are standardized, such as designating the primary key column as "table name id," RDNN can efficiently process multiple tabular datasets. This feature holds significant value for practitioners, as it eliminates the need for complex RDBMS systems to apply RDNN. Instead, the input can be as straightforward as multiple CSV files or Pandas data frames (McKinney et al., 2011). Essentially, we have expanded the input capabilities of RDNN from exclusively handling strict RDBs to encompass a wide range of multi-table databases. In the following discussions, we make the assumption that the input provided to RDNNs consists of relational databases with relaxed constraints.

4.2 Table encoder

We first apply preprocessing on the input tables $\{T_i\}$, which involves column type identification, missing value imputation, and data normalization. We obtain 3 types of feature columns: numerical, categorical, and other modalities. These features are then fed into a table encoder that transforms them into the initial table embeddings $X_{T_t}^{(0)} \in \mathbb{R}^d$ to be passed into RDNN layers. The numerical columns c_{num} are mapped to d dimensional embeddings through a linear layer. For categorical columns c_{cat}, we create a lookup table that stores trainable d-dim embeddings for each category. For columns with other complex modalities c_{other} such as text or molecule string representations, we could apply the corresponding pre-trained encoders *e.g.*, BERT(Devlin et al., 2018), to generate d-dimensional embeddings. As a final step, we sum up the embeddings for different columns to derive the initial table embeddings $X_{T_t}^{(0)}$.

$$X_{T_t}^{(0)} = \sum_{c_{\text{num}}} T_t[:, c_{\text{num}}] W_{c_{\text{num}}} + \sum_{c_{\text{cat}}} W_{c_{\text{cat}}} \left[T_t[:, c_{\text{cat}}], :\right] + \sum_{c_{\text{other}}} \phi(T_t[:, c_{\text{other}}]) \tag{1}$$

We exclude key columns from table features, as they usually contain meaningless identifiers and only serve as index. Note that designing the optimal feature engineering methods is not the focus of the paper. RDNN could easily work with other manual or automatic feature engineering methods. It is also worth mentioning that many public datasets have already preprocessed the column values for users, in which case the abovementioned table encoder can be skipped.

4.3 Cross-Table Communication

The forward pass of an RDNN layer takes as input the table embeddings from previous layer, and updates the embeddings through two steps: *cross-table communication* and *inner-table transformation*. To facilitate efficient information exchange across tables, we introduce the concept of *meta-key*.

Definition 1 *Meta-key K represents the set of key columns across all the tables in an RDB that refers to the same ID space. We define $M(T[:, c_{key}]) = K$ as the map from a key column c_{key} in table T to its corresponding meta-key K.*

In the example in Figure 6, two meta-keys, `author` and `paper`, can be derived and the `authorID` column from the two tables are mapped to `author` whereas the `paperID` column from the author-paper edge table is mapped to `paper`. We accomplish cross-table communication through a 2-stage procedure: *table-to-key communication* and *key-to-table communication*.

Table-to-key communication. In this stage, we derive embeddings for each index in the meta-keys, referred to as *meta-key embeddings* $X_{K_s}^{(l)}$, by aggregating the information from tables that contain the corresponding key column. We start with deriving the per-table key embeddings $X_{k_j|T_t}^{(l)}$ for each key column k_j of a given table T_t. To address duplicated key values, we use a scatter reduce function SCATTER[3] to map multiple rows in the table embeddings to a unique key embedding. Next, we aggregate per-table key embeddings into meta-key embeddings. The aggregator AGG could be standard operators such as MEAN or SUM. Note that there could be non-overlapped key values when merging key embeddings, *e.g.*, merging Author ID 1, 2 and Author ID 1, 3 to obtain $X_{\text{author}}^{(l)}$ in Figure 6. In this case, we zero-pad the missing key embeddings such that their shapes are identical. In summary, the table-to-key communication in the l-th RDNN layer can be summarized as

$$X_{K_s}^{(l)}[r, :] = \text{AGG}\Big\{\text{SCATTER}\big\{X_{T_t}^{(l)}[i, :] \big| \forall i, T_t[i, c_{\text{key}}] = r\big\} \Big| \forall T_t, \forall c_{\text{key}}, M(T_t[:, c_{\text{key}}]) = K_s\Big\} \tag{2}$$

where the output is the meta-key embeddings $X_{K_s}^{(l)}$, SCATTER and AGG has been discussed above, K_s refers to a meta-key column, c_{key} refers to a key column.

