Crate lancedb

LanceDB is an open-source database for vector-search built with persistent storage, which greatly simplifies retrieval, filtering and management of embeddings.

The key features of LanceDB include:

- Production-scale vector search with no servers to manage.
- Store, query and filter vectors, metadata and multi-modal data (text, images, videos, point clouds, and more).
- Support for vector similarity search, full-text search and SQL.
- Native Rust, Python, Javascript/Typescript support.
- Zero-copy, automatic versioning, manage versions of your data without needing extra infrastructure.
- GPU support in building vector indices[1].
- Ecosystem integrations with LangChain 🦜🔗, LlamaIndex 🦙, Apache-Arrow, Pandas, Polars, DuckDB and more on the way.

Getting Started

LanceDB runs in process, to use it in your Rust project, put the following in your `Cargo.toml`:

```
cargo add lancedb
```

Crate Features

Experimental Features

These features are not enabled by default. They are experimental or in-development features that are not yet ready to be released.

- `remote` - Enable remote client to connect to LanceDB cloud. This is not yet fully implemented and should not be enabled.

Quick Start

Connect to a database.

```
let db = lancedb::connect("data/sample-lancedb").execute().await.unwrap();
```

LanceDB accepts the different form of database path:

- `/path/to/database` - local database on file system.
- `s3://bucket/path/to/database` or `gs://bucket/path/to/database` - database on cloud object store
- `db://dbname` - Lance Cloud

You can also use [`ConnectOptions`] to configure the connection to the database.

```
use object_store::aws::AwsCredential;
let db = lancedb::connect("data/sample-lancedb")
    .aws_creds(AwsCredential {
        key_id: "some_key".to_string(),
        secret_key: "some_secret".to_string(),
        token: None,
    })
    .execute()
    .await
    .unwrap();
```

LanceDB uses arrow-rs to define schema, data types and array itself. It treats `FixedSizeList<Float16/Float32>` columns as vector columns.

For more details, please refer to the LanceDB documentation.

Create a table

To create a Table, you need to provide a `arrow_schema::Schema` and a `arrow_array::RecordBatch` stream.

```
use arrow_array::{RecordBatch, RecordBatchIterator};
use arrow_schema::{DataType, Field, Schema};

let schema = Arc::new(Schema::new(vec![
    Field::new("id", DataType::Int32, false),
    Field::new(
```

```
            "vector",
            DataType::FixedSizeList(Arc::new(Field::new("item", DataType::Floa
            true,
        ),
    ]));
    // Create a RecordBatch stream.
    let batches = RecordBatchIterator::new(
        vec![RecordBatch::try_new(
            schema.clone(),
            vec![
                Arc::new(Int32Array::from_iter_values(0..256)),
                Arc::new(
                    FixedSizeListArray::from_iter_primitive::<Float32Type, _,
                        (0..256).map(|_| Some(vec![Some(1.0); 128])),
                        128,
                    ),
                ),
            ],
        )
        .unwrap()]
        .into_iter()
        .map(Ok),
        schema.clone(),
    );
    db.create_table("my_table", Box::new(batches))
        .execute()
        .await
        .unwrap();
```

Create vector index (IVF_PQ)

LanceDB is capable to automatically create appropriate indices based on the data types of the columns. For example,

- If a column has a data type of `FixedSizeList<Float16/Float32>`, LanceDB will create a `IVF-PQ` vector index with default parameters.

```rust
use lancedb::index::Index;
tbl.create_index(&["vector"], Index::Auto)
    .execute()
    .await
    .unwrap();
```

User can also specify the index type explicitly, see `Table::create_index`.

Open table and search

```rust
let results = table
    .query()
    .nearest_to(&[1.0; 128])
    .unwrap()
    .execute()
    .await
    .unwrap()
    .try_collect::<Vec<_>>()
    .await
    .unwrap();
```

---

1. Only in Python SDK. ↩

Re-exports

```rust
pub use connection::ConnectNamespaceBuilder;
pub use connection::Connection;
pub use error::Error;
pub use error::Result;
pub use table::Table;
pub use connection::connect;
pub use connection::connect_namespace;
```

Modules

arrow	
connection	Functions to establish a connection to a LanceDB database
data	Data types, schema coercion, and data cleaning and etc.

database	The database module defines the `Database` trait and related types.
dataloader	
embeddings	
error	
index	
io	
ipc	IPC support
query	
rerankers	
table	LanceDB Table APIs
utils	

Structs

ObjectStoreRegistry	A registry of object store providers.
Session	Re-export Lance Session and ObjectStoreRegistry for custom session creation A user session holds the runtime state for a `crate::Dataset`

Enums

DistanceType