Record Layer 101

Pierre Zemb
FoundationDB Paris Meetup
15, October 2020

$ whoami

- Pierre Zemb (@PierreZ)
- OVHcloud™ Technical leader
- Working around distributed systems
    - HBase/Flink/Kafka/Pulsar/ETCD
- Involved into local communities

Finist Devs DEVOXX 4KIDS DevFest du Bout du Monde

Agenda

- Intro to layers
- What is the Record Layer?
- From the ETCD-layer
    - Bootstrapping the Record Layer
    - Put a record
    - Query a record
    - Handle leases and watches
- From the Record-Store:
    - multi-tenancy
    - Storing metadata
    - schema updates
    - Encryption
    - long records
    - query capabilities
- Things I want to try

My story with FDB

- Discovered FDB two years ago 🚀
- Read/watched a lot of stuff 📖
- I was looking to build something 👨‍💻
- Then french lockdown came 😷
- Started two layers 🤩
    - ETCD
    - Record-Store

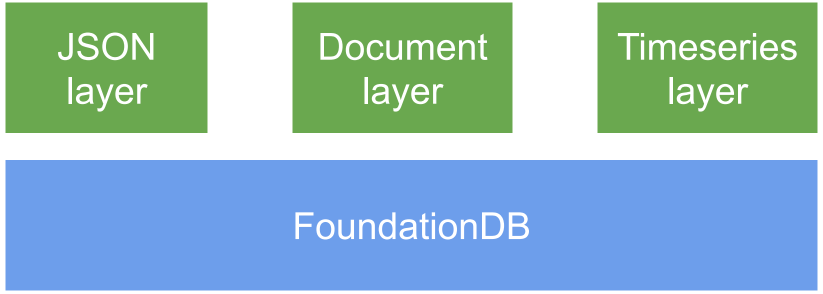
Layers 🤔

- FDB offers amazing foundations 🤡
  - ordered key-value store with multi-key transactions
  - scalable, fault-tolerant, production-ready

Layers 🤔

- FDB offers amazing foundations 🤡
  - ordered key-value store with multi-key transactions
  - scalable, fault-tolerant, production-ready

- But maybe you need
  - Document databases,
  - column-oriented,
  - row-oriented,
  - JSON,
  - key-value,

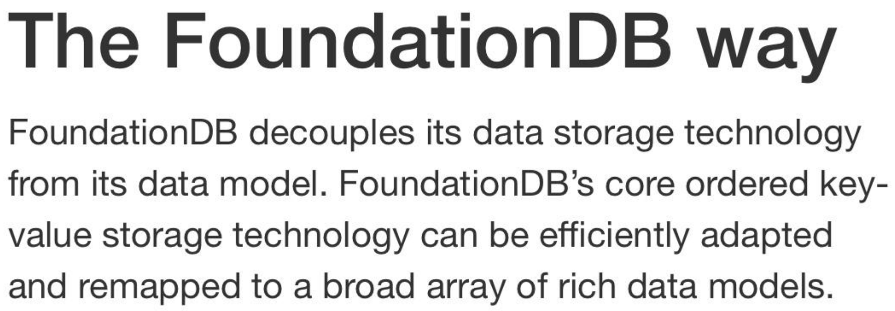


Layers 🤔

Famous layers:

- F1 over Spanner
- tidb over tikv
- Phoenix over HBase
- OpenTSDB over HBase
- CouchDB 4 over FDB soon

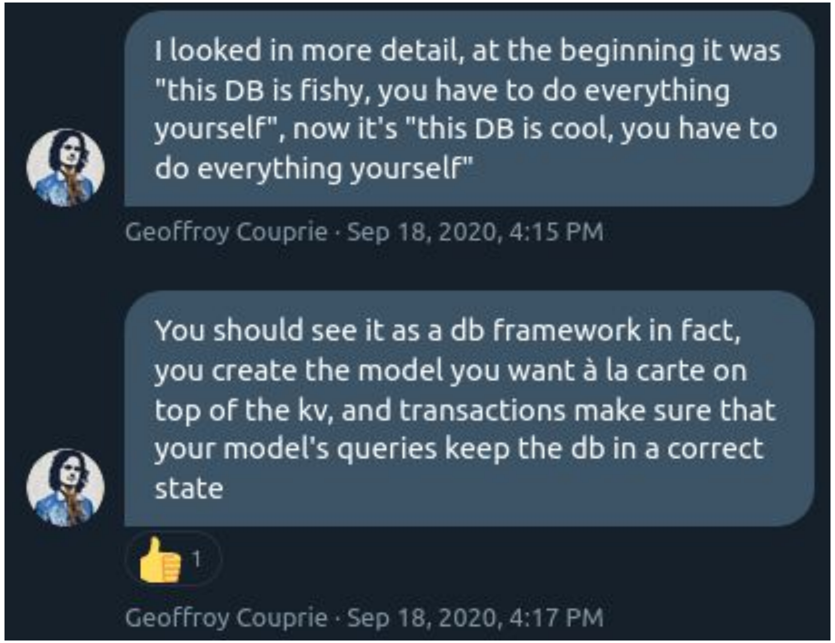
Layers 🤩



The FoundationDB way

FoundationDB decouples its data storage technology from its data model. FoundationDB's core ordered key-value storage technology can be efficiently adapted and remapped to a broad array of rich data models.

Layers 🤩



I looked in more detail, at the beginning it was "this DB is fishy, you have to do everything yourself", now it's "this DB is cool, you have to do everything yourself"

Geoffroy Couprie · Sep 18, 2020, 4:15 PM

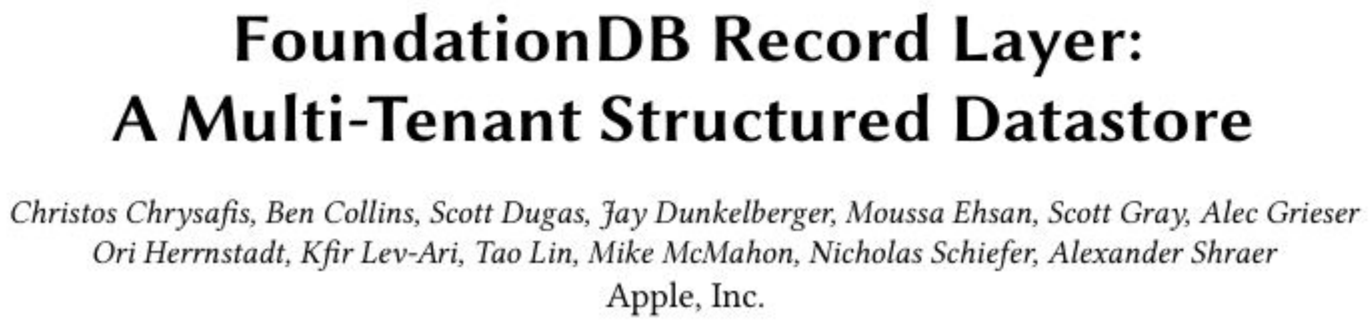
You should see it as a db framework in fact, you create the model you want à la carte on top of the kv, and transactions make sure that your model's queries keep the db in a correct state

👍 1

Geoffroy Couprie · Sep 18, 2020, 4:17 PM

Let's build a layer 👨‍💻

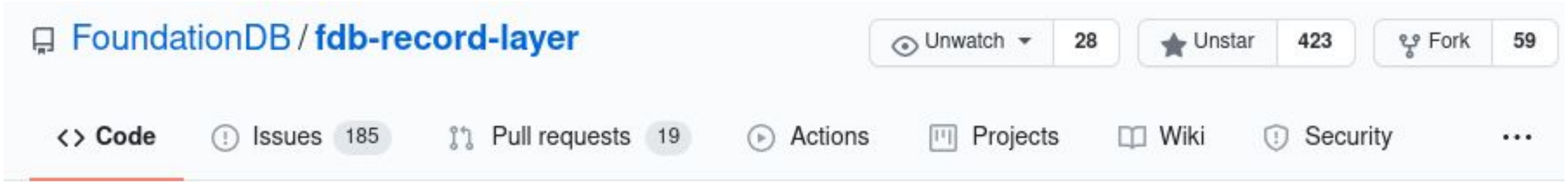
How do I build a layer 🤔?

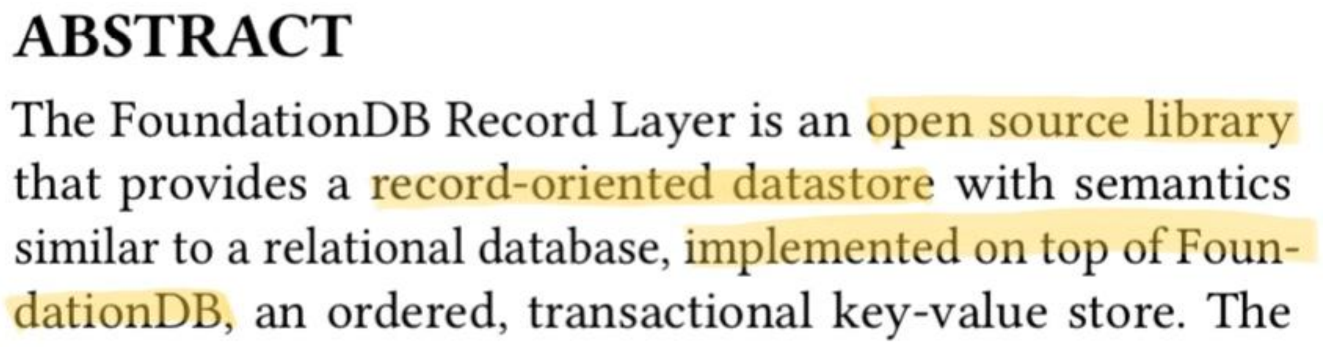


FoundationDB Record Layer: A Multi-Tenant Structured Datastore

Christos Chrysafis, Ben Collins, Scott Dugas, Jay Dunkelberger, Moussa Ehsan, Scott Gray, Alec Grieser Ori Herrnstadt, Kfir Lev-Ari, Tao Lin, Mike McMahon, Nicholas Schiefer, Alexander Shraer
Apple, Inc.

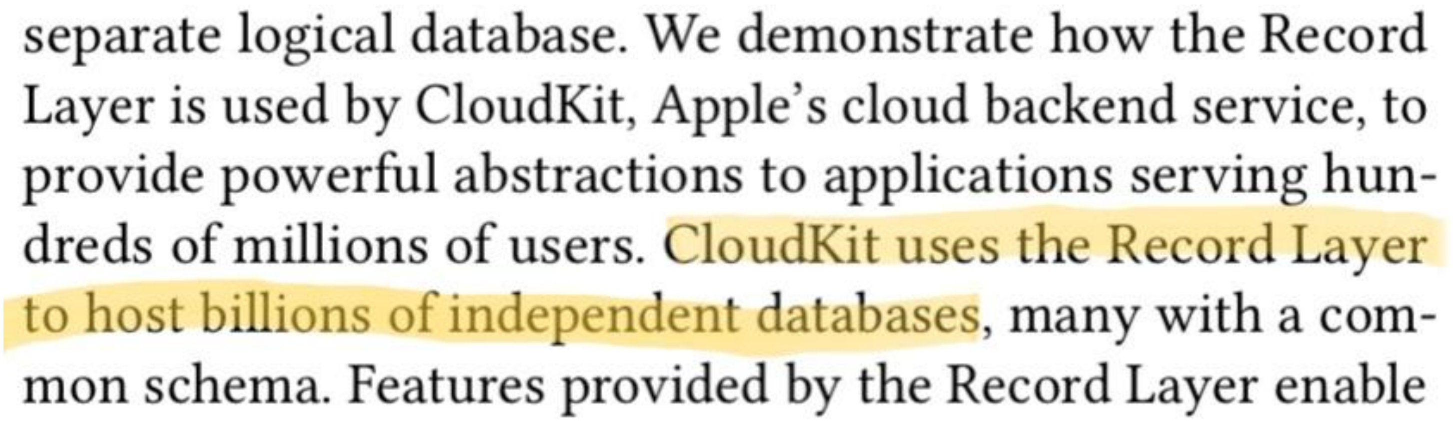
https://www.foundationdb.org/files/record-layer-paper.pdf



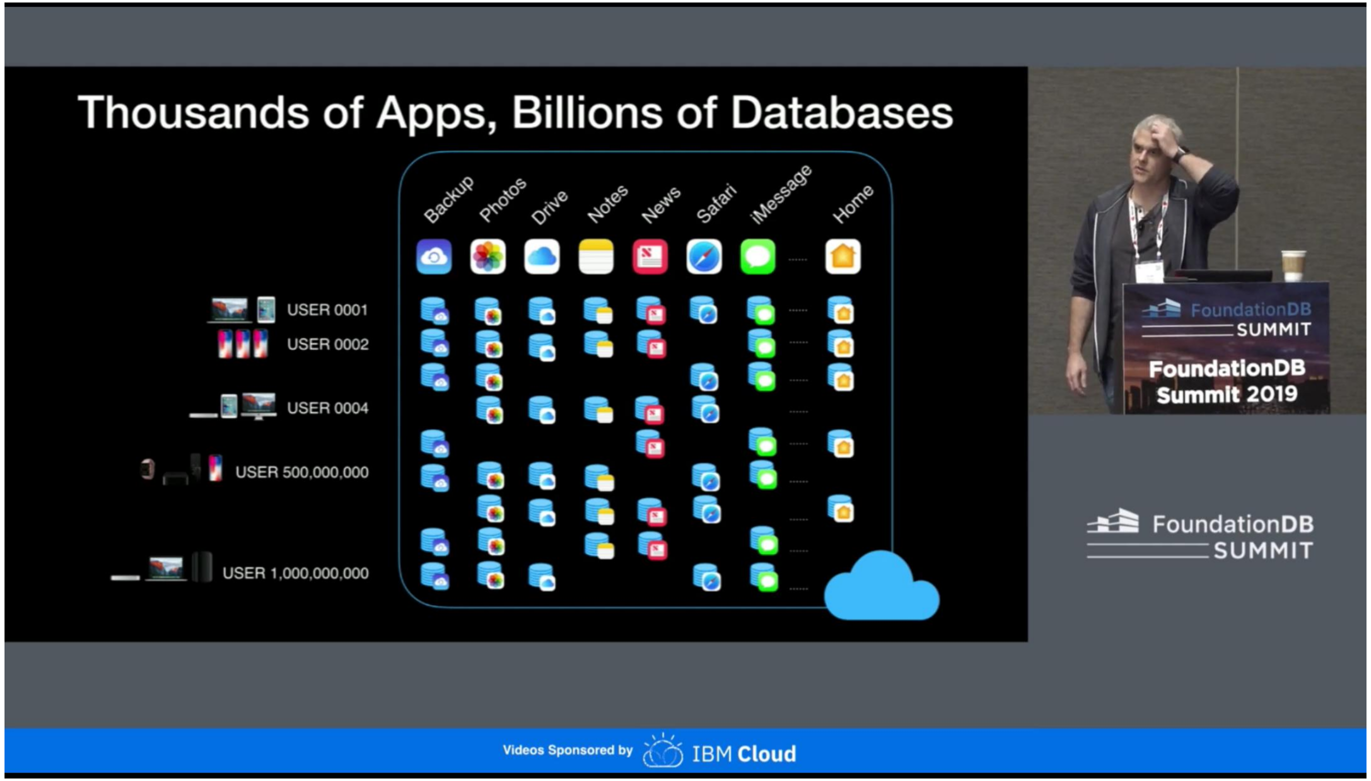


ABSTRACT

The FoundationDB Record Layer is an open source library that provides a record-oriented datastore with semantics similar to a relational database, implemented on top of Foun-dationDB, an ordered, transactional key-value store. The


separate logical database. We demonstrate how the Record Layer is used by CloudKit, Apple's cloud backend service, to provide powerful abstractions to applications serving hundreds of millions of users. CloudKit uses the Record Layer to host billions of independent databases, many with a common schema. Features provided by the Record Layer enable

CloudKit in one slide



CloudKit in more slides

FoundationDB SUMMIT

Using FoundationDB and the FDB Record Layer to Build CloudKit

Scott Gray, Apple

FoundationDB SUMMIT

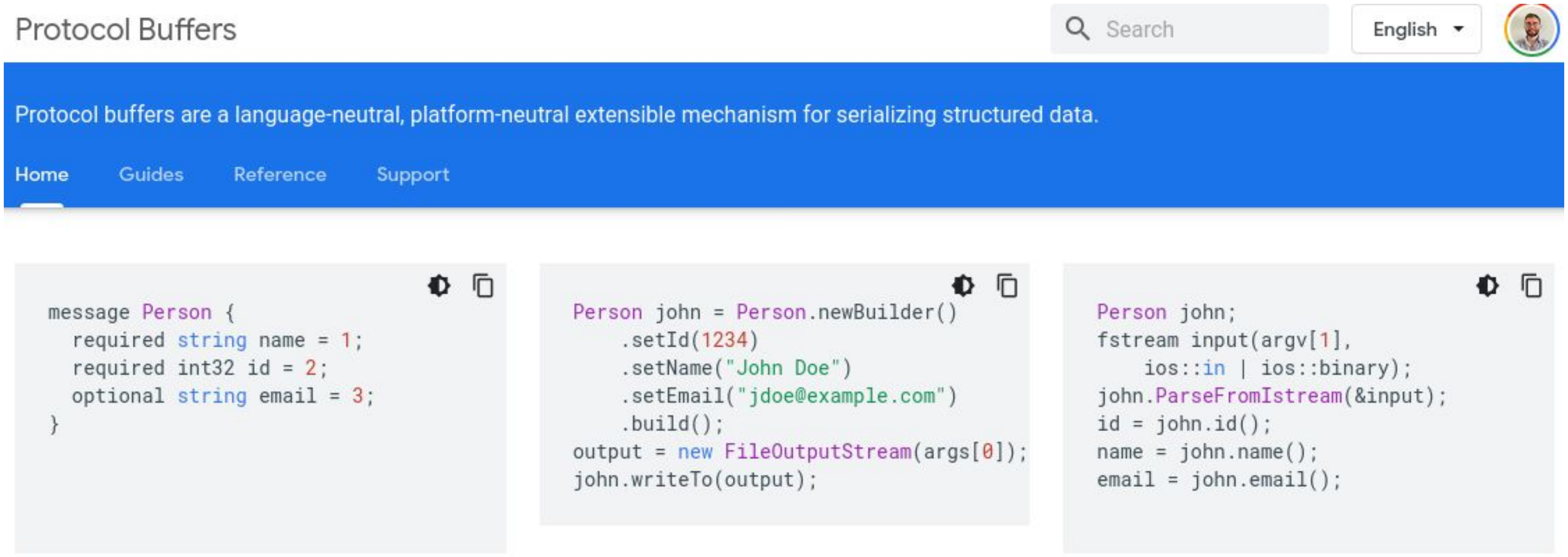
FoundationDB Record Layer: Open Source Structured Storage on FoundationDB

