или

Что делать, когда “интернет стабильный”

Зураб Белый
Рексофт 2017

Дисклеймер

Не верьте мне.

Проверяйте всё сами.

Знакомство с PouchDB

Что это?

Что это?

NoSQL база данных в браузере

Что это?

NoSQL база данных в браузере

Автоматическая синхронизация

Что это?

NoSQL база данных в браузере

Автоматическая синхронизация

Использует IndexedDB и WebSQL

Что это?

NoSQL база данных в браузере

Автоматическая синхронизация

Использует IndexedDB и WebSQL

Free open-source

Что это?

NoSQL база данных в браузере

Автоматическая синхронизация

Использует IndexedDB и WebSQL

Free open-source

Интеграция с CouchDB

Где работает?

Где работает?

Firefox 29+ (Including Firefox OS and Firefox for Android)

Chrome 30+

Safari 5+

Internet Explorer 10+

Opera 21+

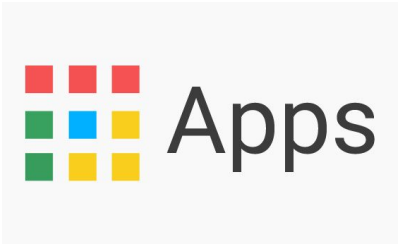
Android 4.0+

iOS 7.1+

Windows Phone 8+

Где работает?

Firefox 29+
Chrome 30+
Safari 5+
Internet Explorer 10+
Opera 21+
Android 4.0+
iOS 7.1+
Windows Phone 8+

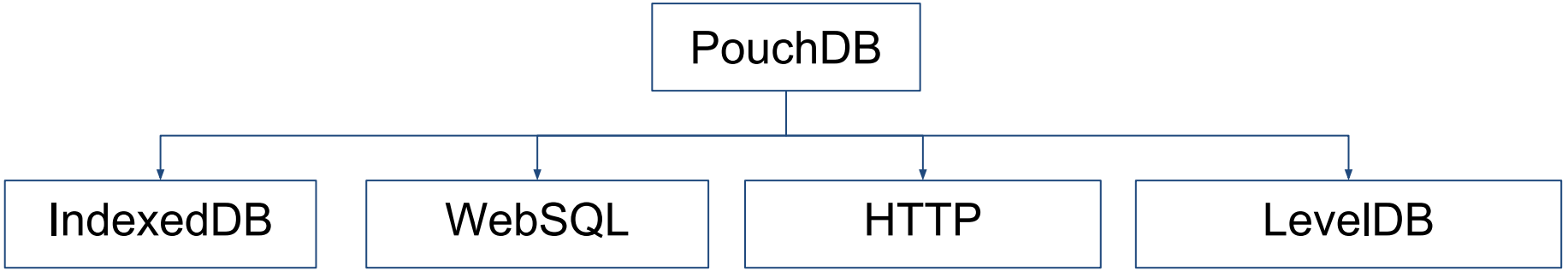


Как работает?