Key-to-table communication. In this stage, each row in the table embeddings $X_{T_t}^{(l)}[i, :]$ will be updated by aggregating information from all the meta-key embeddings $X_{K_s}^{(l)}$ with corresponding index (*i.e.*, INDEXSELECT) based on the meta-key column mapping M.

$$\widetilde{X}_{T_t}^{(l)}[i, :] = X_{T_t}^{(l)}[i, :] + \text{AGG}\Big\{X_{K_s}^{(l)}\left[T_t[i, k], :\right] \Big| \forall k, K_s = M(T_t[:, k])\Big\} \tag{3}$$

where the $[\cdot, \cdot]$ notation refers to INDEXSELECT, and a similiar AGG to Equation (2) is applied to aggregate from multiple meta-keys K_s to a given table T_t.

[3]Scatter reduce is natively supported in deep learning frameworks, *e.g.*, `torch.scatter_reduce`

4.4 Inner-Table Transformation

After the table embeddings are updated with information exchanged during cross-table communication, we apply a non-linear table transformation ϕ_{T_t} over the table embeddings before passing it to the next layer. We used a 2-layer multi-layer perceptron (MLP) with skip connection in the experiments and therefore final table embeddings in the output of the l-th RDNN layer can be computed with:

$$X_{T_t}^{(l+1)} = \sigma(\widetilde{X}_{T_t}^{(l)} W_{T_t} + B_{T_t}) + \widetilde{X}_{T_t}^{(l)} \tag{4}$$

where σ is a non-linear function such as ReLU, W_{T_t} and B_{T_t} are trainable weights. Note that rows in a table are treated as the batch dimension when applying table transformation; therefore, the trainable weights W_{T_t} and bias vector b_{T_t} (broadcasted into B_{T_t}) are shared across different rows of the table.

4.5 Supervised learning with RDNN

Here we describe the concrete steps for making column value predictions based on the generated RDNN embeddings. Given a target table T_t, we obtain the final table embeddings $X_{T_t}^{(L)}$ from the output of the last RDNN layer and apply a prediction head (e.g., linear or shallow MLP) on top of the embeddings to output predictions in the desired shape. To train RDNN end-to-end, we can define loss functions based on the designated supervised learning tasks, such as cross-entropy loss for classification and mean-squared-error for regression. For the column(s) that are selected as prediction targets, we exclude them from the table features and use them as labels for training the RDNN. We randomly split the rows into train, test, and validation sets following conventional machine learning practices. Note that the prediction tasks on RDBs are naturally self-supervised since the labels are defined as columns in RDB with predictive queries. RDNN can be used for multi-task learning and we defer discussions on the extension to multi-task learning setting to the Appendix.

5 Experiments

5.1 Experimental Setup

Graph datasets. We perform experiments over 7 different graph datasets. We consider 3 types of prediction tasks: node classification, graph classification, and heterogeneous node classification. The node classification datasets include (1) `Citeseer` (Sen et al., 2008), (2) `Pubmed` (Sen et al., 2008), and larger scale (3) `ogbn-arxiv` (Hu et al., 2020a) datasets, which are real-world citation networks. We consider graph classification datasets from TUDatasets (Morris et al., 2020), including (4) `ENZYMES`, (5) `PROTEINS`, and (6) `D&D`, which are real-world protein networks. We further consider node classification on a heterogeneous graph dataset, (7) `DBLP` (Gao et al., 2009).