Nicholas Schiefer, Apple

Record Layer

- Open source layer written in Java
- offers:
    - Schema
    - Indexes
    - Stateless design
    - Query planner
    - streaming queries

Handling schemas


Protocol Buffers

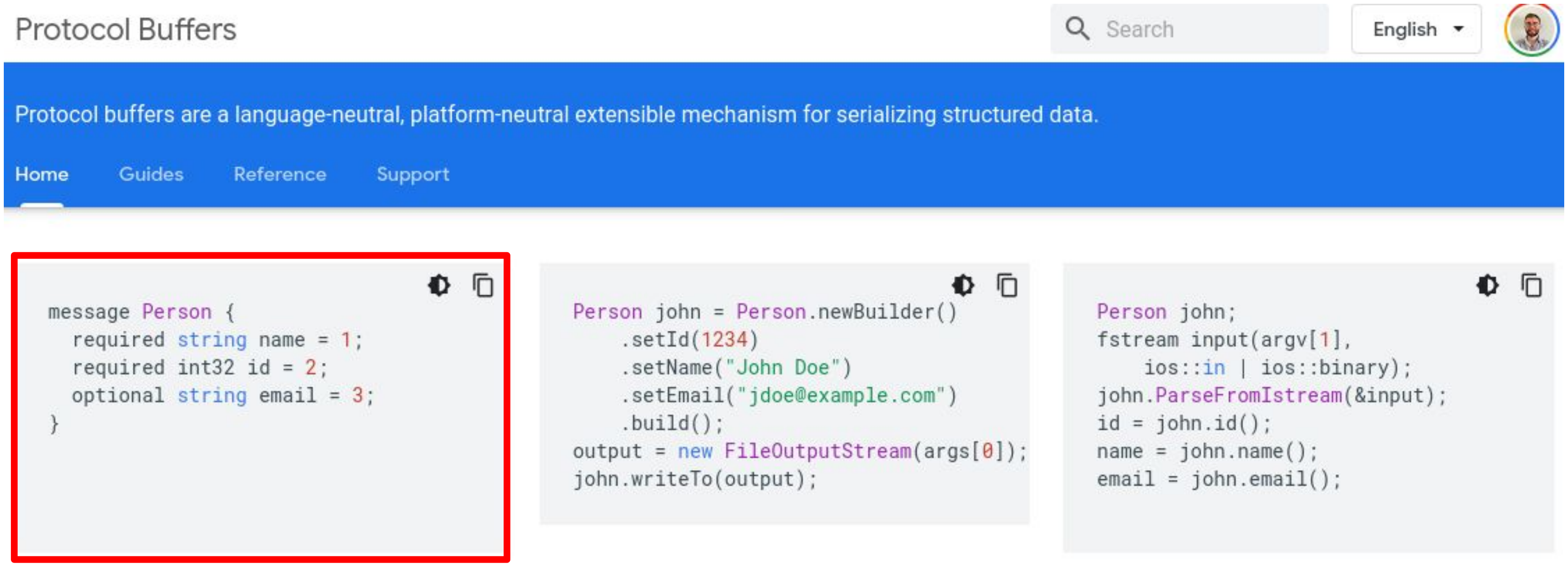
Protocol buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data.

```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;
}
```

```
Person john = Person.newBuilder()
    .setId(1234)
    .setName("John Doe")
    .setEmail("jdoe@example.com")
    .build();
output = new FileOutputStream(args[0]);
john.writeTo(output);
```

```
Person john;
fstream input(argv[1],
    ios::in | ios::binary);
john.ParseFromIstream(&input);
id = john.id();
name = john.name();
email = john.email();
```

Handling schemas



Protocol Buffers

Protocol buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data.

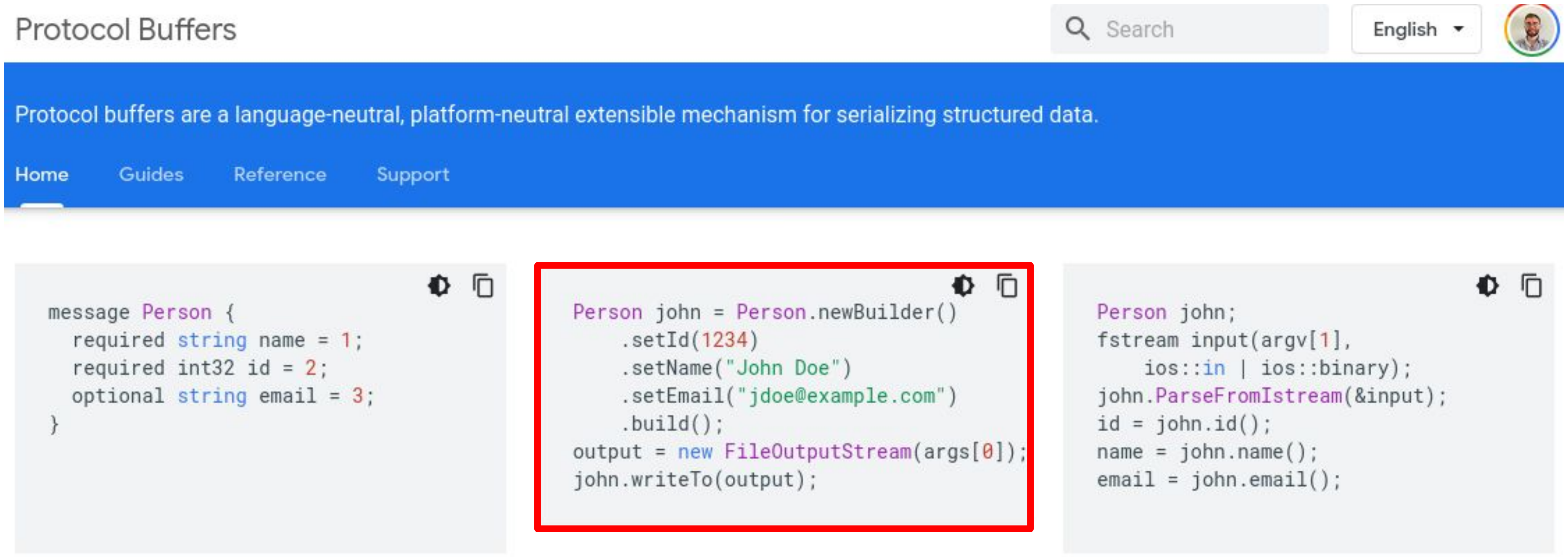
```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;
}
```

```
Person john = Person.newBuilder()
    .setId(1234)
    .setName("John Doe")
    .setEmail("jdoe@example.com")
    .build();
output = new FileOutputStream(args[0]);
john.writeTo(output);
```

```
Person john;
fstream input(argv[1],
    ios::in | ios::binary);
john.ParseFromIstream(&input);
id = john.id();
name = john.name();
email = john.email();
```

Schema

Handling schemas



Protocol buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data.

```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;
}
```

```
Person john = Person.newBuilder()
    .setId(1234)
    .setName("John Doe")
    .setEmail("jdoe@example.com")
    .build();
output = new FileOutputStream(args[0]);
john.writeTo(output);
```

```
Person john;
fstream input(argv[1],
    ios::in | ios::binary);
john.ParseFromIstream(&input);
id = john.id();
name = john.name();
email = john.email();
```

Record

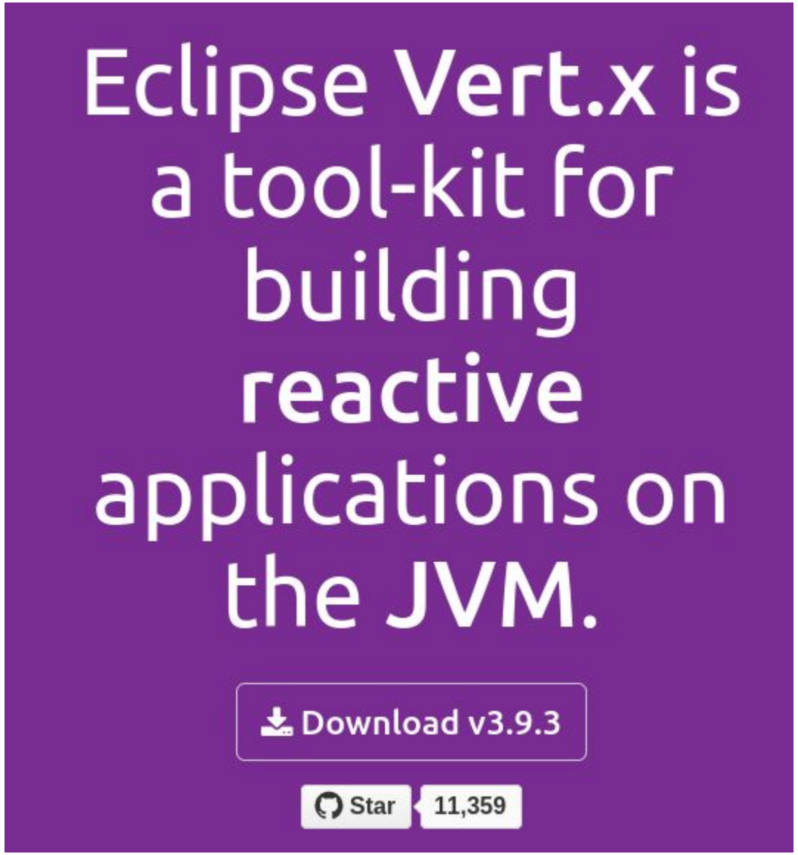
FoundationDB

etcd

- Key-value store written in Go
- Now known as Kubernetes datastore
- use gRPC and Protobuf
- Single-group Raft
- Soon a CNCF graduated project (https://github.com/cncf/toc/pull/541)

etcd

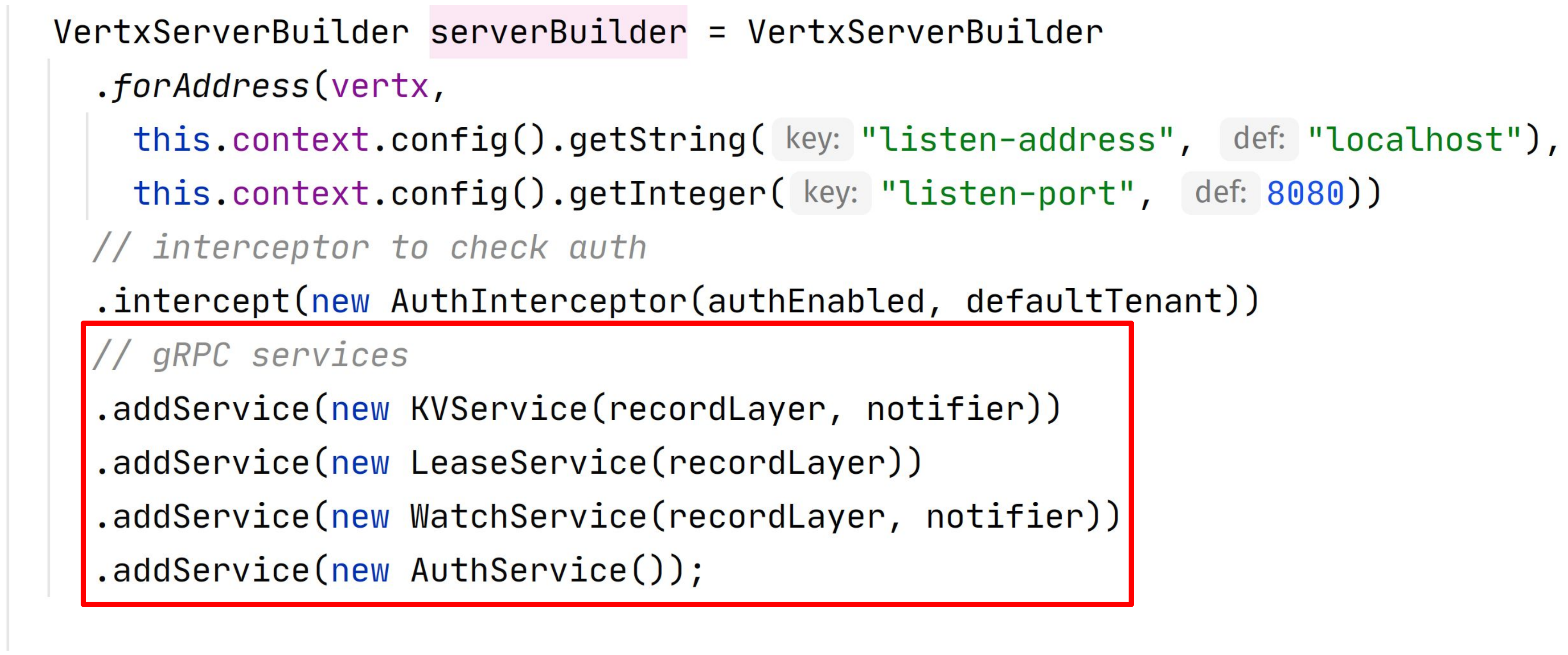
- Key-value store written in Go
- Now known as Kubernetes datastore
- **use gRPC and Protobuf 🤩**
- Single-group Raft
- Soon a CNCF graduated project (https://github.com/cncf/toc/pull/541)


Eclipse Vert.x is a tool-kit for building reactive applications on the JVM.

Implementing ETCD

```
service KV {
  // Range gets the keys in the range from the key-value store.
  rpc Range (RangeRequest) returns (RangeResponse) {
    option (google.api.http) = {
        post: "/v3/kv/range"
        body: "*"
    };
  }
  // ...
```

Implementing ETCD



```
VertxServerBuilder serverBuilder = VertxServerBuilder
    .forAddress(vertx,
        this.context.config().getString( key: "listen-address",  def: "localhost"),
        this.context.config().getInteger( key: "listen-port",  def: 8080))
    // interceptor to check auth
    .intercept(new AuthInterceptor(authEnabled, defaultTenant))
    // gRPC services
    .addService(new KVService(recordLayer, notifier))
    .addService(new LeaseService(recordLayer))
    .addService(new WatchService(recordLayer, notifier))
    .addService(new AuthService());
```

Implementing ETCD

```java
/**
 * KVService corresponds to the KV GRPC service
 */
public class KVService extends KVGrpc.KVVertxImplBase {

  @Override
  public void range(EtcdIoRpcProto.RangeRequest request, Promise<EtcdIoRpcProto.RangeResponse> response) {
    // creating a empty response
    response.complete(EtcdIoRpcProto.RangeResponse.newBuilder().build());
  }
```

FoundationDB

Speaking ETCD

```
// This is the main struct to store keys and values.
message KeyValue {
  // key is the key in bytes. An empty key is not allowed.
  bytes key = 1;
  // create_revision is the revision of last creation on this key.
  int64 create_revision = 2;
  // mod_revision is the revision of last modification on this key.
  int64 mod_revision = 3;
  // version is the version of the key. A deletion resets
  // the version to zero and any modification of the key
  // increases its version.
  int64 version = 4;
  // value is the value held by the key, in bytes.
  bytes value = 5;
  // lease is the ID of the lease that attached to key.
  // When the attached lease expires, the key will be deleted.
  // If lease is 0, then no lease is attached to the key.
  int64 lease = 6;
}
```

Speaking ETCD

```
// This is the main struct to store keys and values.
message KeyValue {
  // key is the key in bytes. An empty key is not allowed.
  bytes key = 1;
  // create_revision is the revision of last creation on this key.
  int64 create_revision = 2;
  // mod_revision is the revision of last modification on this key.
  int64 mod_revision = 3;
  // version is the version of the key. A deletion resets
  // the version to zero and any modification of the key
  // increases its version.
  int64 version = 4;
  // value is the value held by the key, in bytes.
  bytes value = 5;
  // lease is the ID of the lease that attached to key.
  // When the attached lease expires, the key will be deleted.
  // If lease is 0, then no lease is attached to the key.
  int64 lease = 6;
}
```

Speaking ETCD

```
// This is the main struct to store keys and values.
message KeyValue {
  // key is the key in bytes. An empty key is not allowed.
  bytes key = 1;
  // create_revision is the revision of last creation on this key.
  int64 create_revision = 2;
  // mod_revision is the revision of last modification on this key.
  int64 mod_revision = 3;
  // version is the version of the key. A deletion resets
  // the version to zero and any modification of the key
  // increases its version.
  int64 version = 4;
  // value is the value held by the key, in bytes.
  bytes value = 5;
  // lease is the ID of the lease that attached to key.
  // When the attached lease expires, the key will be deleted
  // If lease is 0, then no lease is attached to the key.
  int64 lease = 6;
}
```

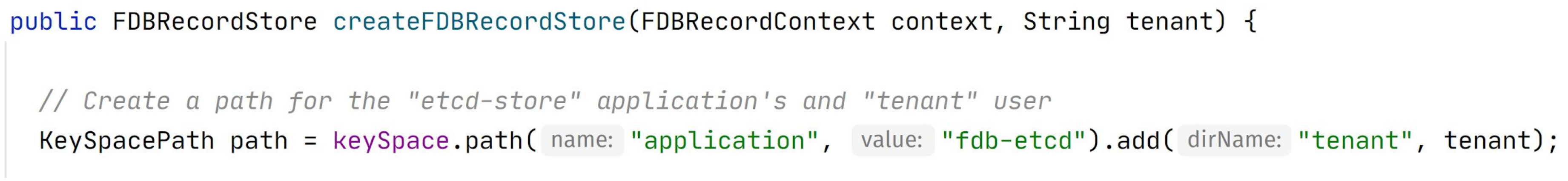
Speaking ETCD

```
// This is the main struct to store keys and values.
message KeyValue {
  // key is the key in bytes. An empty key is not allowed.
  bytes key = 1;
  // create_revision is the revision of last creation on this key.
  int64 create_revision = 2;
  // mod_revision is the revision of last modification on this key.
  int64 mod_revision = 3;
  // version is the version of the key. A deletion resets
  // the version to zero and any modification of the key
  // increases its version.
  int64 version = 4;
  // value is the value held by the key, in bytes.
  bytes value = 5;
  // lease is the ID of the lease that attached to key.
  // When the attached lease expires, the key will be deleted.
  // If lease is 0, then no lease is attached to the key.
  int64 lease = 6;
}
```