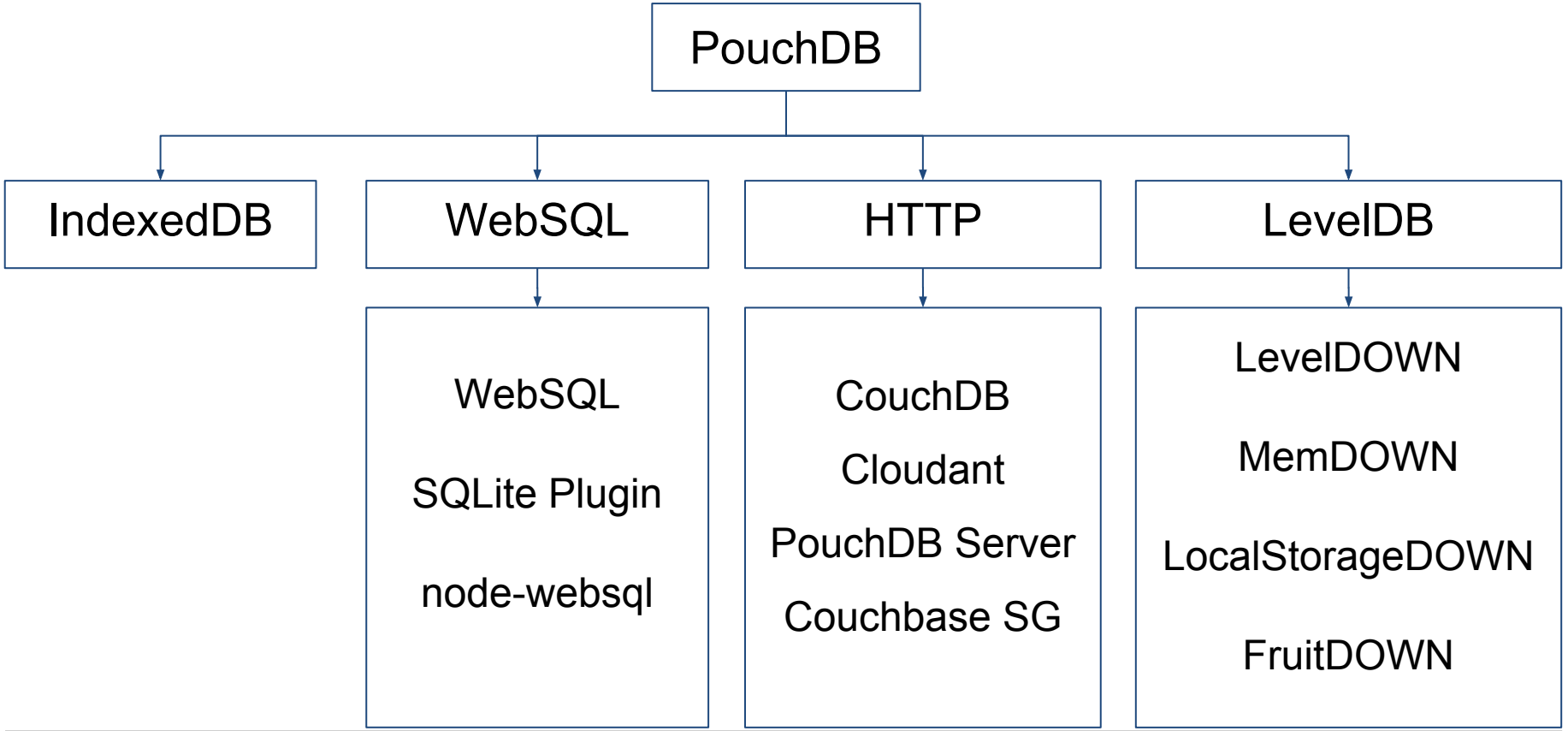
Как работает?

PouchDB

Как работает?

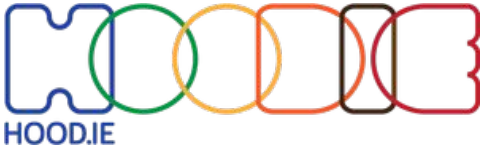


Как работает?



Кто использует?

Кто использует?



Базовый API

```
1.  const db = new PouchDB('databaseName');
```

```
1.  const db = new PouchDB('databaseName');
```

SQL concept	PouchDB concept
table	*no equivalent*
row	document
column	field
primary key	primary key (_id)
index	view

```
1.  const localDB = new PouchDB('databaseName');
```

```
1.  const localDB = new PouchDB('databaseName');

2.  const remoteDbUrl = 'http://domain:5984/dbName';
```

```
1.  const localDB = new PouchDB('databaseName');

2.  const remoteDbUrl = 'http://domain:5984/dbName';

3.  localDB.sync(remoteDbUrl, {live: true, retry: true});
```

```
1.  db.get(data)
2.     .then(doc => { /* do something with document */ })
3.     .catch(err => { /* errors handling */ });
```

```
1.  db.get(data)
2.     .then(doc => { /* do something with document */ })
3.     .catch(err => { /* errors handling */ });


4.  db.put(data)
5.     .then(response => { /* handling result */ })
6.     .catch(err => { /* errors handling */ });
```

```
1.  db.get(data)
2.     .then(doc => { /* do something with document */ })
3.     .catch(err => { /* errors handling */ });


4.  db.put(data)
5.     .then(response => { /* handling result */ })
6.     .catch(err => { /* errors handling */ });


7.  db.remove(data)
8.     .then(response => { /* handling result */ })
9.     .catch(err => { /* errors handling */ });
```

```
{
 "_id": "test-doc-1",
 "_rev": "1-A6157A5EA545C99B00FF904EEF05FD9F",
 "content": "Some content"
}
```

```
1.  {
2.   "_id": "test-doc-1",
3.   "_rev": "1-A6157A5EA545C99B00FF904EEF05FD9F",
4.   "content": "Some content"
5.  }
```

- Вся история ревизий сохраняется в базе

```
1.  {
2.   "_id": "test-doc-1",
3.   "_rev": "1-A6157A5EA545C99B00FF904EEF05FD9F",
4.   "content": "Some content"
5.  }
```

- Вся история ревизий сохраняется в базе
- Номер ревизии генерируется автоматически

```
{
 "_id": "test-doc-1",
 "_rev": "1-A6157A5EA545C99B00FF904EEF05FD9F",
 "content": "Some content"
}
```