RDB datasets. We further prepare 3 different RDB datasets to compare different methods. We consider the following datasets: (1) `Financial`, an RDB dataset provided in PKDD'99 Challenge, recording the information in a banking system including loans, transactions, accounts, etc. (Levin et al., 1999); (2) `Hepatitis`, and RDB dataset provided in PKDD'02 Challenge, which describes 206 instances of Hepatitis B patients (Khosravi et al., 2012); (3) `PTC`, an RDB dataset from Predictive Toxicology Challenge 2000, which includes organic molecules marked according to their carcinogenicity (Helma et al., 2001). For each RDB dataset, we prepare different predictive queries to represent different column prediction tasks within the RDB. We downloaded the RDB datasets from the website[4]. Visualization of the data schema and dataset statistics can be found in the Appendix.

Tasks. In graph datasets, we use the dataset splits provided by `ogbn-arxiv` dataset; for the remaining node and graph classification dataset, we use a random 80%/10%/10% train/validation/test split, where we report the test metrics under the best validation epoch; for the heterogeneous node classification dataset `DBLP`, following prior works Jin et al. (2021), we consider 3 different dataset splits, where we have 10%, 20%, and 40% data for training, respectively. Following the practice in prior works, we report ROC AUC for the `ogbn-arxiv` dataset, and Marco-F1 score for the `DBLP` dataset; we report classification accuracy for the remaining datasets.

[4]`https://relational.fit.cvut.cz/`

Table 1: **Comparing GNNs with RDNNs on real-world graph learning tasks**, with simple graphs (Left) and heterogeneous graphs (Right). Hyperparameters for all the models are controlled. Numbers are reported as ROC AUC (`ogbn-arxiv`), Marco-F1 (`DBLP`), and accuracy (remaining datasets).

	Node classification			Graph classification		
	`CiteSeer`	`Pubmed`	`ogbn-arxiv`	`ENZYMES`	`PROTEINS`	`D&D`
GCN	78.98	87.32	**71.24**	36.67	72.32	75.42
GraphSAGE	78.38	88.24	70.15	41.67	73.21	79.66
GAT	77.18	86.51	71.48	40.00	71.17	72.03
GIN	77.18	83.32	41.47	55.00	**74.10**	74.58
RDNN	**79.88**	**89.05**	71.03	**58.33**	73.21	**83.05**

	Heterogeneous node class.		
`DBLP`	10% train	20% train	40% train
GCN	43.70	44.75	45.26
GAT	53.61	54.81	55.09
HAN	**57.02**	57.61	57.75
MAGNN	56.39	58.11	59.39
RDNN	56.70	**60.48**	**64.15**

Models. For node classification and graph classification datasets, we use 4 widely adopted GNN models as the baseline models, including GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), GAT (Velickovic et al., 2018), where we use an improved GAT implementation based on (Brody et al., 2021)), and GIN (Xu et al., 2019). For node classification on heterogeneous graphs, we further include HAN (Wang et al., 2019) and MAGNN (Fu et al., 2020) as representative heterogeneous GNN baselines. For RDB prediction tasks, we consider tabular machine-learning baselines, including MLP, which follows all the designs of RDNN, except that there is no cross-table communication stage, and XGBoost (Chen & Guestrin, 2016); we further consider variants for MLP and XGBoost, where we additionally use SQL Left Join operation to merge all the available tables to the target table, mimicking how a real-world data scientist would approach the task with tabular learning methods.

Note that our goal is not to pursue state-of-the-art performance on any of the datasets; instead, we want to provide informative comparisons between RDNNs and existing methods; consequently, we do our best to control the hyperparameters during the experimentation. Specifically, all the GNNs, MLPs, and RDNNs have 3 layers and a hidden dimension of 128, unless a model experience notable overfitting, where we will switch them to 64 dimensions. We use ReLU activation and Layer Normalization (Ba et al., 2016) for all the models; when Layer Normalization provides unstable results, we consider using Batch Normalization (Ioffe & Szegedy, 2015) as an alternative approach to stabilize training. For all the models. We use Adam optimizer with a learning rate choosing between 0.01 and 0.001. For all the graph datasets, we converted them into tabular data following the illustration in Figure 1 and fed them into RDNN. Note that the reported performance of baselines may be different from the reported value from other sources, *e.g.*, OGB leaderboard, since we fix all the hyperparameters to ensure a fair comparison as discussed above. Restricted by computational resources, performing architecture searches and evaluating all the models under optimal hyperparameters are left for future research. We will release all the code and datasets at the time of publication to ensure reproducibility.