Creating a FDBRecordStore

```
public FDBRecordStore createFDBRecordStore(FDBRecordContext context, String tenant) {
```

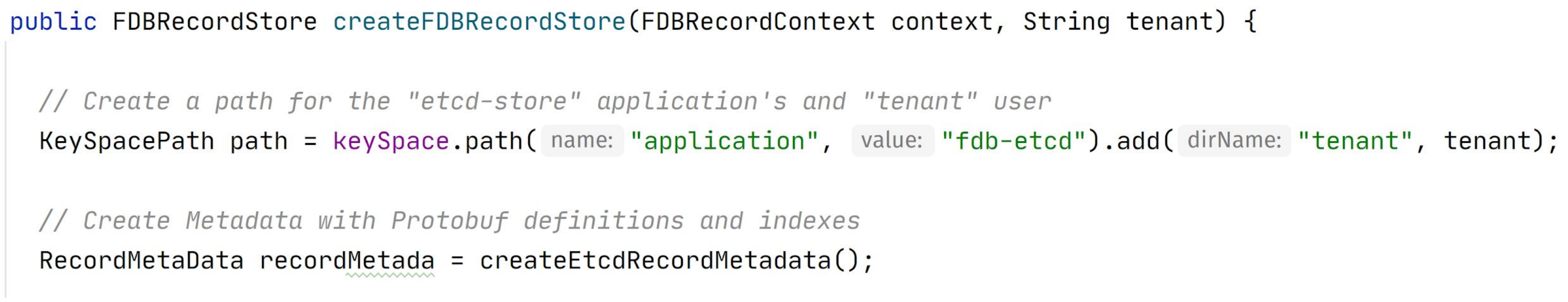
Creating a FDBRecordStore



```java
public FDBRecordStore createFDBRecordStore(FDBRecordContext context, String tenant) {

  // Create a path for the "etcd-store" application's and "tenant" user
  KeySpacePath path = keySpace.path( name: "application",  value: "fdb-etcd").add( dirName: "tenant", tenant);
```

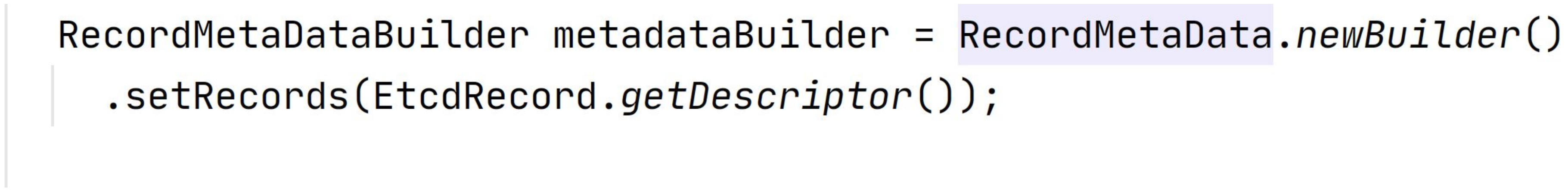
```
application
└── fdb-etcd
    └── tenant
        ├── MyTenant1
        └── MyTenant2
```

Creating a FDBRecordStore



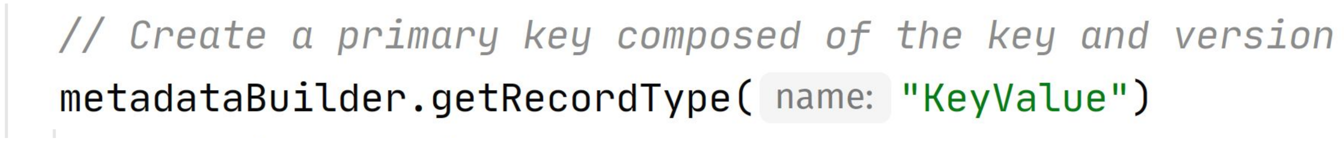
```
public FDBRecordStore createFDBRecordStore(FDBRecordContext context, String tenant) {

  // Create a path for the "etcd-store" application's and "tenant" user
  KeySpacePath path = keySpace.path( name: "application",  value: "fdb-etcd").add( dirName: "tenant", tenant);

  // Create Metadata with Protobuf definitions and indexes
  RecordMetaData recordMetada = createEtcdRecordMetadata();
```

Creating metadata from Protobuf



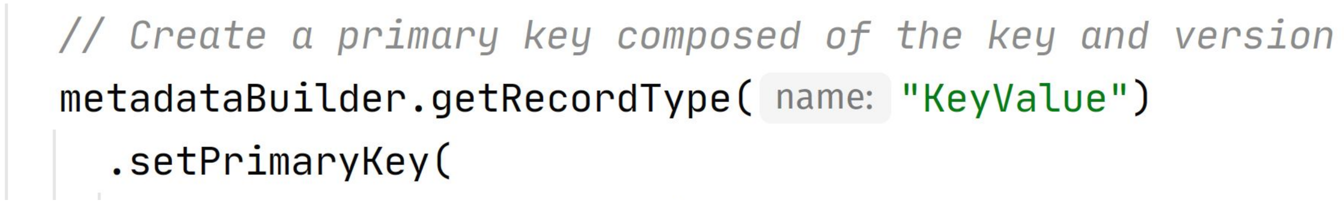
```
RecordMetaDataBuilder metadataBuilder = RecordMetaData.newBuilder()
    .setRecords(EtcdRecord.getDescriptor());
```

Creating metadata from Protobuf



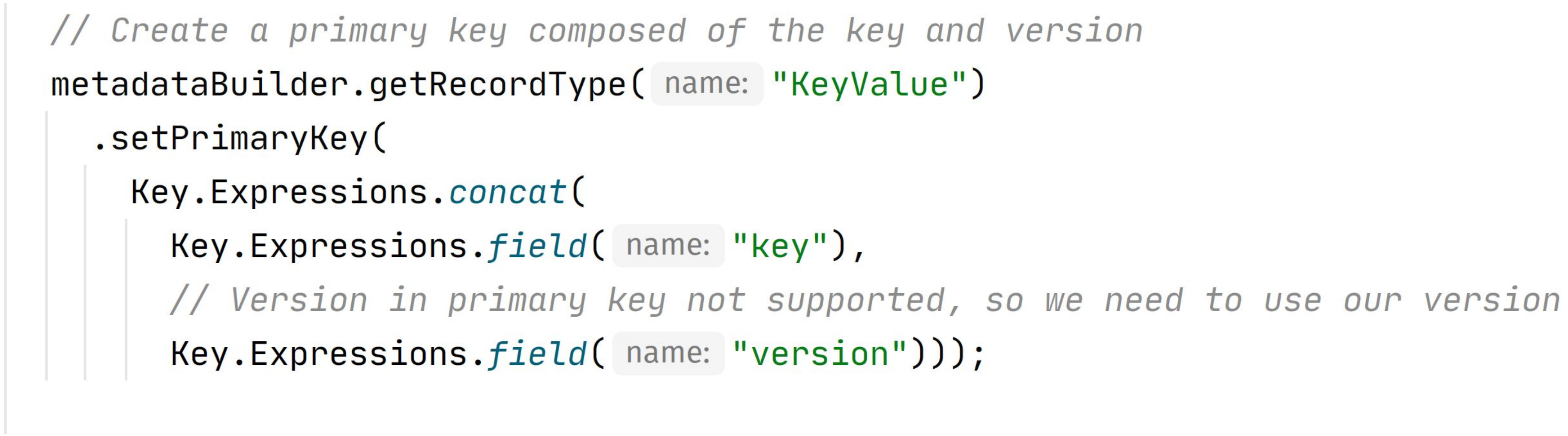
```
// Create a primary key composed of the key and version
metadataBuilder.getRecordType( name: "KeyValue")
```

Creating metadata from Protobuf



```
// Create a primary key composed of the key and version
metadataBuilder.getRecordType( name: "KeyValue")
    .setPrimaryKey(
```

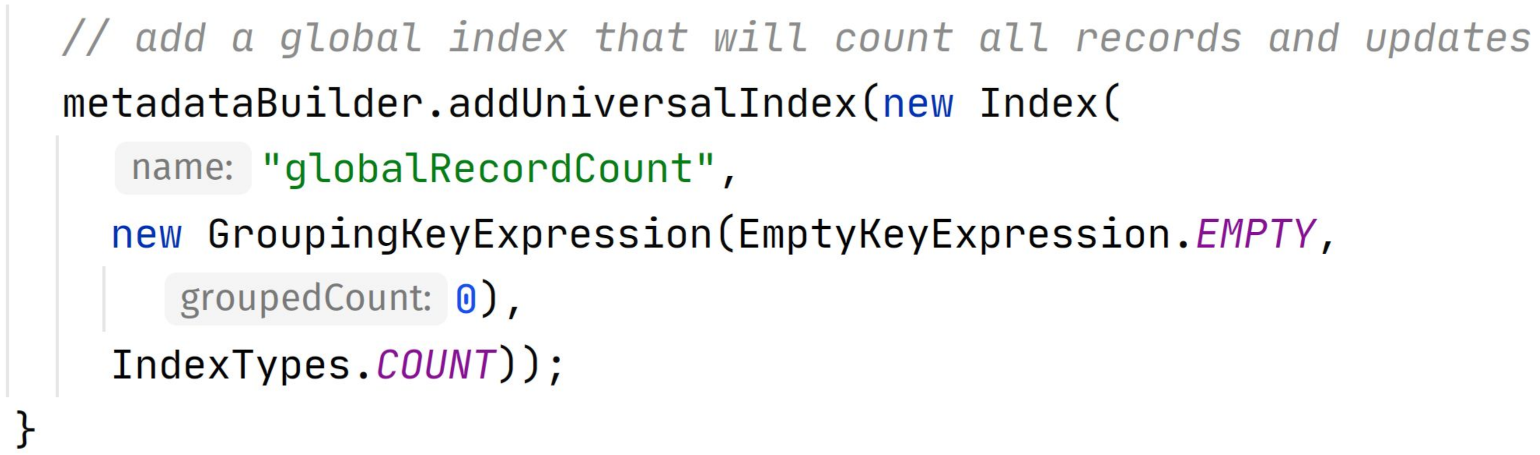
Creating metadata from Protobuf



```
// Create a primary key composed of the key and version
metadataBuilder.getRecordType( name: "KeyValue")
    .setPrimaryKey(
        Key.Expressions.concat(
            Key.Expressions.field( name: "key"),
            // Version in primary key not supported, so we need to use our version
            Key.Expressions.field( name: "version")));
```

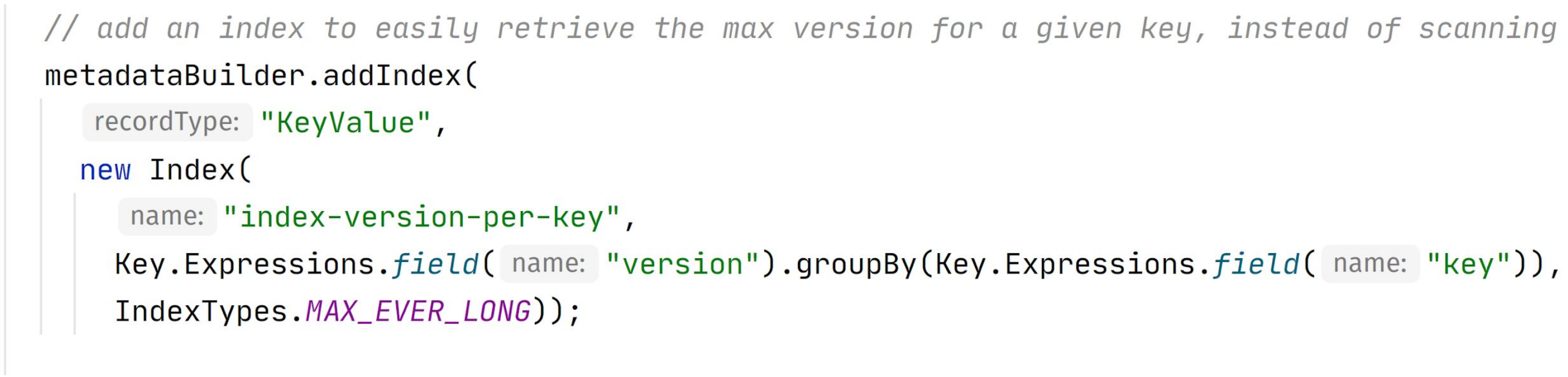
FoundationDB

Adding indexes



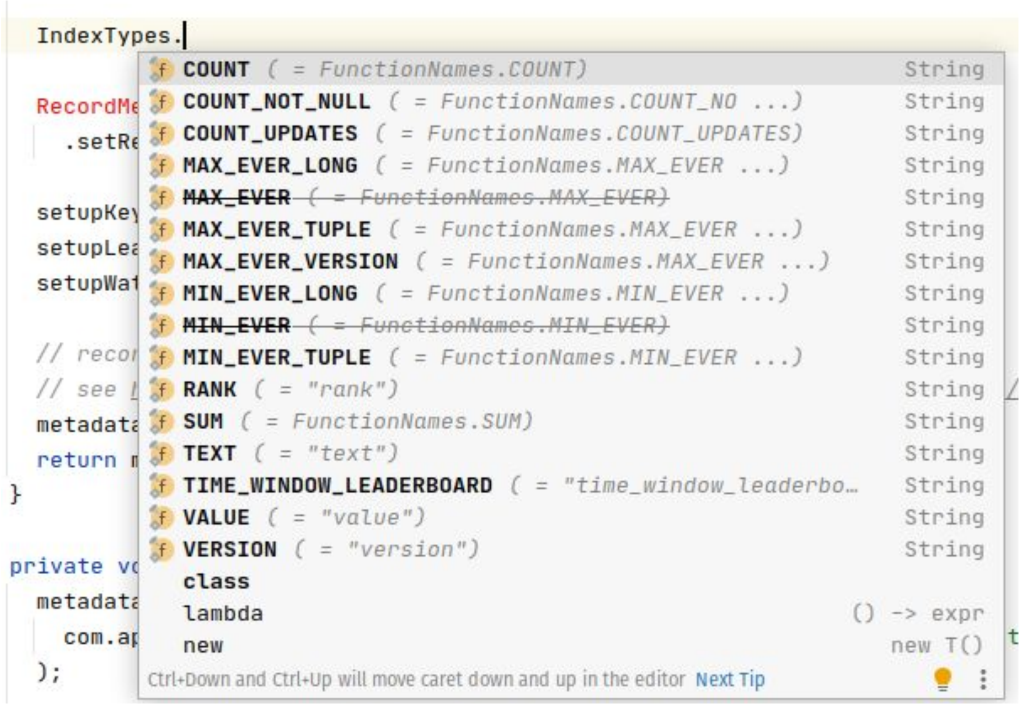
```
  // add a global index that will count all records and updates
  metadataBuilder.addUniversalIndex(new Index(
      name: "globalRecordCount",
      new GroupingKeyExpression(EmptyKeyExpression.EMPTY,
          groupedCount: 0),
      IndexTypes.COUNT));
}
```

Adding indexes



```java
// add an index to easily retrieve the max version for a given key, instead of scanning
metadataBuilder.addIndex(
    recordType: "KeyValue",
    new Index(
        name: "index-version-per-key",
        Key.Expressions.field( name: "version").groupBy(Key.Expressions.field( name: "key")),
        IndexTypes.MAX_EVER_LONG));
```

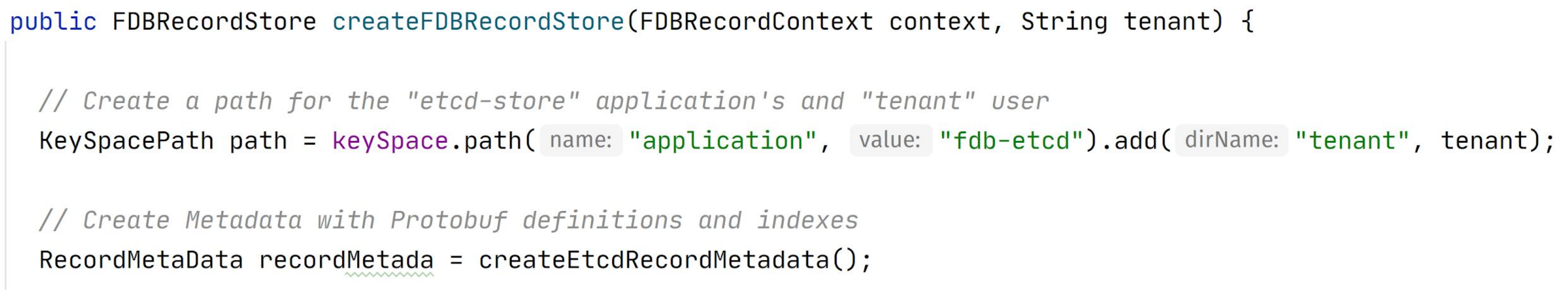
Adding indexes



```
IndexTypes.

RecordMe
  .setRe

setupKey
setupLea
setupWat

// recor
// see 
metadata
return m
}

private vo
  metadata
    com.ap
  );
```

