SurrealDB 2.1 Overview

write a comprehensive wikipedia article on surreal db 2.1 including specific examples in Rust sdk. span all topics from basic to advanced

SurrealDB 2.1 Wikipedia-Style Comprehensive Article

Introduction

SurrealDB 2.1 is a multi-model, distributed database management system designed to provide a unified platform for handling relational, document, and graph-like data models. Engineered for modern applications, SurrealDB supports schema flexibility, advanced querying capabilities, and a powerful Rust SDK. Version 2.1 introduces several enhancements in concurrency, live query handling, and integrations with Rust-based systems, making it a favored choice for real-time applications and AI-driven workloads.

Core Features

- **Multi-Model Database**: Supports SQL-like querying over relational, document-based, and graph data models.
- **Concurrency**: Optimized for high-performance with asynchronous Rust SDK and Tokio runtimecombined.
- **Embedded and Remote Operation**: Can operate as an in-memory database, file-based storage, or in distributed clusters combined.

- **Advanced Querying**: Full support for SurrealQL with capabilities for live queries, manual transactions, and flexible joinscombinedcombined.
- **Security Features**: Role-based access control, JWT authentication, and advanced access definitionscombined combined.
- **Rust SDK**: Comprehensive support for interacting with SurrealDB in Rust, including live notifications, advanced transactions, and user-friendly API integrationcombined combined.

Setting Up SurrealDB

1. Installation

SurrealDB can be installed as a standalone binary or embedded in Rust applications. For standalone use, download the executable from the official site.

2. Starting the Server

Run the following command to start a SurrealDB server with default configurations:

```bash
surreal start --user root --pass root
```

The server runs at `` `http://127.0.0.1:8000` `` by default.

Key Concepts

Authentication and Security

- **JWT Authentication**: Securely authenticate users using JWTs with fine-grained access control policiescombined.
- **Dynamic Access Control**: Define permissions using SurrealQL's `` `DEFINE ACCESS` `` statements, ensuring contextual security combinedcombined.

Concurrency

SurrealDB employs Rust's `` `tokio` `` runtime for high-concurrency scenarios. Clients can customize connection capacities to optimize memory usagecombined.

Live Queries

Monitor database changes in real-time using `` `LIVE SELECT` `` statements. This is particularly useful for applications requiring instantaneous data synchronizationcombinedcombined.

Using the Rust SDK

The Rust SDK enables seamless integration with SurrealDB for building performant applications. Below are examples from basic to advanced usage:

---

1. Basic Connection

```
rust


  use surrealdb::Surreal;
  use surrealdb::engine::remote::ws::Ws;
```

```
#[tokio::main]
async fn main() -> surrealdb::Result<()> {
    let db = Surreal::new::<Ws>("127.0.0.1:8000").await?;
    db.signin("root", "root").await?;
    db.use_ns("test").use_db("test").await?;
    Ok(())
}
```

---

2. CRUD Operations

Creating Records

```rust
use surrealdb::Surreal;

#[derive(Debug, Serialize)]
struct Person {
    name: String,
    age: u32,
}

#[tokio::main]
async fn main() -> surrealdb::Result<()> {
    let db = Surreal::new::<Ws>("127.0.0.1:8000").await?;
    let person = Person { name: "John Doe".to_string(), age: 30 };
    let record: Option<RecordId> = db.create("person").content(person).await?;
    dbg!(record);
    Ok(())
}
```

Fetching Records

```rust
let people: Vec<Person> = db.select("person").await?;
```

---

3. Live Queries

```rust
use futures::StreamExt;
use surrealdb::engine::remote::ws::Ws;
use surrealdb::opt::auth::Root;
use surrealdb::{Notification, Surreal};

#[derive(Debug, Deserialize)]
struct Person {
    id: String,
    name: String,
}

#[tokio::main]
async fn main() -> surrealdb::Result<()> {
    let db = Surreal::new::<Ws>("127.0.0.1:8000").await?;
    db.signin(Root { username: "root", password: "root"
}).await?;
    db.use_ns("test").use_db("test").await?;

    let mut live_stream = db.select("person").live().awai
t?;
    while let Some(result) = live_stream.next().await {
        println!("{:?}", result);
    }
    Ok(())
}
```

---

4. Manual Transactions

```rust
  let response = db
      .query("BEGIN")
      .query("CREATE account:one SET balance = 1000;")
      .query("UPDATE account:one SET balance += 500;")
      .query("COMMIT")
      .await?;
```

---

5. Advanced Features

Using `FETCH` for Linked Records

```rust
  let results = db.query("SELECT * FROM classroom FETCH teacher, students").await?;
```

Performance Optimization

- **Concurrency Management**: Optimize `tokio` runtime configurations and database connection pools.
- **Custom Live Query Notifications**: Streamline live query notifications to handle real-time updates efficientlycombined combined.

Conclusion

SurrealDB 2.1 integrates seamlessly with Rust and offers extensive capabilities for managing modern application data. Its multi-model architecture, coupled with robust Rust SDK support, makes it a prime choice for real-time, scalable, and secure systems. For further resources and community support, visit SurrealDB's official site.