5.2 Prediction Results on Graph Datasets

First, we compare RDNNs against popular GNNs on diverse graph prediction datasets and tasks. As is shown in Table 1, RDNN could outperform all the baselines in 6 out of 9 settings that we have experimented with. In the cases where a GNN method performs the best, the performance gap between RDNN and the best method is often negligible. Notably, on `D&D` and `DBLP` with 40% training data, RDNN could significantly outperform the GNN baselines. It is worth emphasizing that when we apply GNN models to node/graph classification and heterogeneous graphs, we have to develop a completely different machine learning pipeline to handle the input data, prediction heads, and evaluation. In contrast, we use *the same RDNN implementation pipeline* for all the diverse graph learning tasks, *without any special customization for the task of interest*. We only use 2 different predictive queries, "node.label" and "graph.label" to inform RDNN which column to predict Given the discussions above, we found the fact that a general-purpose RDNN can often outperform specialized GNN models is quite encouraging.

5.3 Results on RDB datasets

Next, we compare RDNNs with tabular machine-learning methods on RDB datasets. We have created multiple predictive queries for each RDB dataset, where "loan.status" means we predict the column "status" in table "loan". As is shown in Table 2, RDNNs can significantly outperform all the baselines

Table 2: **Comparing GNNs with RDNNs on real-world RDB prediction tasks**. We control the hyperparameters for all the models. Numbers are reported as the accuracy (higher is better).

	`Financial`		`PTC`		`Hepatitis`		
Predictive query:	"loan.status"	"orders.symbol"	"molecule.label"	"atom.type"	"indis.alb"	"indis.got"	"indis.gpt"
MLP	80.88	69.35	N/A	N/A	86.47	69.95	82.95
MLP+Join	82.35	55.74	53.46	47.31	86.82	74.69	83.13
XGBoost	82.35	70.92	N/A	N/A	77.57	68.54	82.25
XGBoost+Join	81.25	50.00	55.88	45.87	74.72	73.81	82.65
RDNN	**94.12**	**77.80**	**80.28**	**91.57**	**88.05**	**78.38**	**85.59**

Table 3: **Ablation study: varying the number of RDNN layers** on `Financial` dataset with predictive query "loan.status". We report accuracy as the metric.

	Number of RDNN layers				
	0	1	2	3	4
RDNN's accuracy on `Financial` "loan.status"	80.88	81.25	89.85	**94.12**	93.52

in all the tasks we considered, even after we include manual data engineering processing for the MLP and XGBoost baselines. Notably, in the `Financial` dataset that consists of 8 tables, RDNN could effectively leverage the tables that are multi-hops away to make accurate predictions. We show an ablation study in table 3, where we vary the number of RDNN layers on the `Financial` dataset. Interestingly, we observed a significant performance jump when we started using 2-layer RDNNs; in the schema of `Financial` dataset, the table "transaction", which has over 1 million rows and therefore has rich information, is exactly 2-hop away from the target table "loan". Therefore, this observation demonstrates that RDNN could make effective information communication across tables, and consequently, holistically utilize the information across a given RDB.

6 Conclusions and Discussion

Limitations. Due to limitations in computational resources, most of our experiments are run with CPU nodes and we were not able to systematically assess RDNNs' scalability on larger-scale datasets. Furthermore, despite the abundant presence of relational DB in real-world applications, there is limited available benchmark dataset and open-source implementation of ML algorithms on RDBs, and therefore refrain from a more exhaustive comparison.

Future directions. In the Appendix, we describe a multi-task learning strategy to extend RDNN beyond single-column prediction. The idea is to perform masked self-supervised prediction. We envision that such a methodology can be further extended to pre-train RDNNs for more versatile and flexible predictive queries over RDB, which could pave the path for the exciting direction of building foundational models for relational databases and beyond. Additionally, since real-world relational databases could have billions of rows, studying memory-efficient algorithms for inference and training for RDNNs is quite valuable. Some preliminary results based on per-layer RDNN training have shown encouraging outcomes, and we encourage the community to further investigate the directions in the future.