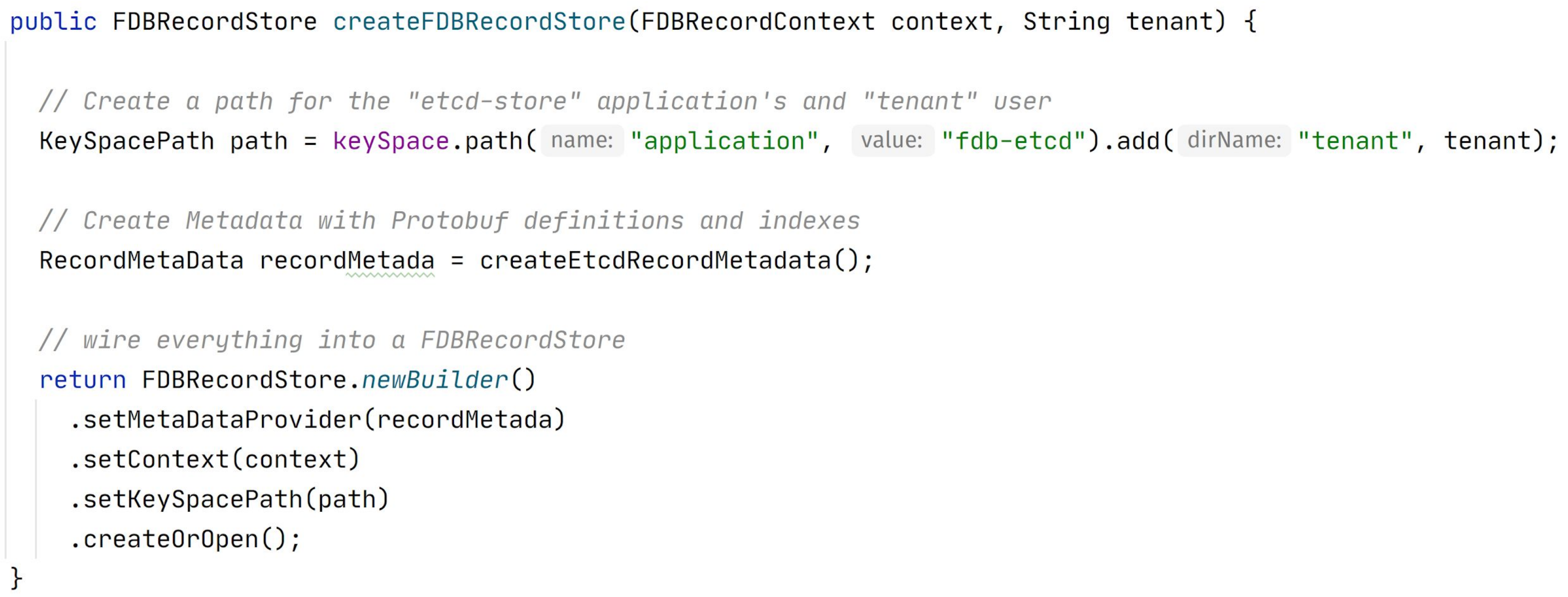
```
COUNT ( = FunctionNames.COUNT)                                  String
COUNT_NOT_NULL ( = FunctionNames.COUNT_NO ...)                  String
COUNT_UPDATES ( = FunctionNames.COUNT_UPDATES)                  String
MAX_EVER_LONG ( = FunctionNames.MAX_EVER ...)                   String
~~MAX_EVER ( = FunctionNames.MAX_EVER)~~                        String
MAX_EVER_TUPLE ( = FunctionNames.MAX_EVER ...)                  String
MAX_EVER_VERSION ( = FunctionNames.MAX_EVER ...)                String
MIN_EVER_LONG ( = FunctionNames.MIN_EVER ...)                   String
~~MIN_EVER ( = FunctionNames.MIN_EVER)~~                        String
MIN_EVER_TUPLE ( = FunctionNames.MIN_EVER ...)                  String
RANK ( = "rank")                                                String
SUM ( = FunctionNames.SUM)                                      String
TEXT ( = "text")                                                String
TIME_WINDOW_LEADERBOARD ( = "time_window_leaderbo…              String
VALUE ( = "value")                                              String
VERSION ( = "version")                                          String
class
lambda                                                      () -> expr
new                                                          new T()
Ctrl+Down and Ctrl+Up will move caret down and up in the editor Next Tip
```

Creating a FDBRecordStore



```
public FDBRecordStore createFDBRecordStore(FDBRecordContext context, String tenant) {

  // Create a path for the "etcd-store" application's and "tenant" user
  KeySpacePath path = keySpace.path( name: "application",  value: "fdb-etcd").add( dirName: "tenant", tenant);

  // Create Metadata with Protobuf definitions and indexes
  RecordMetaData recordMetada = createEtcdRecordMetadata();
```

Creating a FDBRecordStore

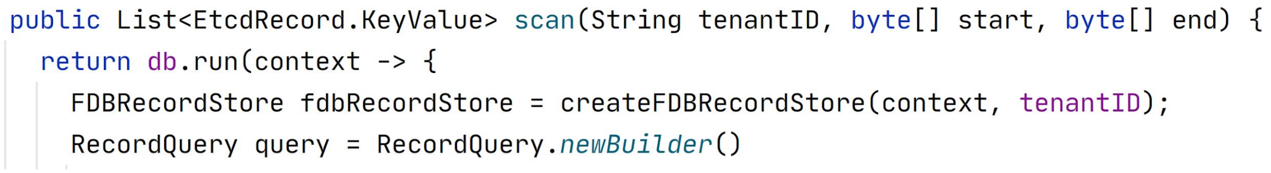


```java
public FDBRecordStore createFDBRecordStore(FDBRecordContext context, String tenant) {

  // Create a path for the "etcd-store" application's and "tenant" user
  KeySpacePath path = keySpace.path( name: "application",  value: "fdb-etcd").add( dirName: "tenant", tenant);

  // Create Metadata with Protobuf definitions and indexes
  RecordMetaData recordMetada = createEtcdRecordMetadata();

  // wire everything into a FDBRecordStore
  return FDBRecordStore.newBuilder()
    .setMetaDataProvider(recordMetada)
    .setContext(context)
    .setKeySpacePath(path)
    .createOrOpen();
}
```

Inserting a record

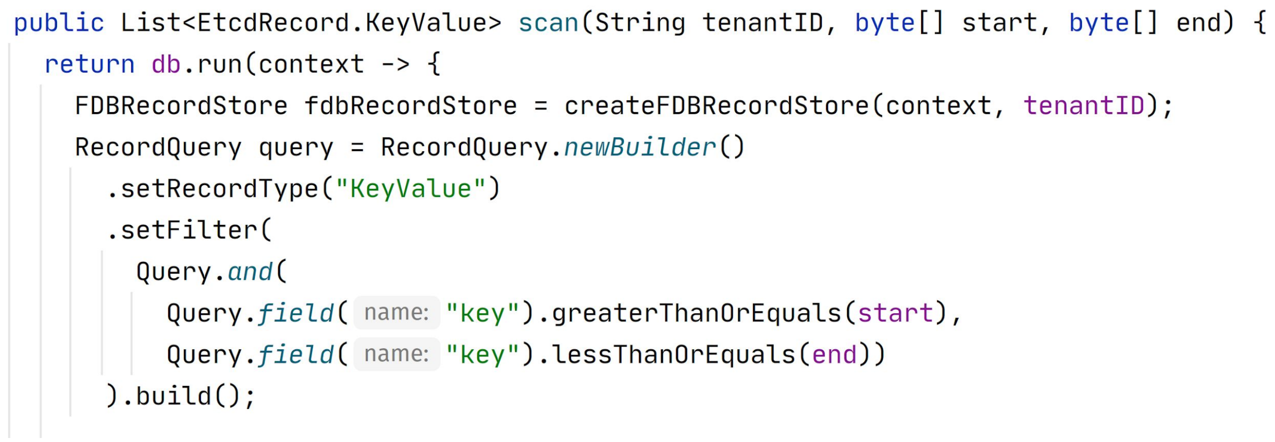
```java
public EtcdRecord.KeyValue put(String tenantID, EtcdRecord.KeyValue record) {
  return db.run(context -> {
    FDBRecordStore fdbRecordStore = createFDBRecordStore(context, tenantID);
    fdbRecordStore.saveRecord(record);
    context.commit();
    return record;
  });
}
```

Querying records



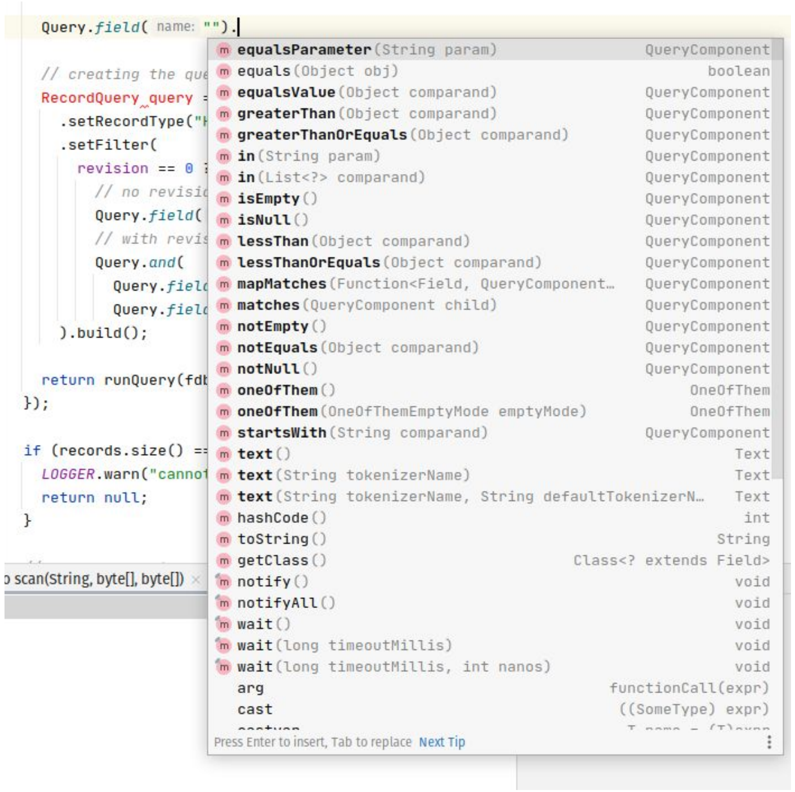
```java
public List<EtcdRecord.KeyValue> scan(String tenantID, byte[] start, byte[] end) {
  return db.run(context -> {
    FDBRecordStore fdbRecordStore = createFDBRecordStore(context, tenantID);
    RecordQuery query = RecordQuery.newBuilder()
```

Querying records



```java
public List<EtcdRecord.KeyValue> scan(String tenantID, byte[] start, byte[] end) {
  return db.run(context -> {
    FDBRecordStore fdbRecordStore = createFDBRecordStore(context, tenantID);
    RecordQuery query = RecordQuery.newBuilder()
      .setRecordType("KeyValue")
      .setFilter(
        Query.and(
          Query.field( name: "key").greaterThanOrEquals(start),
          Query.field( name: "key").lessThanOrEquals(end))
      ).build();
```

Many operations supported



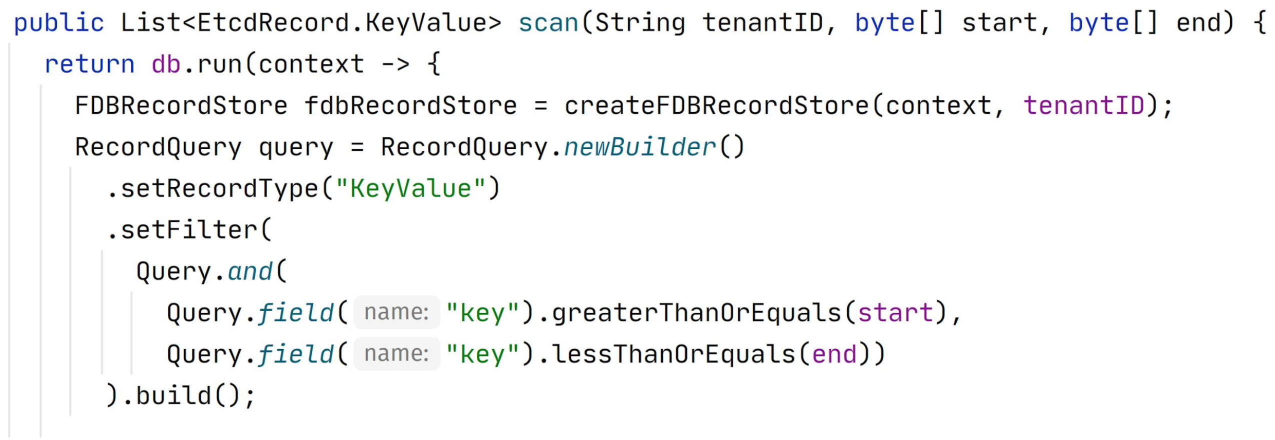
```
  Query.field( name: "").

  // creating the que
  RecordQuery query =
    .setRecordType("H
    .setFilter(
      revision == 0 ?
        // no revisio
        Query.field(
        // with revis
        Query.and(
          Query.fiel
          Query.fiel
    ).build();

  return runQuery(fdb
});

if (records.size() ==
  LOGGER.warn("cannot
  return null;
}
```

scan(String, byte[], byte[]) ×

```
m equalsParameter(String param)                               QueryComponent
m equals(Object obj)                                                 boolean
m equalsValue(Object comparand)                               QueryComponent
m greaterThan(Object comparand)                               QueryComponent
m greaterThanOrEquals(Object comparand)                       QueryComponent
m in(String param)                                            QueryComponent
m in(List<?> comparand)                                       QueryComponent
m isEmpty()                                                   QueryComponent
m isNull()                                                    QueryComponent
m lessThan(Object comparand)                                  QueryComponent
m lessThanOrEquals(Object comparand)                          QueryComponent
m mapMatches(Function<Field, QueryComponent…                  QueryComponent
m matches(QueryComponent child)                               QueryComponent
m notEmpty()                                                  QueryComponent
m notEquals(Object comparand)                                 QueryComponent
m notNull()                                                   QueryComponent
m oneOfThem()                                                      OneOfThem
m oneOfThem(OneOfThemEmptyMode emptyMode)                          OneOfThem
m startsWith(String comparand)                                QueryComponent
m text()                                                                Text
m text(String tokenizerName)                                            Text
m text(String tokenizerName, String defaultTokenizerN…                  Text
m hashCode()                                                             int
m toString()                                                          String
m getClass()                                        Class<? extends Field>
m notify()                                                              void
m notifyAll()                                                           void
m wait()                                                                void
m wait(long timeoutMillis)                                              void
m wait(long timeoutMillis, int nanos)                                   void
  arg                                                     functionCall(expr)
  cast                                                     ((SomeType) expr)
```

Press Enter to insert, Tab to replace Next Tip

Querying records

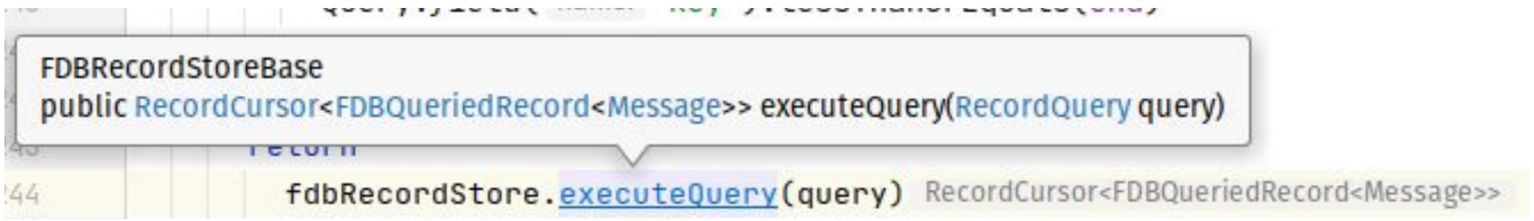


```
public List<EtcdRecord.KeyValue> scan(String tenantID, byte[] start, byte[] end) {
  return db.run(context -> {
    FDBRecordStore fdbRecordStore = createFDBRecordStore(context, tenantID);
    RecordQuery query = RecordQuery.newBuilder()
      .setRecordType("KeyValue")
      .setFilter(
        Query.and(
          Query.field( name: "key").greaterThanOrEquals(start),
          Query.field( name: "key").lessThanOrEquals(end))
      ).build();
```

Querying (async) records

```java
public List<EtcdRecord.KeyValue> scan(String tenantID, byte[] start, byte[] end) {
  return db.run(context -> {
    FDBRecordStore fdbRecordStore = createFDBRecordStore(context, tenantID);
    RecordQuery query = RecordQuery.newBuilder()
      .setRecordType("KeyValue")
      .setFilter(
        Query.and(
          Query.field( name: "key").greaterThanOrEquals(start),
          Query.field( name: "key").lessThanOrEquals(end))
      ).build();

    return fdbRecordStore
      // this returns an asynchronous cursor over the results of our query
      .executeQuery(query)
```

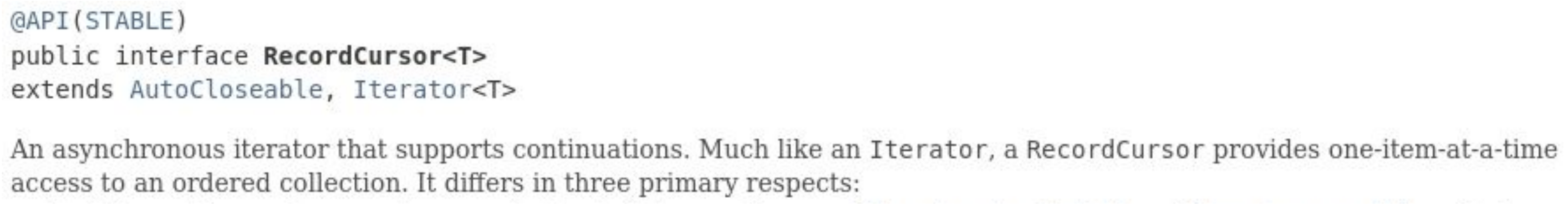
Querying (async) records



```
FDBRecordStoreBase
public RecordCursor<FDBQueriedRecord<Message>> executeQuery(RecordQuery query)
```

```
        fdbRecordStore.executeQuery(query) RecordCursor<FDBQueriedRecord<Message>>
```

Querying (async) records

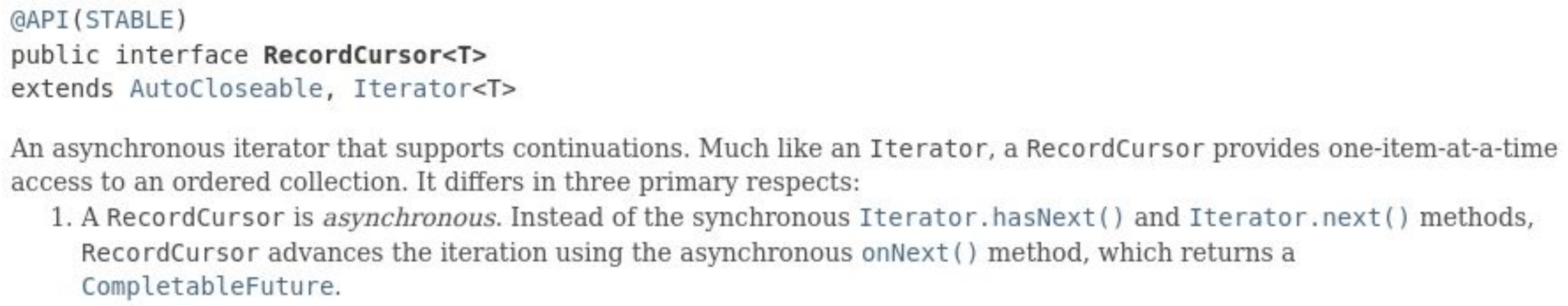


```
@API(STABLE)
public interface RecordCursor<T>
extends AutoCloseable, Iterator<T>
```