- Вся история ревизий сохраняется в базе
- Номер ревизии генерируется автоматически
- Для функций `put()`, `post()`, `remove()`, `bulkDocs()`, и `putAttachment()` указание ревизии обязательно

```
1.  {
2.   "_id": "test-doc-1",
3.   "_rev": "
4.   "content"
5.  }
```

- Вся истори
- Номер реви
- Для функц

`bulkDocs()`

ревизии об

Конфликты и их решения

Типы конфликтов

Типы конфликтов

1. Насильственные (антагонистичные) конфликты
2. Компромиссные конфликты
3. Политические конфликты
4. Социальные конфликты
5. Экономические конфликты
6. Организационные конфликты

Типы конфликтов

1. Насильственные (антагонистичные) конфликты
2. Компромиссные конфликты
3. Политические конфликты
4. Социальный конфликт
5. Экономические конфликты
6. Организационные конфликты

Типы конфликтов

1. Насильственные (антагонистичные) конфликты
2. Компромиссные конфликты
3. Политические конфликты
4. Социальный конфликт
5. Экономические конфликты
6. Организационные конфликты

1. Непосредственные конфликты (Immediate conflicts)

Типы конфликтов

1. Насильственные (антагонистичные) конфликты
2. Компромиссные конфликты
3. Политические конфликты
4. Социальный конфликт
5. Экономические конфликты
6. Организационные конфликты

1. Непосредственные конфликты (Immediate conflicts)
2. Возможные конфликты (Eventual conflicts)

Непосредственные конфликты

Непосредственные конфликты

- Оптимистическая блокировка

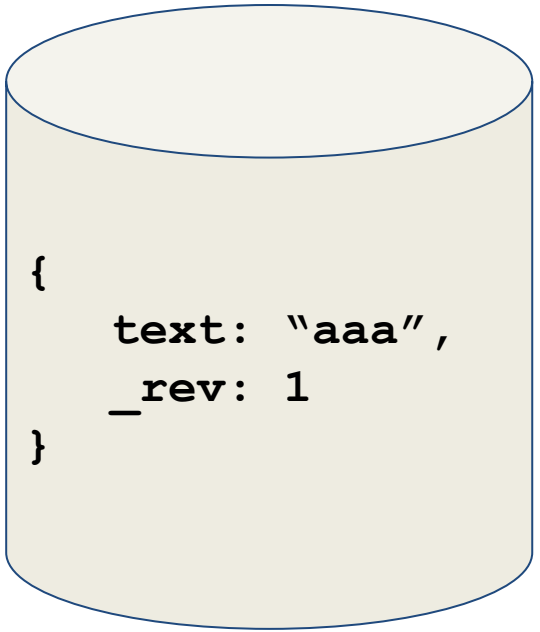
Непосредственные конфликты

- Оптимистическая блокировка

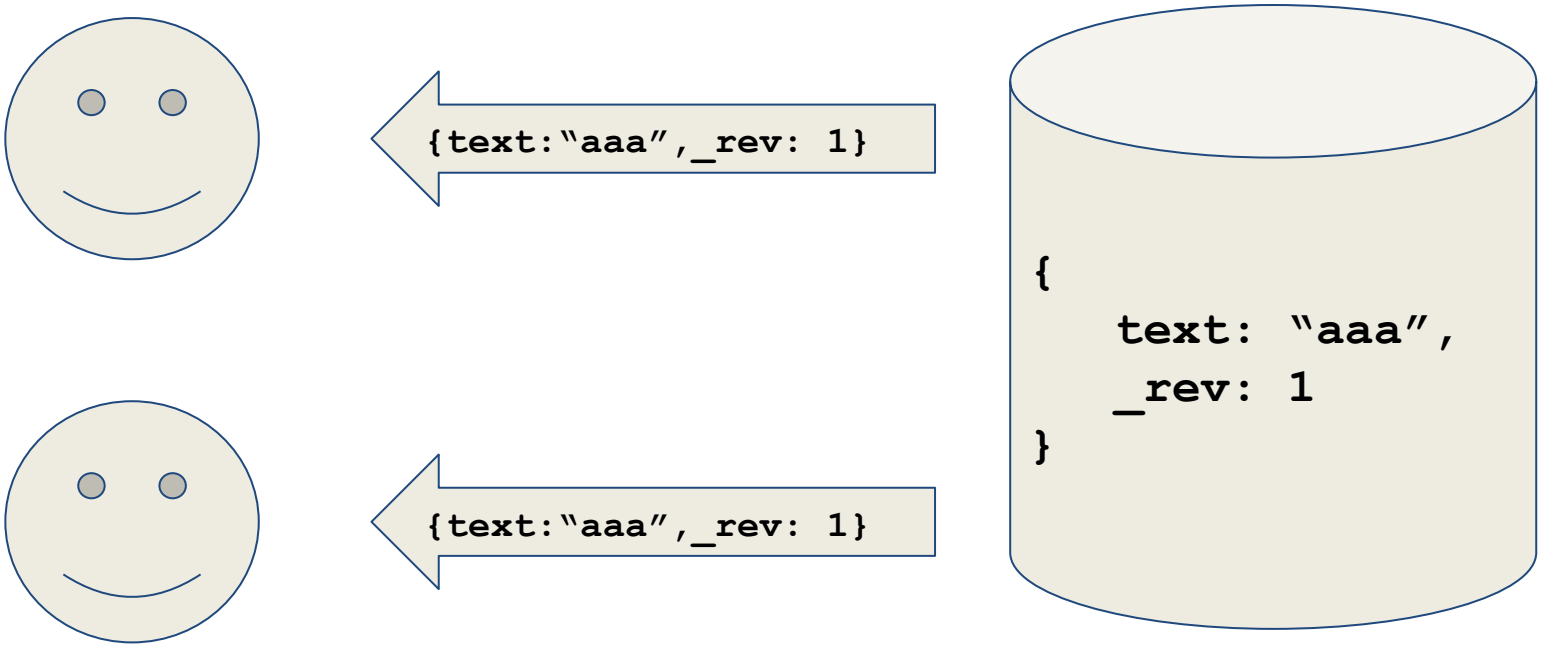
```
{
  error: true,
  status: 409,
  name: 'conflict',
  message: 'Document update conflict'
}
```