Conclusion. In this paper, we demonstrate how leveraging relational databases (RDBs) can streamline the representation of relational data and unify diverse graph-based learning approaches. The transformative influence of the Transformer architecture on AI is evident in recent years; while not universally optimal, its versatility has cultivated a thriving ML ecosystem and remarkable achievements. Likewise, considering the widespread presence of RDBs in scientific and industrial realms, our modest aspiration is for our holistic perspective and RDB-focused approach to kindle a novel path in deep learning for this context. We aspire to catalyze an improved ML ecosystem tailored to relational data, mirroring the success prompted by Transformer's universality. Collaboratively engaging with the community, encompassing open-source initiatives, ML system optimization, benchmark establishment, and broadened applications, we envision endowing AI with the novel ability to learn from relational data — a key to unlocking an uncharted realm of knowledge.

REFERENCES

Sercan Ö Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 6679–6687, 2021.

Siddhant Arora, Vinayak Gupta, Garima Gaur, and Srikanta Bedathur. Bert meets relational db: Contextual representations of relational databases. *arXiv preprint arXiv:2104.14914*, 2021.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Jinze Bai, Jialin Wang, Zhao Li, Donghui Ding, Ji Zhang, and Jun Gao. Atj-net: Auto-table-join network for automatic learning on relational databases. In *Proceedings of the Web Conference 2021*, pp. 1540–1551, 2021.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

Landon Butler, Alejandro Parada-Mayorga, and Alejandro Ribeiro. Convolutional learning on multigraphs. *IEEE Transactions on Signal Processing*, 71:933–946, 2023.

Surajit Chaudhuri, Usama Fayyad, and Jeff Bernhardt. Scalable classification over sql databases. In *Proceedings 15th International Conference on Data Engineering (Cat. No. 99CB36337)*, pp. 470–479. IEEE, 1999.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

Milan Cvitkovic. Supervised learning on relational databases with graph neural networks. *arXiv preprint arXiv:2002.02046*, 2020.

Justas Dauparas, Ivan Anishchenko, Nathaniel Bennett, Hua Bai, Robert J Ragotte, Lukas F Milles, Basile IM Wicky, Alexis Courbet, Rob J de Haas, Neville Bethel, et al. Robust deep learning–based protein sequence design using proteinmpnn. *Science*, 378(6615):49–56, 2022.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 3558–3565, 2019.

Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. In *Proceedings of The Web Conference 2020*, pp. 2331–2341, 2020.

Jing Gao, Feng Liang, Wei Fan, Yizhou Sun, and Jiawei Han. Graph-based consensus maximization among multiple supervised and unsupervised models. *Advances in neural information processing systems*, 22, 2009.

Paul WPJ Grefen and Peter MG Apers. Integrity control in relational database systems—an overview. *Data & Knowledge Engineering*, 10(2):187–223, 1993.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Christoph Helma, R. D. King, S. Kramer, and A. Srinivasan. The Predictive Toxicology Challenge 2000-2001. *Bioinformatics*, 17(1):107–108, January 2001. ISSN 1367-4803. doi: 10.1093/bioinformatics/17.1.107. URL `http://bioinformatics.oxfordjournals.org/cgi/doi/10.1093/bioinformatics/17.1.107`.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020a.

Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of the web conference 2020*, pp. 2704–2710, 2020b.

Jing Huang and Jie Yang. Unignn: a unified framework for graph and hypergraph neural networks. *arXiv preprint arXiv:2105.00956*, 2021.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 448–456, Lille, France, 07–09 Jul 2015. PMLR. URL `https://proceedings.mlr.press/v37/ioffe15.html`.

Di Jin, Cuiying Huo, Chundong Liang, and Liang Yang. Heterogeneous graph neural network via attribute completion. In *Proceedings of the web conference 2021*, pp. 391–400, 2021.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. *International Conference on Machine Learning (ICML)*, 2018.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

Hassan Khosravi, Oliver Schulte, Jianfeng Hu, and Tianxiang Gao. Learning compact Markov logic networks with decision trees. *Machine Learning*, 89(3):257–277, 2012. ISSN 08856125. doi: 10.1007/s10994-012-5307-6.

Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. Learned cardinalities: Estimating correlated joins with deep learning. *arXiv preprint arXiv:1809.00677*, 2018.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations (ICLR)*, 2017.

Hai Lan, Zhifeng Bao, and Yuwei Peng. An index advisor using deep reinforcement learning. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 2105–2108, 2020.

Boris Levin, Abraham Meidan, Alex Cheskis, Ohad Gefen, and Ilya Vorobyov. Pkdd'99 discovery challenge—financial domain. In *Workshop Notes on Discovery Challenge. Workshop at 3rd European Conference on Principles and Practice of Knowledge Discovery and Data Mining (PKDD'99)*, 1999.

Ryan Marcus and Olga Papaemmanouil. Deep reinforcement learning for join order enumeration. In *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, pp. 1–4, 2018.

Wes McKinney et al. pandas: a foundational python library for data analysis and statistics. *Python for high performance and scientific computing*, 14(9):1–9, 2011.

Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020.

Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 5363–5370, 2020.

Yongjoo Park, Shucheng Zhong, and Barzan Mozafari. Quicksel: Quick selectivity learning with mixture models. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 1017–1033, 2020.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15*, pp. 593–607. Springer, 2018.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Jian Tan, Tieying Zhang, Feifei Li, Jie Chen, Qixing Zheng, Ping Zhang, Honglin Qiao, Yue Shi, Wei Cao, and Rui Zhang. ibtune: Individualized buffer tuning for large-scale cloud databases. *Proceedings of the VLDB Endowment*, 12(10):1221–1234, 2019.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *International Conference on Learning Representations (ICLR)*, 2018.

Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The world wide web conference*, pp. 2022–2032, 2019.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *International Conference on Learning Representations (ICLR)*, 2019.

Jiaxuan You, Tianyu Du, and Jure Leskovec. Roland: graph learning framework for dynamic graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2358–2366, 2022.

Haitao Yuan, Guoliang Li, Ling Feng, Ji Sun, and Yue Han. Automatic view generation with deep learning and reinforcement learning. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pp. 1501–1512. IEEE, 2020.

Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):i190–i198, 2017.

A Discussions: Comparing with GNNs and Table Join + MLP

Benefits and limitations of RDNNs comparing with GNNs. Compared to GNNs, a majority difference of RDNN is that node and edge are treated equally (they are regarded as two different tables), whereas in GNN designs, computation defined on node and edge are different. This leads to several benefits of RDNNs. (1) User no longer need to worry about properly defining node, edge, and hyperedge from their relational data, everything is rows in tables. (2) When defining the model, the same set of computations is applied across all the tables, without the need to define specialized computations for different tables. (3) RDNN is simpler for machine learning researcher to explore new designs; furthermore, the homogeneous computation behind RDNN provide additional room for system researchers to optimize the pipeline. Indeed, if simple graphs are of concern, then RDNN does not seem to have an obvious advantage and incur additional computation on the edge case, which may be redundant; we would recommend researchers continually use GNNs over RDNNs if the input is simple graphs. However, when the relational data are complex, then the generality of RDB representation and simplicity of defining RDNN has a clear advantage over the alternatively GNN approach, where a user would have to first organize the data into heterogeneous graphs, then define Heterogenous GNN.

Connectsions with MLP+Join. The concept of RDNN is also related to the approach of table join + MLP. The index selection phase is conceptually related to many-to-one join, while the scatter reduce phase is conceptually related to one-to-many join. The one-to-many join operation is particularly challenging as it entails either losing information by selecting a single row for joining or facing scalability issues. This is due to the exponential expansion of target rows, necessitating a late aggregation over all the expanded rows, which can be a complex task. Instead, RDNN can be understood as a neural version of table join + MLP that can be optimized end-to-end.