An asynchronous iterator that supports continuations. Much like an `Iterator`, a `RecordCursor` provides one-item-at-a-time access to an ordered collection. It differs in three primary respects:

FoundationDB

Querying (async) records

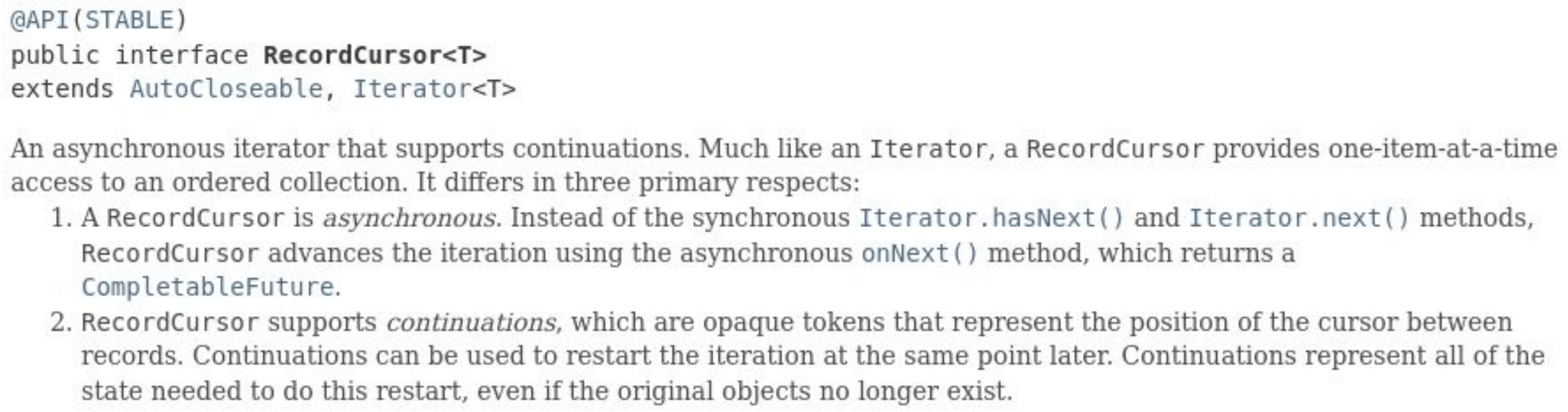


```
@API(STABLE)
public interface RecordCursor<T>
extends AutoCloseable, Iterator<T>
```

An asynchronous iterator that supports continuations. Much like an `Iterator`, a `RecordCursor` provides one-item-at-a-time access to an ordered collection. It differs in three primary respects:

1. A `RecordCursor` is *asynchronous*. Instead of the synchronous `Iterator.hasNext()` and `Iterator.next()` methods, `RecordCursor` advances the iteration using the asynchronous `onNext()` method, which returns a `CompletableFuture`.

Querying (maybe later) records

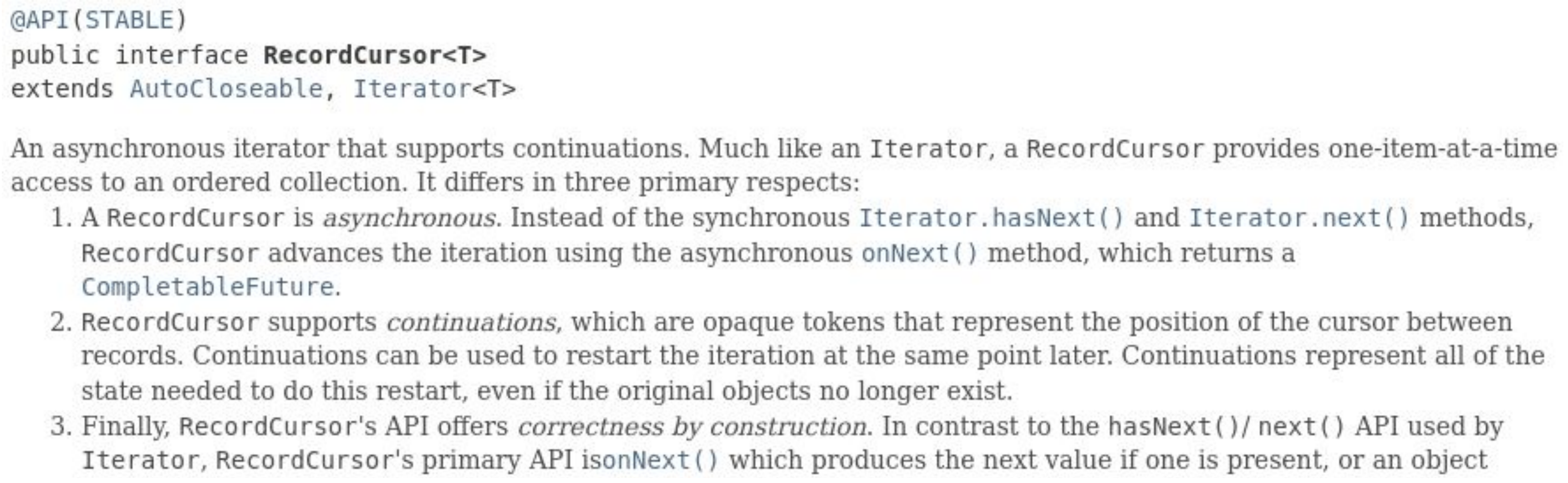


```
@API(STABLE)
public interface RecordCursor<T>
extends AutoCloseable, Iterator<T>
```

An asynchronous iterator that supports continuations. Much like an `Iterator`, a `RecordCursor` provides one-item-at-a-time access to an ordered collection. It differs in three primary respects:

1. A `RecordCursor` is *asynchronous*. Instead of the synchronous `Iterator.hasNext()` and `Iterator.next()` methods, `RecordCursor` advances the iteration using the asynchronous `onNext()` method, which returns a `CompletableFuture`.
2. `RecordCursor` supports *continuations*, which are opaque tokens that represent the position of the cursor between records. Continuations can be used to restart the iteration at the same point later. Continuations represent all of the state needed to do this restart, even if the original objects no longer exist.

Querying (correct) records



```
@API(STABLE)
public interface RecordCursor<T>
extends AutoCloseable, Iterator<T>
```

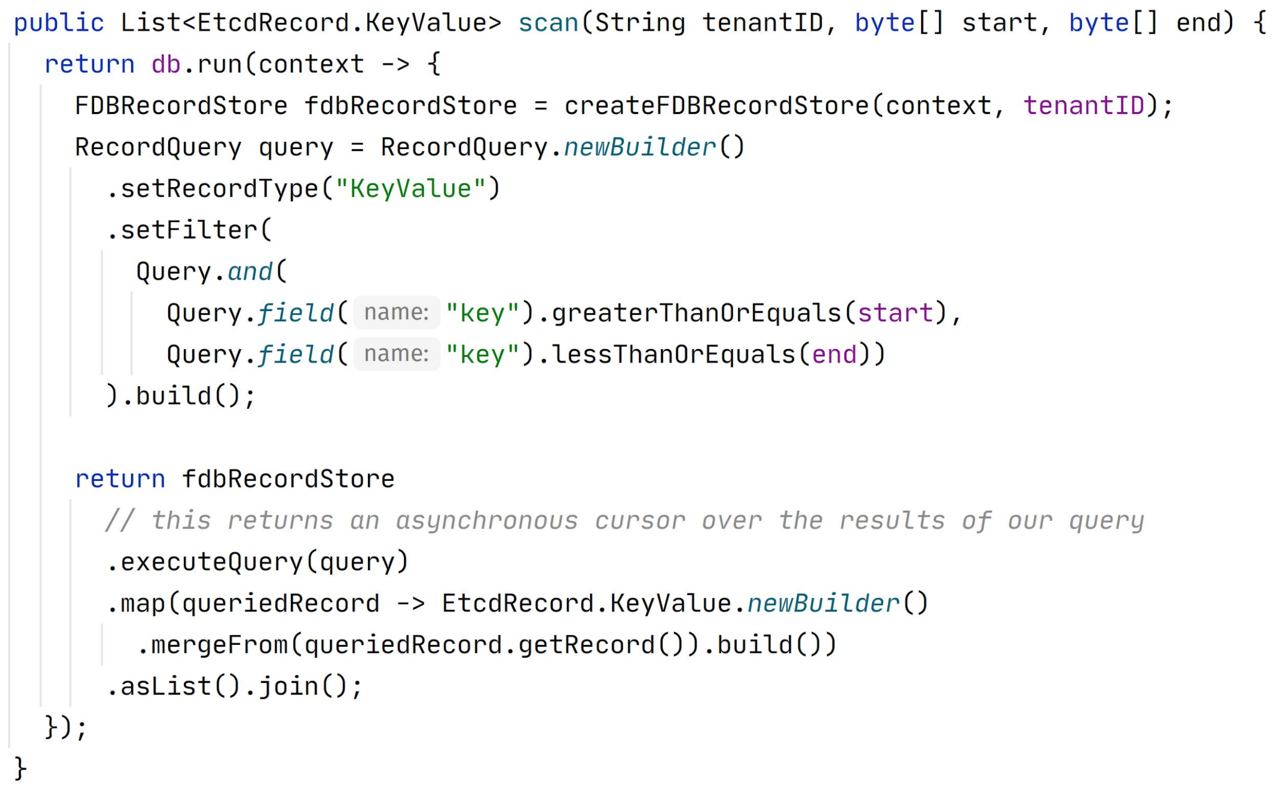
An asynchronous iterator that supports continuations. Much like an `Iterator`, a `RecordCursor` provides one-item-at-a-time access to an ordered collection. It differs in three primary respects:

1. A `RecordCursor` is *asynchronous*. Instead of the synchronous `Iterator.hasNext()` and `Iterator.next()` methods, `RecordCursor` advances the iteration using the asynchronous `onNext()` method, which returns a `CompletableFuture`.
2. `RecordCursor` supports *continuations*, which are opaque tokens that represent the position of the cursor between records. Continuations can be used to restart the iteration at the same point later. Continuations represent all of the state needed to do this restart, even if the original objects no longer exist.
3. Finally, `RecordCursor`'s API offers *correctness by construction*. In contrast to the `hasNext()`/ `next()` API used by `Iterator`, `RecordCursor`'s primary API is`onNext()` which produces the next value if one is present, or an object

Querying records

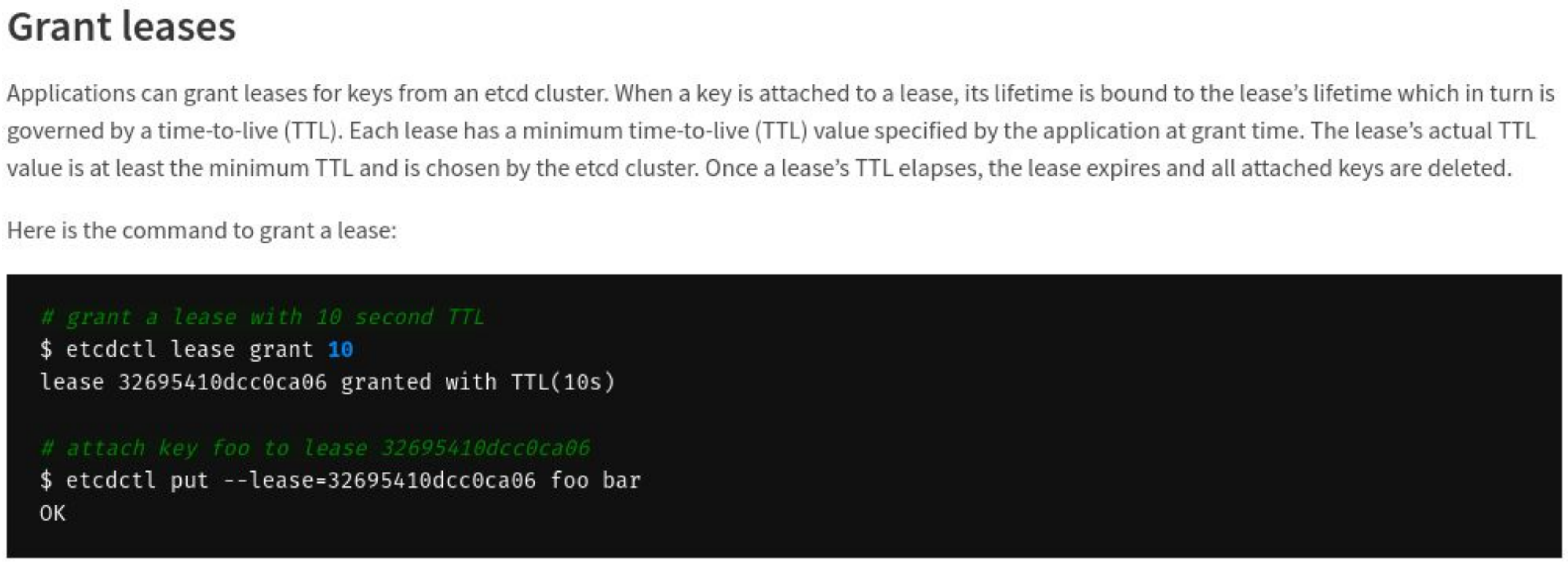
```
public List<EtcdRecord.KeyValue> scan(String tenantID, byte[] start, byte[] end) {
  return db.run(context -> {
    FDBRecordStore fdbRecordStore = createFDBRecordStore(context, tenantID);
    RecordQuery query = RecordQuery.newBuilder()
      .setRecordType("KeyValue")
      .setFilter(
        Query.and(
          Query.field( name: "key").greaterThanOrEquals(start),
          Query.field( name: "key").lessThanOrEquals(end))
      ).build();

    return fdbRecordStore
      // this returns an asynchronous cursor over the results of our query
      .executeQuery(query)
```

Querying records



```
public List<EtcdRecord.KeyValue> scan(String tenantID, byte[] start, byte[] end) {
  return db.run(context -> {
    FDBRecordStore fdbRecordStore = createFDBRecordStore(context, tenantID);
    RecordQuery query = RecordQuery.newBuilder()
      .setRecordType("KeyValue")
      .setFilter(
        Query.and(
          Query.field( name: "key").greaterThanOrEquals(start),
          Query.field( name: "key").lessThanOrEquals(end))
      ).build();

    return fdbRecordStore
      // this returns an asynchronous cursor over the results of our query
      .executeQuery(query)
      .map(queriedRecord -> EtcdRecord.KeyValue.newBuilder()
        .mergeFrom(queriedRecord.getRecord()).build())
      .asList().join();
  });
}
```

What is a Lease?



Grant leases

Applications can grant leases for keys from an etcd cluster. When a key is attached to a lease, its lifetime is bound to the lease's lifetime which in turn is governed by a time-to-live (TTL). Each lease has a minimum time-to-live (TTL) value specified by the application at grant time. The lease's actual TTL value is at least the minimum TTL and is chosen by the etcd cluster. Once a lease's TTL elapses, the lease expires and all attached keys are deleted.

Here is the command to grant a lease:

```
# grant a lease with 10 second TTL
$ etcdctl lease grant 10
lease 32695410dcc0ca06 granted with TTL(10s)

# attach key foo to lease 32695410dcc0ca06
$ etcdctl put --lease=32695410dcc0ca06 foo bar
OK
```

FoundationDB

Lease has his own service to implement

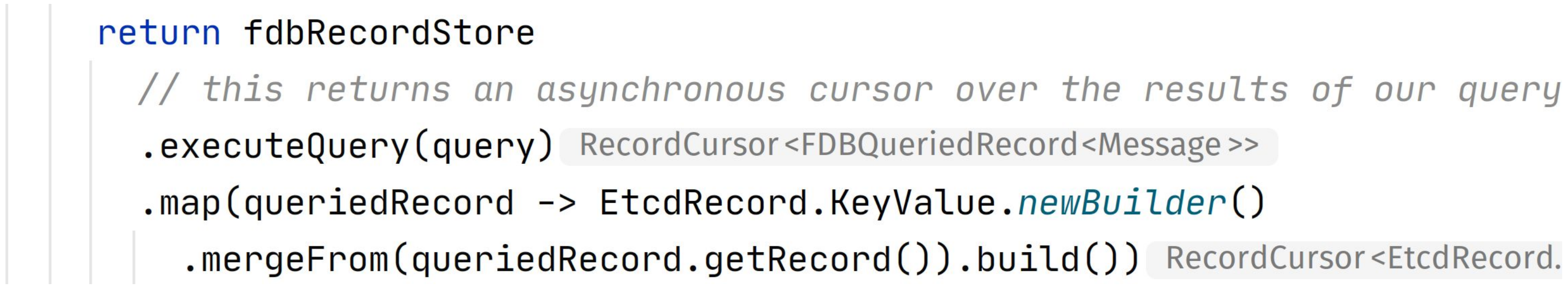
```
service Lease {
  // LeaseGrant creates a lease which expires if the server does not receive a keepAlive
  // within a given time to live period. All keys attached to the lease will be expired and
  // deleted if the lease expires. Each expired key generates a delete event in the event history.
  rpc LeaseGrant (LeaseGrantRequest) returns (LeaseGrantResponse) {
    option (google.api.http) = {
        post: "/v3/lease/grant"
        body: "*"
    };
  }
```

How to handle lease?

1. Store Lease as another RecordType
2. During Put, checking Lease validity
3. During query, checking if lease is expired

How to handle lease?

