TypeDB

TypeDB Philosophy

Programming languages have evolved rapidly, but database models have failed to keep up - and modern applications must use complex layered architectures to manage data as a result. In order to resolve this, we built TypeDB on a completely new, highly expressive database paradigm. This page introduces its origins and core ideas.

Visit learning center

Review features

Why do we need a new kind of database?

"The limits of my language mean the limits of my world" — **Ludwig Wittgenstein, philosopher and logician**

Visit learning center

Read article

Application code uses complex data structures

TypeDB was conceived to solve the lack of expressivity in current database paradigms. Programming languages are becoming increasingly more declarative, empowering engineers to quickly write safe and expressive code backed by static type checking and abstract data constructs. Object-oriented programming involves complex modeling constructs such as abstraction, inheritance, and polymorphism, requiring the expression of multidimensional data structures. However, current databases are unable to natively handle these data structures that we so easily take for granted.

```
1
2  // Class inheritance in Java
3
4  class User {
5      String email;
6      String name;
7
8      User(String email, String name) {
9          this.name = name;
10         this.email = email;
11     }
```

Relational databases lack the same expressivity

Relational databases were designed at a time when procedural programming languages were the norm. They are built on Codd's relational algebra and implemented using tables and one-dimensional tuples. As a result, relational databases are unable to natively model complex data structures due to their lack of expressivity. This fundamental incompatibility between object and relational models has become one of the biggest challenges in database engineering and left databases unable to evolve alongside programming languages.

```
1
2  -- Table inheritance in SQL
3
4  CREATE TABLE users (
5      id SERIAL PRIMARY KEY,
6      email TEXT NOT NULL UNIQUE,
7      name TEXT NOT NULL,
8  );
9
10 CREATE TABLE employees (
11     id INTEGER PRIMARY KEY REFERENCES users(id),
12     employee_id INTEGER NOT NULL UNIQUE
```

Schemaless databases come at a high cost

The limitations of relational databases led to the emergence of NoSQL databases, particularly document and graph databases. These databases eliminated the predefined schema, making data insertion trivial, but this comes with the cost of complicating retrieval. Without a schema, structural metadata must be stored as data, hardcoded into queries, or modeled in a secondary data store. This forces engineers to access their data imperatively, as the database does not have the context to correctly interpret declarative polymorphic queries.

Document | Graph

```
// Retrieval of polymorphic data in MongoDB

db.resource_ownerships.aggregate([
    {
        $lookup:
        {
            from: "resources",
            localField: "resource",
            foreignField: "_id",
```

What makes TypeDB unique among databases?

TypeDB has a strongly-typed schema for defining inheritance hierarchies and interfaces, a variablizable query language for composing truly declarative queries, and a type inference engine for resolving queries against the schema.

Visit learning center

Read article

Types are defined in a hierarchy

TypeDB models are described by types, defined in a schema as templates for data instances, analogous to classes. Each user-defined type extends one of three root types: `entity`, `relation`, and `attribute`, or a previously user-defined type. Entities represent independent concepts. Relations represent concepts dependent on roles played by entities and other relations. Attributes represent properties of entities and relations. Any type can be made concrete or abstract, and roles in relations can be overridden by their subtypes.

```
define

  user sub entity;
  admin sub user;
  user-group sub entity;
```

```
7     resource sub entity, abstract;
8     file sub resource;
9
10    ownership sub relation, abstract,
11      relates owned,
12      relates owner;
```

Behaviors are shared and inherited

Once the entity, relation, and attribute type hierarchies have been defined, the attributes that entities and relations `own` and the roles they `play` in relations are implemented like interfaces. Declarations of owned attributes and played roles are independent of each other, and multiple entity and relation types can own the same attribute or play the same role. The attributes a type owns and the roles it plays are inherited by its subtypes, and can be overridden to specialize them.

```
1
2   define
3     user owns email,
4       plays resource-ownership:owner;
5     admin plays group-ownership:owner;
6     user-group owns name,
7       plays group-ownership:group,
8       plays resouce-ownership:owner;
9     resource owns id,
10      plays resource-ownership:resource;
11    file owns path as id;
12
```