B Multi-task Self-supervised Learning with RDNN

So far, we have assumed that RDNNs are used to predict a single column within a database. In practice, we could define multiple-column prediction tasks for a database. The common practice of separating features and labels in ML task formulation is not ideal: excluding known labels inevitably results in the loss of useful information for the prediction task. In the extreme case of a missing value imputation task where all columns have missing values, all the columns should be regarded as labels, and there are no features.

Instead, we propose an effective approach to handling multiple prediction tasks in an RDB. First, we record the masks for the missing values and apply naive missing value imputation methods to fill the unknown values. For numerical columns, we impute with the mean values of a given column, and for categorical columns, we impute by creating an additional 'UNK' category. Then, we feed the full RDB with completed feature values to RDNN, generate the embeddings for rows with masked missing values, and attach a different prediction head for each column to be predicted. We will synthetically create different training, validation, and optionally test masking to consistently train RDNN under the same setting. With this simple yet effective approach, a single RDNN can be trained for multiple prediction tasks in an RDB. Due to the limited computational resources, we defer the exploration of multi-task self-supervised learning for the future work.

C Statistics of the RDB Datasets in experiments

Table 4: Statistics of the RDB datasets used in the experiment

Dataset	Domain	# Tables	Diameter	# Cols	# Rows
`Financial`	Economic	8	3	55	1,079,680
`Hepatitis`	Society	7	4	26	12,927
`PTC`	Economic	4	2	11	49,239