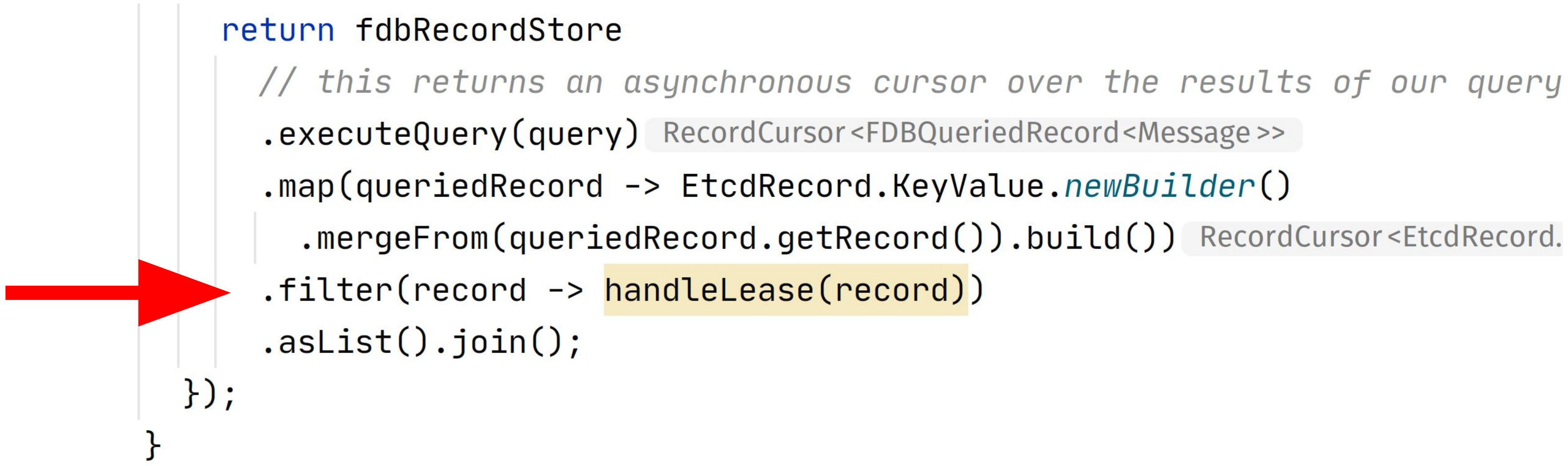
1. Store Lease as another RecordType
2. During Put, checking Lease validity
3. During query, checking if lease is expired



```
return fdbRecordStore
    // this returns an asynchronous cursor over the results of our query
    .executeQuery(query) RecordCursor<FDBQueriedRecord<Message>>
    .map(queriedRecord -> EtcdRecord.KeyValue.newBuilder()
        .mergeFrom(queriedRecord.getRecord()).build()) RecordCursor<EtcdRecord.
```

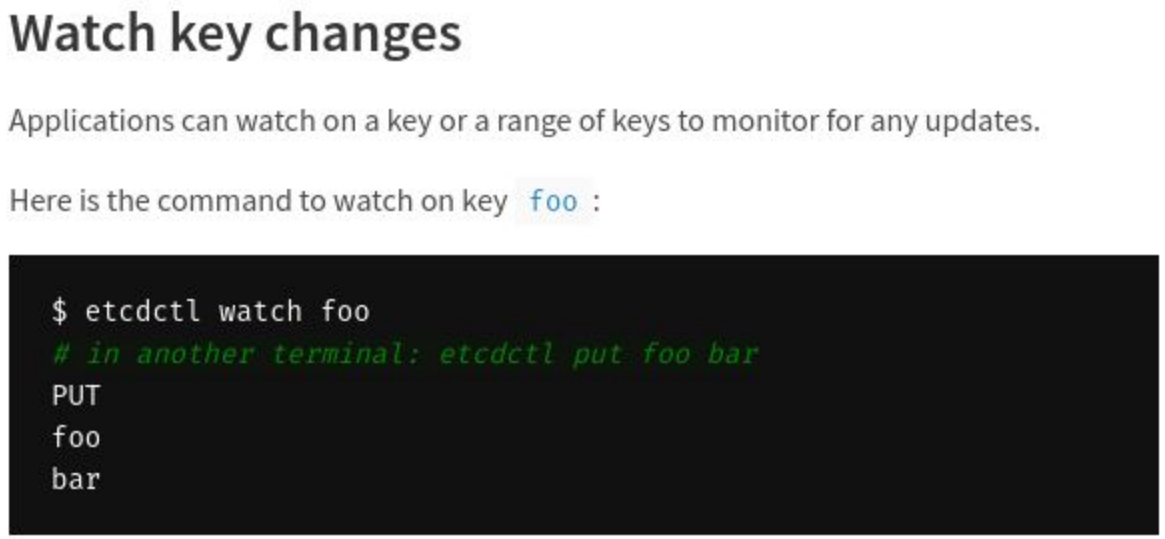
How to handle lease?

1. Store Lease as another RecordType
2. During Put, checking Lease validity
3. During query, checking if lease is expired



```
        return fdbRecordStore
          // this returns an asynchronous cursor over the results of our query
          .executeQuery(query) RecordCursor<FDBQueriedRecord<Message>>
          .map(queriedRecord -> EtcdRecord.KeyValue.newBuilder()
            .mergeFrom(queriedRecord.getRecord()).build()) RecordCursor<EtcdRecord.
          .filter(record -> handleLease(record))
          .asList().join();
    });
}
```

What is a Watch?



Watch key changes

Applications can watch on a key or a range of keys to monitor for any updates.

Here is the command to watch on key `foo` :

```
$ etcdctl watch foo
# in another terminal: etcdctl put foo bar
PUT
foo
bar
```

How to handle a Watch?

```
service Watch {
  // Watch watches for events happening or that have happened. Both input and output
  // are streams; the input stream is for creating and canceling watchers and the output
  // stream sends events. One watch RPC can watch on multiple key ranges, streaming events
  // for several watches at once. The entire event history can be watched starting from the
  // last compaction revision.
  rpc Watch (stream WatchRequest) returns (stream WatchResponse) {
    option (google.api.http) = {
        post: "/v3/watch"
        body: "*"
    };
  }
}
```

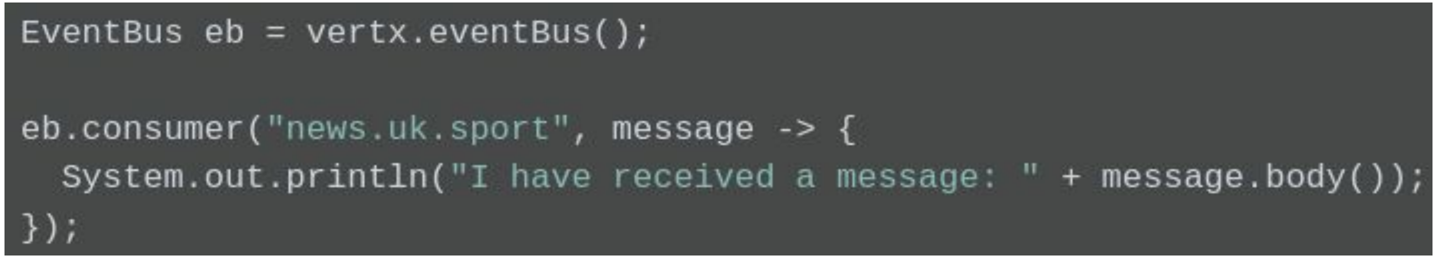
How to handle Watch?

1. Store Watch as another RecordType

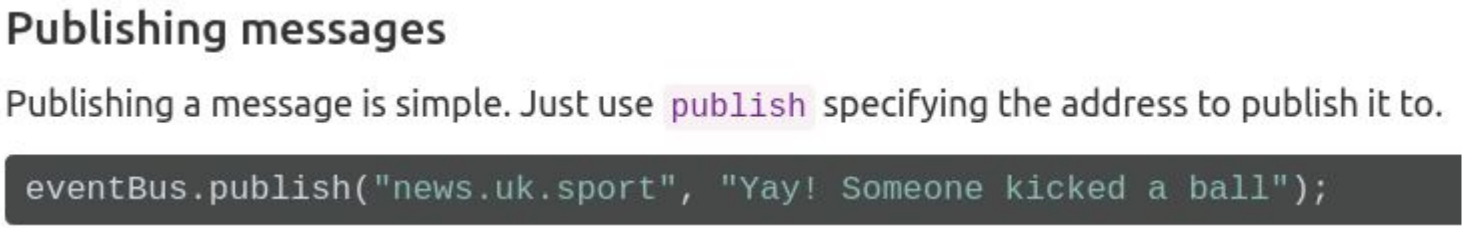
```
message Watch {
  // key is the key to register for watching.
  bytes key = 1;

  // range_end is the end of the range [key, range_end) to watch. If range_end is not given,
  // only the key argument is watched. If range_end is equal to '\0', all keys greater than
  // or equal to the key argument are watched.
  // If the range_end is one bit larger than the given key,
  // then all keys with the prefix (the given key) will be watched.
  bytes range_end = 2;

  // If watch_id is provided and non-zero, it will be assigned to this watcher.
  // Since creating a watcher in etcd is not a synchronous operation,
  // this can be used ensure that ordering is correct when creating multiple
  // watchers on the same stream. Creating a watcher with an ID already in
  // use on the stream will cause an error to be returned.
  int64 watch_id = 7;
}
```

How to handle Watch?

1. Store Watch as another RecordType
2. Start listening to the event-bus



```
EventBus eb = vertx.eventBus();

eb.consumer("news.uk.sport", message -> {
  System.out.println("I have received a message: " + message.body());
});
```



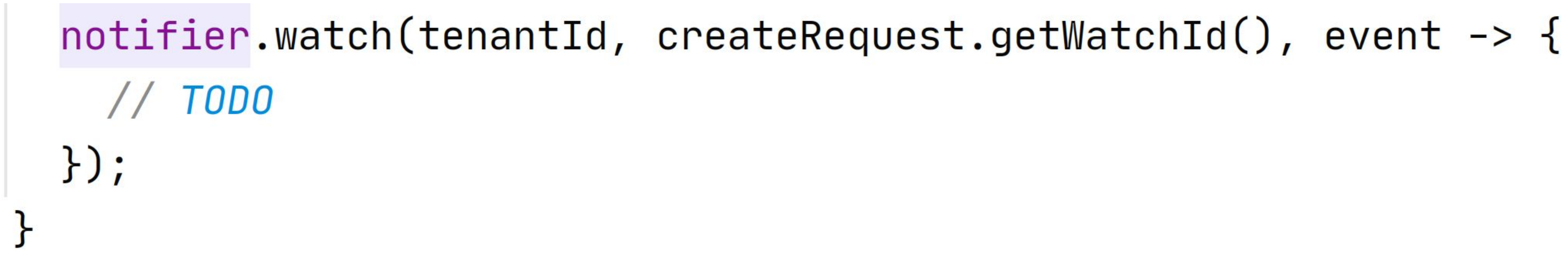
Publishing messages

Publishing a message is simple. Just use `publish` specifying the address to publish it to.

```
eventBus.publish("news.uk.sport", "Yay! Someone kicked a ball");
```

How to handle Watch?

1. Store Watch as another RecordType
2. Start listening to the event-bus



```
  notifier.watch(tenantId, createRequest.getWatchId(), event -> {
    // TODO
  });
}
```

How to handle Watch?

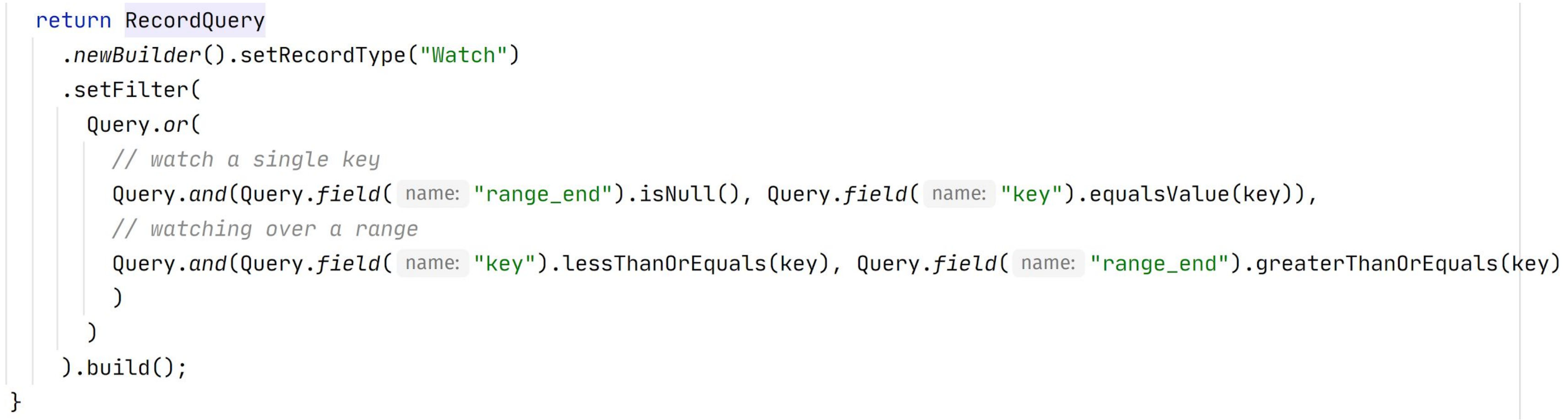
1. Store Watch as another RecordType
2. Start listening to the event-bus
3. During Put, check if a watch exists

```java
// checking if we have a Watch underneath
RecordQuery watchQuery = createWatchQuery(fixedRecord.getKey().toByteArray());
```

FoundationDB

How to handle Watch?

1. Store Watch as another RecordType
2. Start listening to the event-bus
3. During Put, check if a watch exists

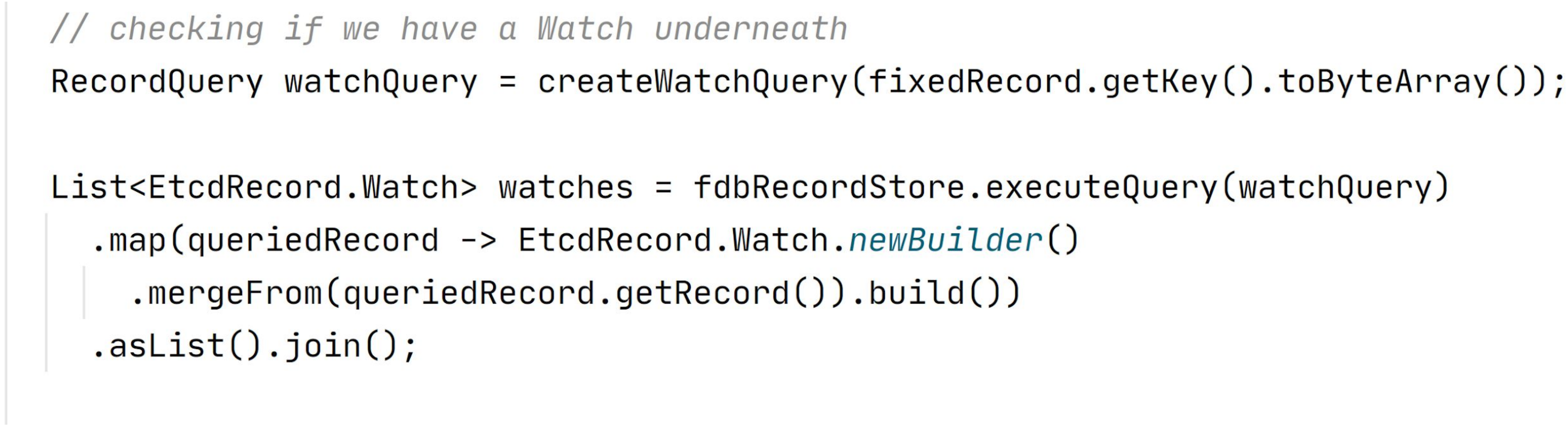


```
  return RecordQuery
    .newBuilder().setRecordType("Watch")
    .setFilter(
      Query.or(
        // watch a single key
        Query.and(Query.field( name: "range_end").isNull(), Query.field( name: "key").equalsValue(key)),
        // watching over a range
        Query.and(Query.field( name: "key").lessThanOrEquals(key), Query.field( name: "range_end").greaterThanOrEquals(key)
        )
      )
    ).build();
}
```

FoundationDB

How to handle Watch?

1. Store Watch as another RecordType
2. Start listening to the event-bus
3. During Put, check if a watch exists



```java
// checking if we have a Watch underneath
RecordQuery watchQuery = createWatchQuery(fixedRecord.getKey().toByteArray());

List<EtcdRecord.Watch> watches = fdbRecordStore.executeQuery(watchQuery)
  .map(queriedRecord -> EtcdRecord.Watch.newBuilder()
    .mergeFrom(queriedRecord.getRecord()).build())
  .asList().join();
```

FoundationDB

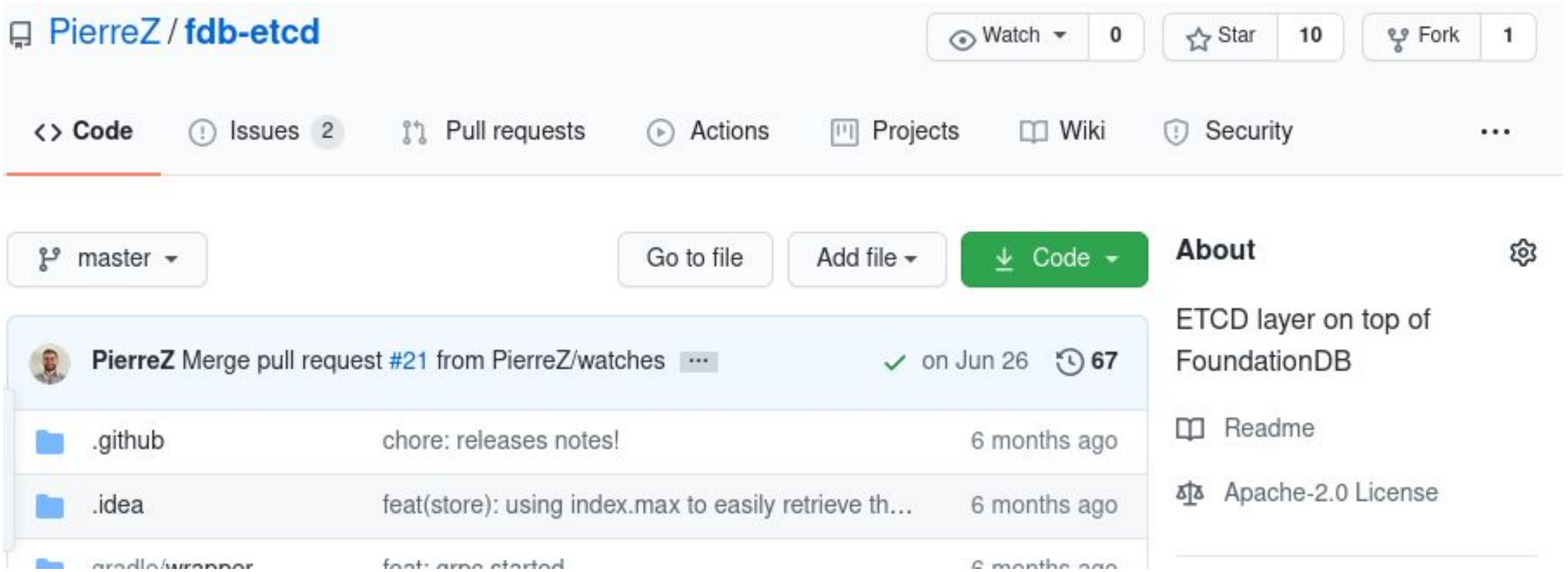
How to handle Watch?