Data is semantically validated

With the type hierarchies and interfaces defined in the schema, data can be instantiated with an `insert` query. Data instances and types are defined by variables using a `$variable-name` that exists in the scope of the query and can be reused to describe complex data patterns. Relations are defined with a tuple of roleplayers and the roles they play. Write queries undergo semantic validation against the schema, ensuring that the inserted or modified data patterns are valid. Queries that would insert data not allowed by the schema are rejected.

```
1
2   insert
3
4     $naomi isa admin, has email "naomi@typedb.com";
5     $amos isa user, has email "amos@typedb.com";
6     $engineers isa user-group, has name "engineers";
```

```
7     $benchmark isa file, has path "/amos/benchmark-results.xlsx";
8     $roadmap isa file, has path "/typedb/feature-roadmap.pdf";
9
10    (group: $engineers, owner: $naomi) isa group-ownership;
11    (resource: $benchmark, owner: $amos) isa resource-ownership;
12    (resource: $roadmap, owner: $engineers) isa resource-ownership;
```

Data is queried declaratively

Data is queried with high-level patterns, in which any element can be variablized. Queries are analyzed by the type inference engine before going to the query planner. It resolves polymorphism by identifying possible types that could fit patterns as defined by the schema, and queries return instances of all those types. Querying a supertype returns instances of subtypes that inherit from it. Querying an interface returns instances of types that implement it. Querying a variablized type parametrically returns instances of all types that match the pattern.

```
1
2   match
3     (owned: $object, owner: $owner) isa! $ownership-type;
4   fetch
5     $ownership-type;
6     $object: id;
7     $owner: id;
8
9   # Results:
10
11  [{
12      "ownership-type": { "root": "relation", "label": "group-ownership" },
```

How does TypeDB impact database engineering?

TypeDB natively implements high-level model features like abstraction, inheritance, and polymorphism. This enables engineers to work with flexible and adaptable data models, making it easier to manage, query, and reason over complex data structures.

Visit learning center

Review features

A unified way of working with data

TypeDB enables engineers to use the same conceptual data models in their application and database, by directly implementing high-level abstractions. Unlike with other databases, additional backend layers are not necessary, and all of these features are built-in, robust, and performant. This empowers engineers to quickly write queries that are as safe and expressive as code. TypeDB redefines database architecture by providing the tools required for modern application development, leading to a number of unique benefits.

Database | Application

```
# Type inheritance in TypeQL

define

  user sub entity;
  email sub attribute, value string;
  name sub attribute, value string;
  user owns name, owns email;

```

Data integrity

The conceptual schema and advanced constraint language provide integrity guarantees even when working with polymorphic data structures. Unlike with other databases, where integrity is managed in the application, TypeDB validates data directly at the source.

Read article >

Continuous extensibility

Queries are truly declarative, so the results of queries automatically extend to include new valid types that are added to the schema after the query is written. This minimizes the need to maintain and update queries when the schema is extended.

Read article >

Consolidated models

The high-level data model means that schemas in relational, document, and graph databases can be translated to TypeDB with no loss of information. This allows for easy transfer of data from other databases, during migrations or when building an analytics layer.

Read article >

Integrated application logic

Integrate application logic by defining functions in the schema that can be called directly from queries. Functions use the same syntax as queries, allowing functions to provide high-level abstractions for complex constraints, contained within their logic.

Read article >

Object model parity

The conceptual schema allows for perfect parity with object models. Data structures that are challenging to model in other databases are simple and intuitive in TypeDB, with type hierarchies, abstract types, multivalued attributes, n-ary relations, and more.

Read article >

Near-natural language

Schemas and queries read close to natural language, without having to use imperative language to describe low-level data structures. As a result, Engineers and domain experts can understand the intent of queries, even with no experience of writing them.

Read article >

Get started today

Deploy locally or in the cloud and start building now, or explore more about TypeDB and how its unique capabilities as a polymorphic database can refine and empower your applications.

Start building with TypeDB

Cloud or container, a polymorphic database with a conceptual data model, a strong subtyping system, a symbolic reasoning engine, and a type-theoretic language is minutes away.

Deploy

Learn more

Review features >

Visit learning center >

Subscribe to newsletter >

Join Discord community >

Subscribe to Newsletter

Connect

Discuss on Forum

Chat on Discord

Contact Us

Provide Feedback

Technology

Philosophy

Documentation

Overview

Resources

Lectures

Company

LinkedIn