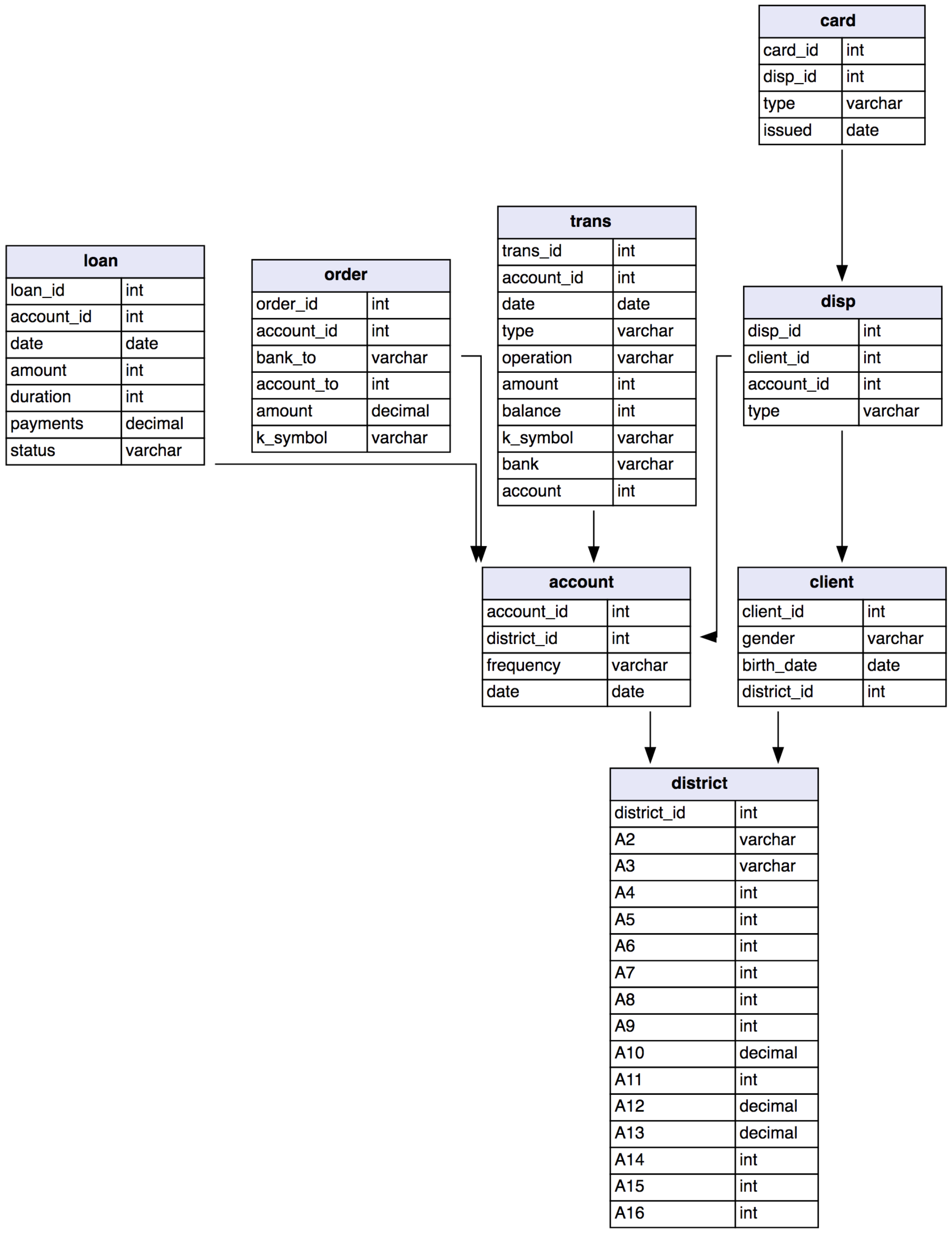


Figure 4: `Financial` RDB dataset

D VISUALIZATIONS OF THE RDB DATASETS IN EXPERIMENTS

Here we include the RDB schemas for the datasets investigated in this paper, obtained from `https://relational.fit.cvut.cz/`.

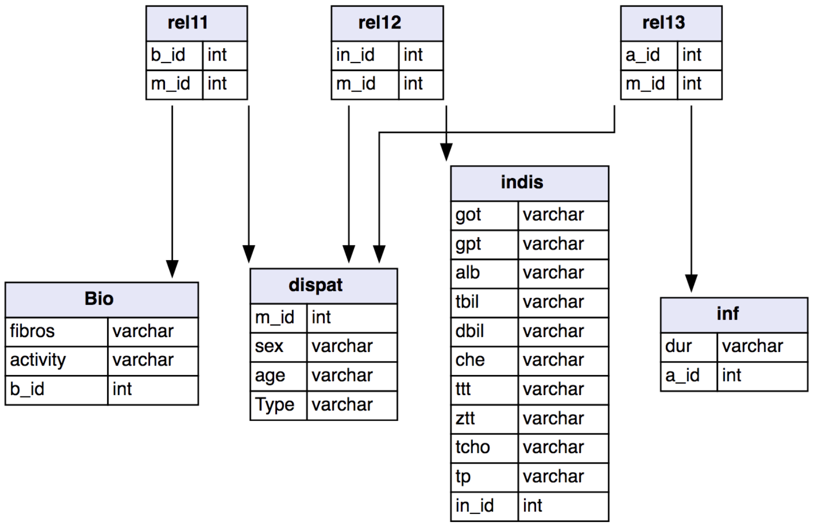


Figure 5: `Hepatitis` RDB dataset

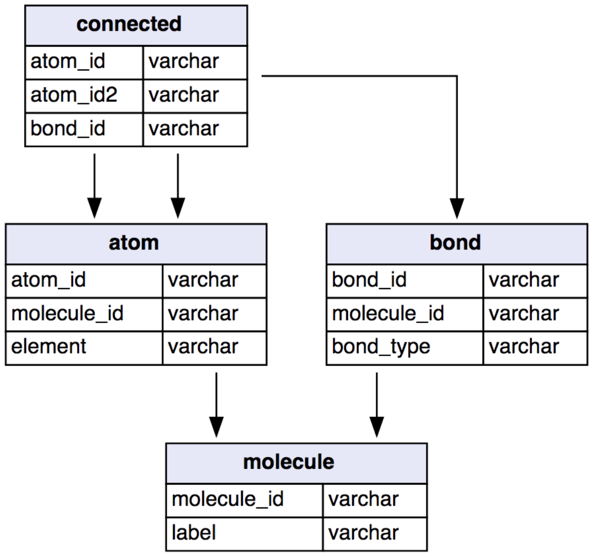


Figure 6: `PTC` RDB dataset