1. Store Watch as another RecordType
2. Start listening to the event-bus
3. During Put, check if a watch exists
4. For each watch, send a message to the event-bus

```
if (notifier != null) {
  notifier.publish(tenantID, w.getWatchId(), event);
}
```

How to handle Watch

```
/**
 * Notifier are used to pass watched keys between KVService and WatchService
 * Standalone version can use the Vertx Event bus implementation,
 * but we should add Kafka/Pulsar implementation for distributed mode
 */
public interface Notifier {
  void publish(String tenant, long watchID, EtcdIoKvProto.Event event);
  void watch(String tenant, long watchID, Handler<EtcdIoKvProto.Event> handler);
}
```



PierreZ / fdb-etcd

PierreZ Merge pull request #21 from PierreZ/watches ... ✓ on Jun 26 67

.github	chore: releases notes!	6 months ago
.idea	feat(store): using index.max to easily retrieve th...	6 months ago

About

ETCD layer on top of FoundationDB

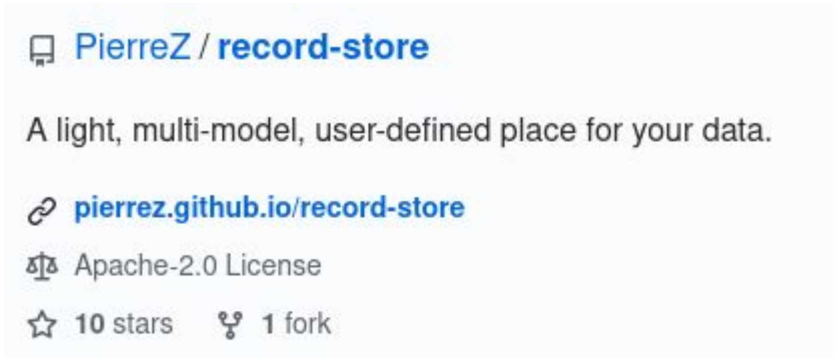
Readme

Apache-2.0 License

Now what?

- Playing with ETCD and Record layer was fun
- BUT I feel like I was under-using the Record Layer

What would a **gRPC abstraction** of the Record-Layer look like?


PierreZ / **record-store**

A light, multi-model, user-defined place for your data.

pierrez.github.io/record-store

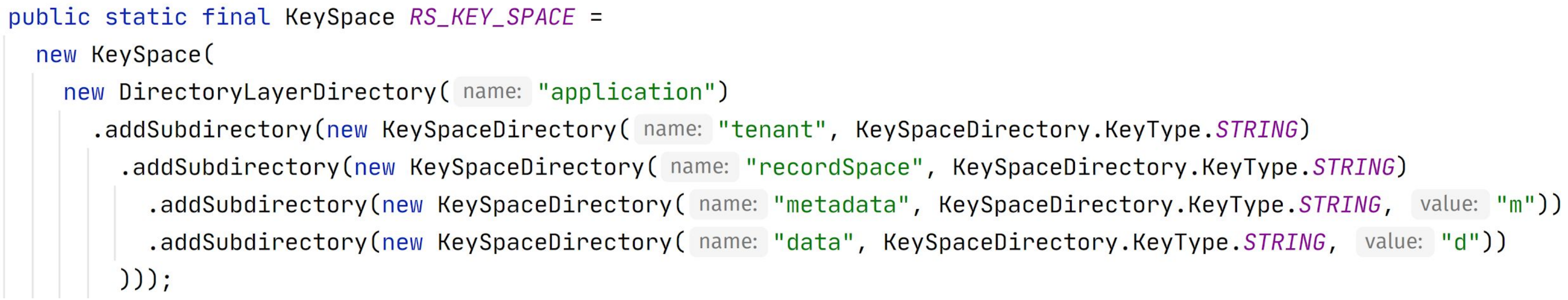
Apache-2.0 License

10 stars **1** fork

Record-Store workflow

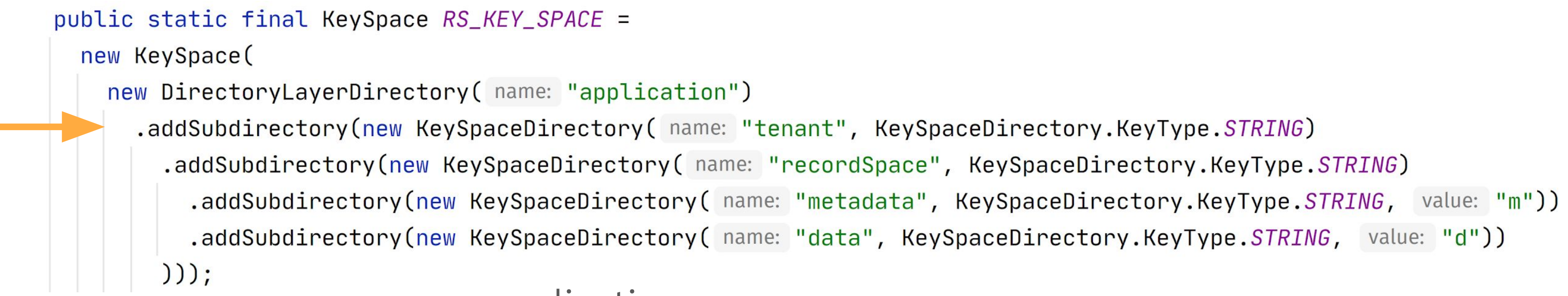
1. Opening RecordSpace, for example *“prod/users”*
2. Create a protobuf definition which will be used as schema
3. Upsert schema
4. Push records
5. Query records
6. delete records

Record-Store features: multi-tenancy



```
public static final KeySpace RS_KEY_SPACE =
    new KeySpace(
        new DirectoryLayerDirectory( name: "application")
            .addSubdirectory(new KeySpaceDirectory( name: "tenant", KeySpaceDirectory.KeyType.STRING)
                .addSubdirectory(new KeySpaceDirectory( name: "recordSpace", KeySpaceDirectory.KeyType.STRING)
                    .addSubdirectory(new KeySpaceDirectory( name: "metadata", KeySpaceDirectory.KeyType.STRING,  value: "m"))
                    .addSubdirectory(new KeySpaceDirectory( name: "data", KeySpaceDirectory.KeyType.STRING,  value: "d"))
                )));
```

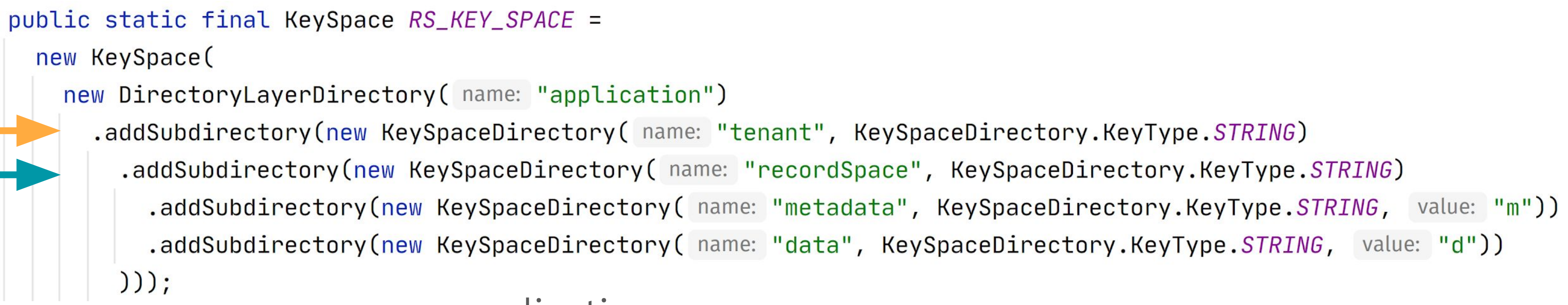
Record-Store features: multi-tenancy



```
public static final KeySpace RS_KEY_SPACE =
  new KeySpace(
    new DirectoryLayerDirectory( name: "application")
      .addSubdirectory(new KeySpaceDirectory( name: "tenant", KeySpaceDirectory.KeyType.STRING)
        .addSubdirectory(new KeySpaceDirectory( name: "recordSpace", KeySpaceDirectory.KeyType.STRING)
          .addSubdirectory(new KeySpaceDirectory( name: "metadata", KeySpaceDirectory.KeyType.STRING, value: "m"))
          .addSubdirectory(new KeySpaceDirectory( name: "data", KeySpaceDirectory.KeyType.STRING, value: "d"))
        )));
```

```
application
 └── record-store
      └── tenant
           └── my-tenant
```

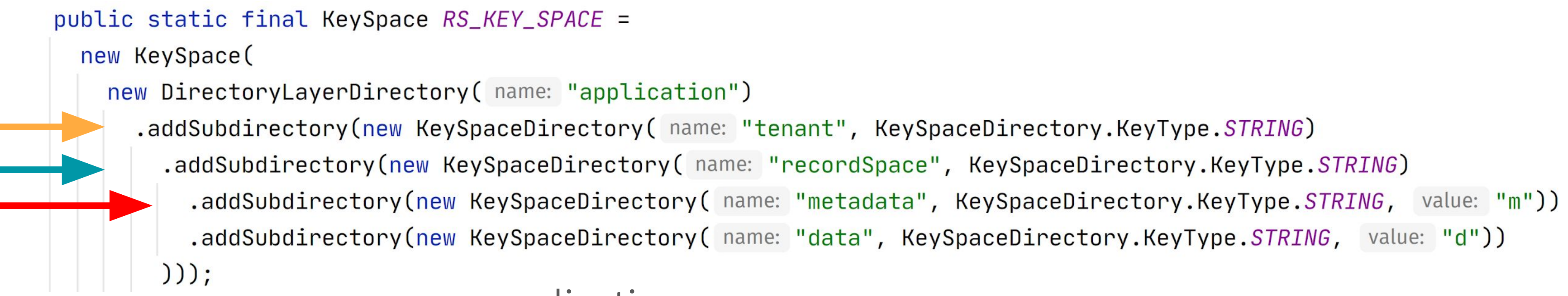
Record-Store features: multi-tenancy



```
public static final KeySpace RS_KEY_SPACE =
  new KeySpace(
    new DirectoryLayerDirectory( name: "application")
      .addSubdirectory(new KeySpaceDirectory( name: "tenant", KeySpaceDirectory.KeyType.STRING)
        .addSubdirectory(new KeySpaceDirectory( name: "recordSpace", KeySpaceDirectory.KeyType.STRING)
          .addSubdirectory(new KeySpaceDirectory( name: "metadata", KeySpaceDirectory.KeyType.STRING,  value: "m"))
          .addSubdirectory(new KeySpaceDirectory( name: "data", KeySpaceDirectory.KeyType.STRING,  value: "d"))
        )));
```

```
application
└── record-store
    └── tenant
        └── my-tenant
            └── recordSpace
                └── dev/users
```

Record-Store features: multi-tenancy

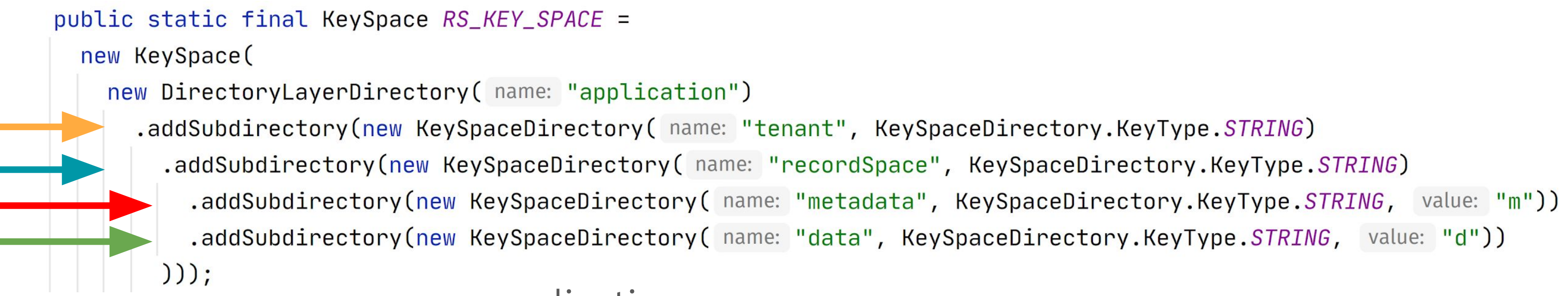


```
public static final KeySpace RS_KEY_SPACE =
  new KeySpace(
    new DirectoryLayerDirectory( name: "application")
      .addSubdirectory(new KeySpaceDirectory( name: "tenant", KeySpaceDirectory.KeyType.STRING)
        .addSubdirectory(new KeySpaceDirectory( name: "recordSpace", KeySpaceDirectory.KeyType.STRING)
          .addSubdirectory(new KeySpaceDirectory( name: "metadata", KeySpaceDirectory.KeyType.STRING,  value: "m"))
          .addSubdirectory(new KeySpaceDirectory( name: "data", KeySpaceDirectory.KeyType.STRING,  value: "d"))
        )));
```

```
application
└── record-store
    └── tenant
        └── my-tenant
            └── recordSpace
                └── dev/users
                    ├── data
```

FoundationDB

Record-Store features: multi-tenancy



```
public static final KeySpace RS_KEY_SPACE =
  new KeySpace(
    new DirectoryLayerDirectory( name: "application")
      .addSubdirectory(new KeySpaceDirectory( name: "tenant", KeySpaceDirectory.KeyType.STRING)
        .addSubdirectory(new KeySpaceDirectory( name: "recordSpace", KeySpaceDirectory.KeyType.STRING)
          .addSubdirectory(new KeySpaceDirectory( name: "metadata", KeySpaceDirectory.KeyType.STRING,  value: "m"))
          .addSubdirectory(new KeySpaceDirectory( name: "data", KeySpaceDirectory.KeyType.STRING,  value: "d"))
        )));
```

```
application
└── record-store
    └── tenant
        └── my-tenant
            └── recordSpace
                └── dev/users
                    ├── data
                    └── meta
```

Record-Store features: Schema upgrades

```
// administrative part of the recordSpace
service SchemaService {
  // Upsert a schema
  rpc Upsert (UpsertSchemaRequest) returns (EmptyResponse);
  // Retrieve the stats for a container
  rpc Stat (StatRequest) returns (StatResponse);
  // Get the current schema for a container
  rpc Get (GetSchemaRequest) returns (GetSchemaResponse);
}
```

How to send a Protobuf schema?

```
syntax = "proto3";

import "google/protobuf/any.proto";
import "google/protobuf/descriptor.proto";

message SelfDescribingMessage {
  // Set of FileDescriptorProtos which describe the type and its dependencies.
  google.protobuf.FileDescriptorSet descriptor_set = 1;

  // The message and its type, encoded as an Any message.
  google.protobuf.Any message = 2;
}
```

How to upgrade schema?

1. Retrieve old schemas and indexes
2. Create new metadata
3. Add old schema
4. Add new FileDescriptors
5. Add old indexes
6. Add new indexes if any
7. metadata.build() // build and validate metadata

split records

```
metadataBuilder.setSplitLongRecords(true);
```

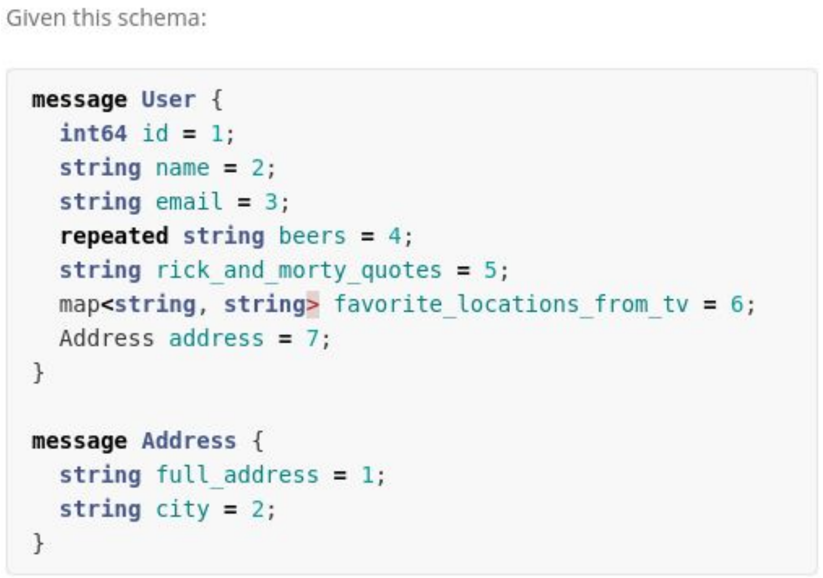
Manipulating records

```
// Service which manipulate records within a recordSpace
service RecordService {
  // put a record
  rpc Put (PutRecordRequest) returns (EmptyResponse);
  // query records
  rpc Query (QueryRequest) returns (stream QueryResponse);
  // return the queryPlan for a query
  rpc GetQueryPlan (QueryRequest) returns (GetQueryPlanResponse);
  // Delete records
  rpc Delete (DeleteRecordRequest) returns (DeleteRecordResponse);
}
```

Data encryption

```
  private FDBRecordStore createFDBRecordStore(FDBRecordContext context,
                                              FDBMetaDataStore metaDataStore,
                                              SecretKey key, String tenantID, String container) {

    TransformedRecordSerializer<Message> serializer = TransformedRecordSerializerJCE.newDefaultBuilder()
      .setEncryptWhenSerializing(true)
      .setCompressWhenSerializing(true)
      .setEncryptionKey(key)
      .build();

    return FDBRecordStore.newBuilder()
      .setMetaDataProvider(metaDataStore)
      .setContext(context)
      .setSerializer(serializer)
      .setKeySpacePath(RecordStoreKeySpace.getDataKeySpacePath(tenantID, container))
      .createOrOpen();
  }
}
```

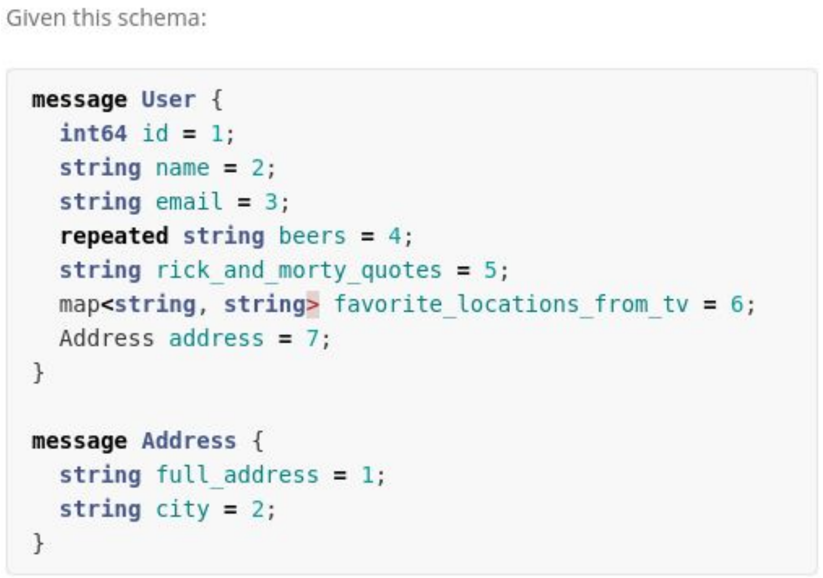
Query capabilities



Given this schema:

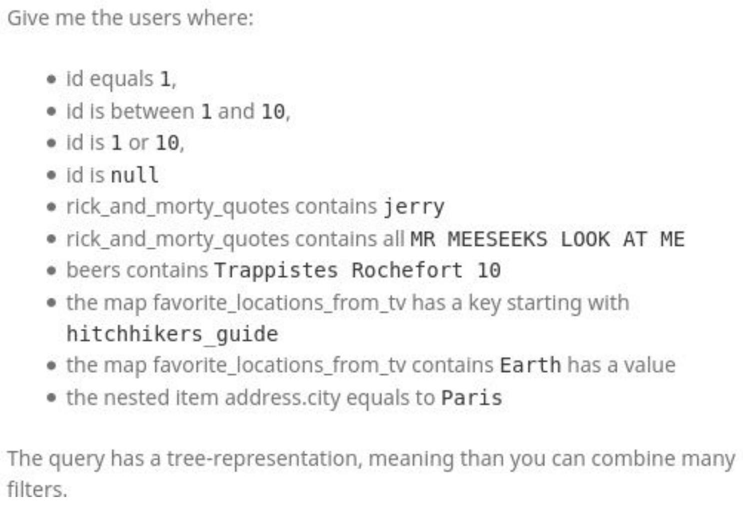
```
message User {
  int64 id = 1;
  string name = 2;
  string email = 3;
  repeated string beers = 4;
  string rick_and_morty_quotes = 5;
  map<string, string> favorite_locations_from_tv = 6;
  Address address = 7;
}

message Address {
  string full_address = 1;
  string city = 2;
}
```

Query capabilities



Given this schema:

```
message User {
  int64 id = 1;
  string name = 2;
  string email = 3;
  repeated string beers = 4;
  string rick_and_morty_quotes = 5;
  map<string, string> favorite_locations_from_tv = 6;
  Address address = 7;
}

message Address {
  string full_address = 1;
  string city = 2;
}
```



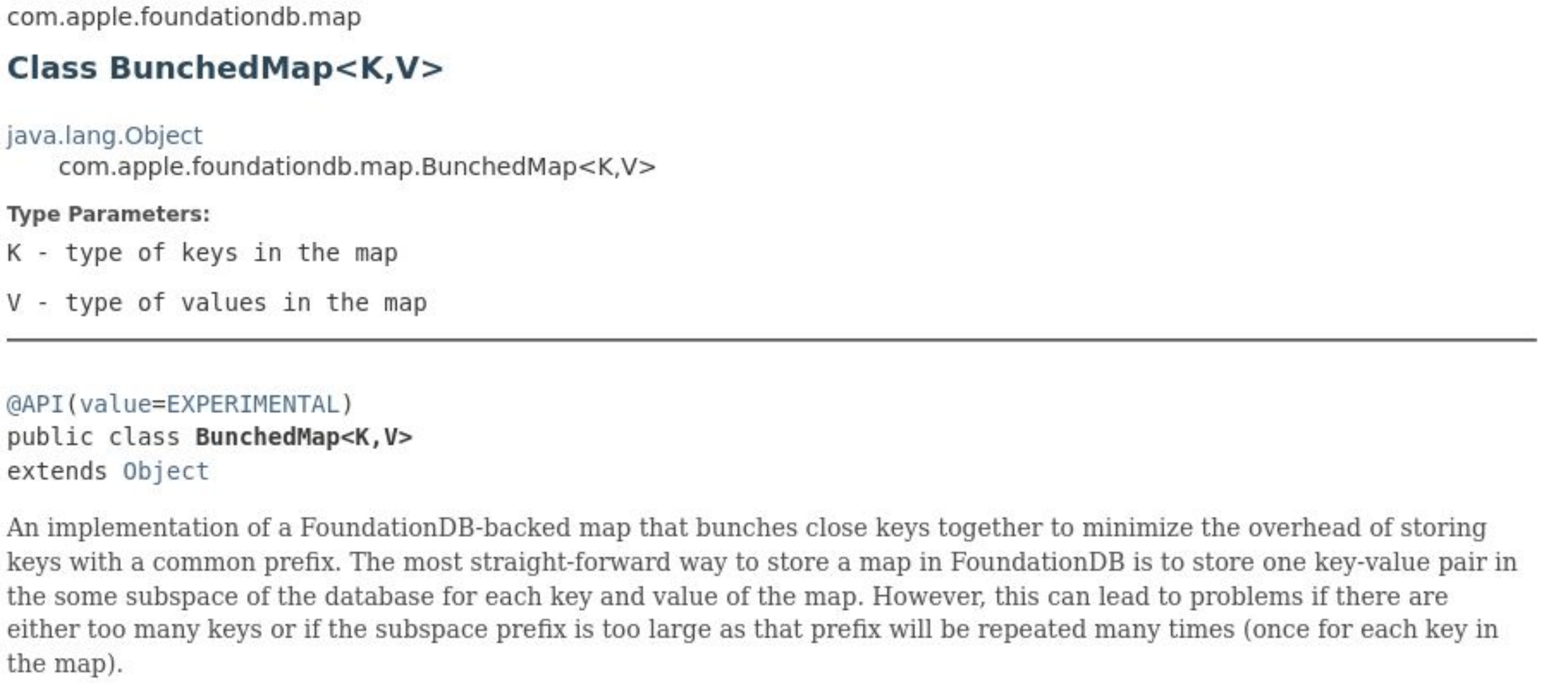
Give me the users where:

- id equals `1`,
- id is between `1` and `10`,
- id is `1` or `10`,
- id is `null`
- rick_and_morty_quotes contains `jerry`
- rick_and_morty_quotes contains all `MR MEESEEKS LOOK AT ME`
- beers contains `Trappistes Rochefort 10`
- the map favorite_locations_from_tv has a key starting with `hitchhikers_guide`
- the map favorite_locations_from_tv contains `Earth` has a value
- the nested item address.city equals to `Paris`

The query has a tree-representation, meaning than you can combine many filters.

FoundationDB

Things I want to try



com.apple.foundationdb.map

Class BunchedMap<K,V>

java.lang.Object
    com.apple.foundationdb.map.BunchedMap<K,V>

Type Parameters:

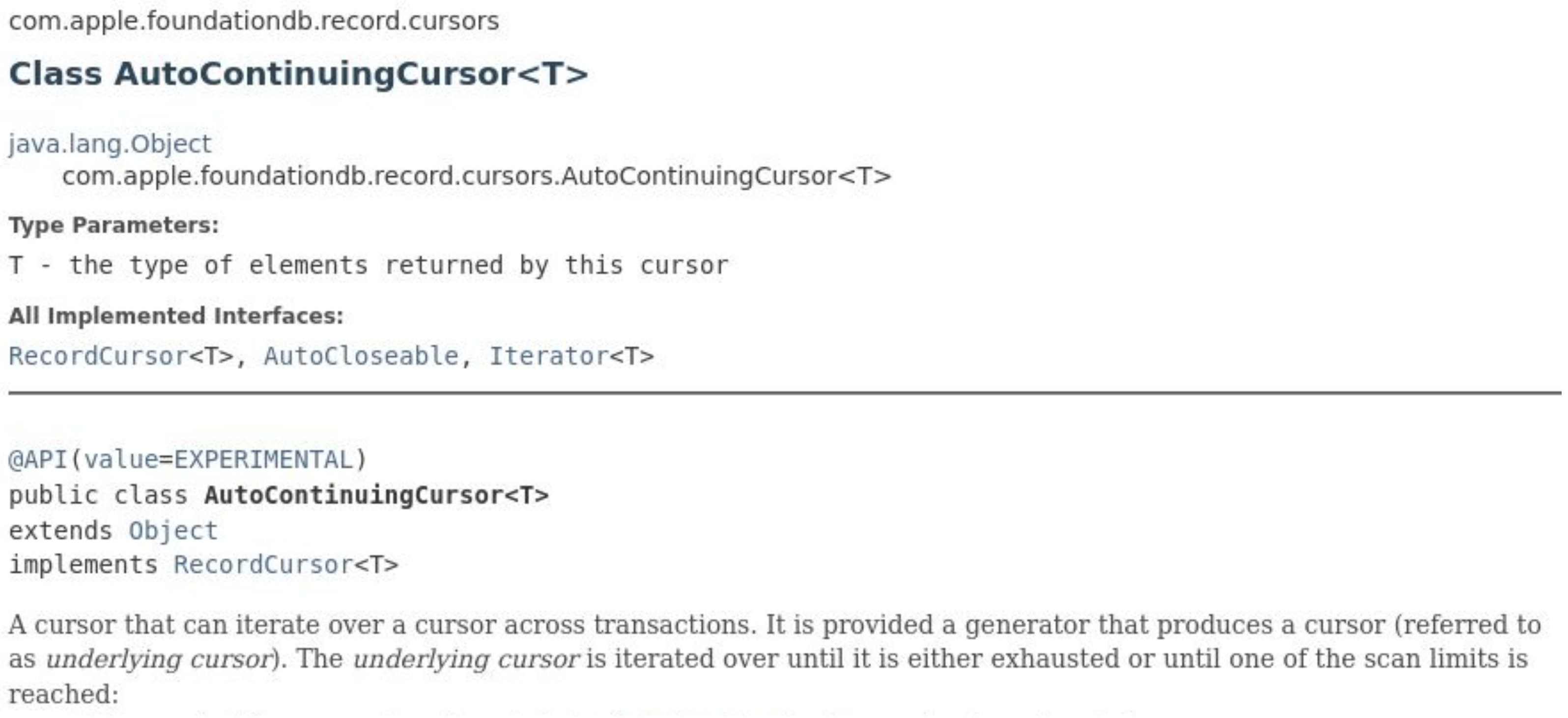
`K - type of keys in the map`

`V - type of values in the map`

```
@API(value=EXPERIMENTAL)
public class BunchedMap<K,V>
extends Object
```

An implementation of a FoundationDB-backed map that bunches close keys together to minimize the overhead of storing keys with a common prefix. The most straight-forward way to store a map in FoundationDB is to store one key-value pair in the some subspace of the database for each key and value of the map. However, this can lead to problems if there are either too many keys or if the subspace prefix is too large as that prefix will be repeated many times (once for each key in the map).

Things I want to try



com.apple.foundationdb.record.cursors

Class AutoContinuingCursor<T>

java.lang.Object
    com.apple.foundationdb.record.cursors.AutoContinuingCursor<T>

Type Parameters:

`T - the type of elements returned by this cursor`

All Implemented Interfaces:

`RecordCursor<T>, AutoCloseable, Iterator<T>`

```
@API(value=EXPERIMENTAL)
public class AutoContinuingCursor<T>
extends Object
implements RecordCursor<T>
```

A cursor that can iterate over a cursor across transactions. It is provided a generator that produces a cursor (referred to as *underlying cursor*). The *underlying cursor* is iterated over until it is either exhausted or until one of the scan limits is reached:

Things I want to try

```
/**
 * A ranked set index, allowing efficient rank and select operations.
 */
public static final String RANK = "rank";

/**
 * An index with sliding time window leaderboards.
 */
public static final String TIME_WINDOW_LEADERBOARD = "time_window_leaderboard";
```

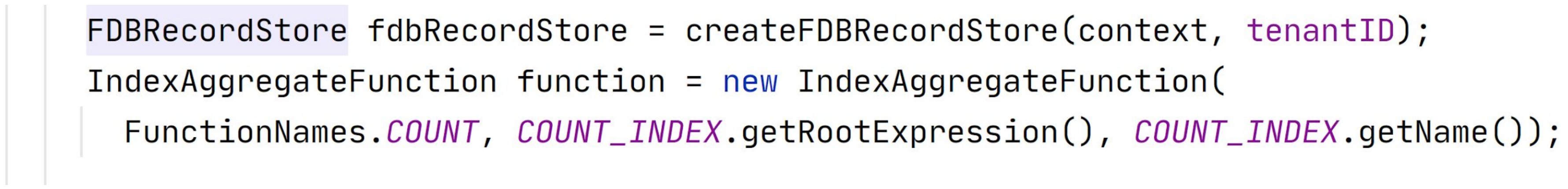
Thanks!
Do you have questions?

Slides	**https://pierrezemb.fr**
Twitter	**PierreZ**
Github	**PierreZ**

Bonus 🎓

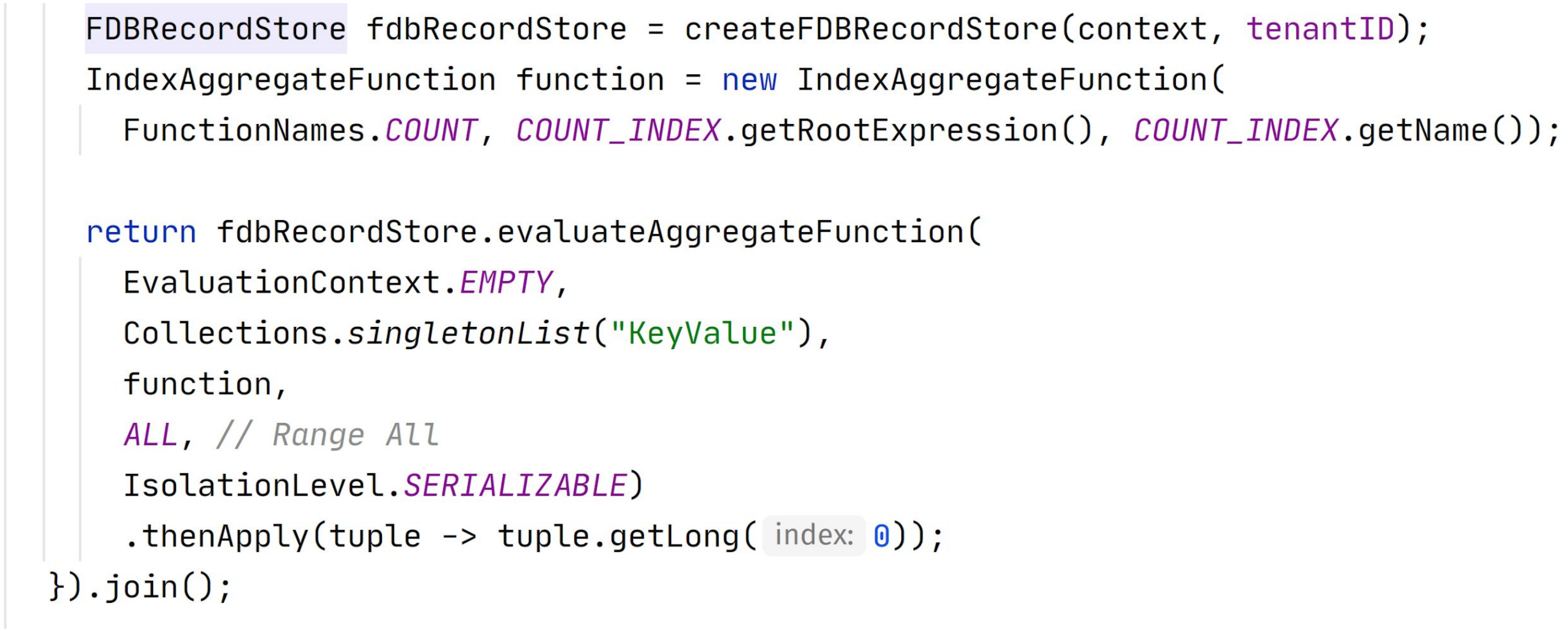
How to keep track of the number of records?

Index can be scanned and evaluated



```
FDBRecordStore fdbRecordStore = createFDBRecordStore(context, tenantID);
IndexAggregateFunction function = new IndexAggregateFunction(
  FunctionNames.COUNT, COUNT_INDEX.getRootExpression(), COUNT_INDEX.getName());
```

Index can be scanned and evaluated



```java
  FDBRecordStore fdbRecordStore = createFDBRecordStore(context, tenantID);
  IndexAggregateFunction function = new IndexAggregateFunction(
    FunctionNames.COUNT, COUNT_INDEX.getRootExpression(), COUNT_INDEX.getName());

  return fdbRecordStore.evaluateAggregateFunction(
    EvaluationContext.EMPTY,
    Collections.singletonList("KeyValue"),
    function,
    ALL, // Range All
    IsolationLevel.SERIALIZABLE)
    .thenApply(tuple -> tuple.getLong( index: 0));
}).join();
```

FoundationDB