Оптимистическая блокировка

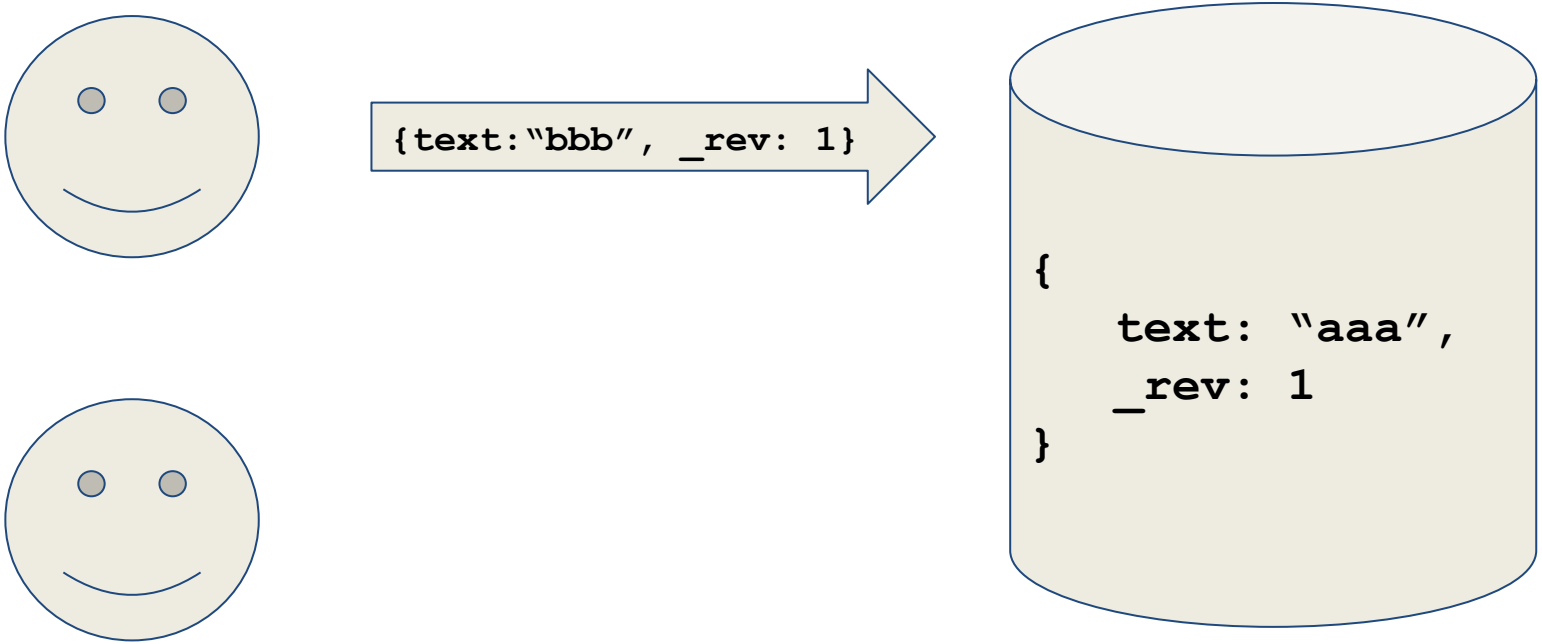
Оптимистическая блокировка



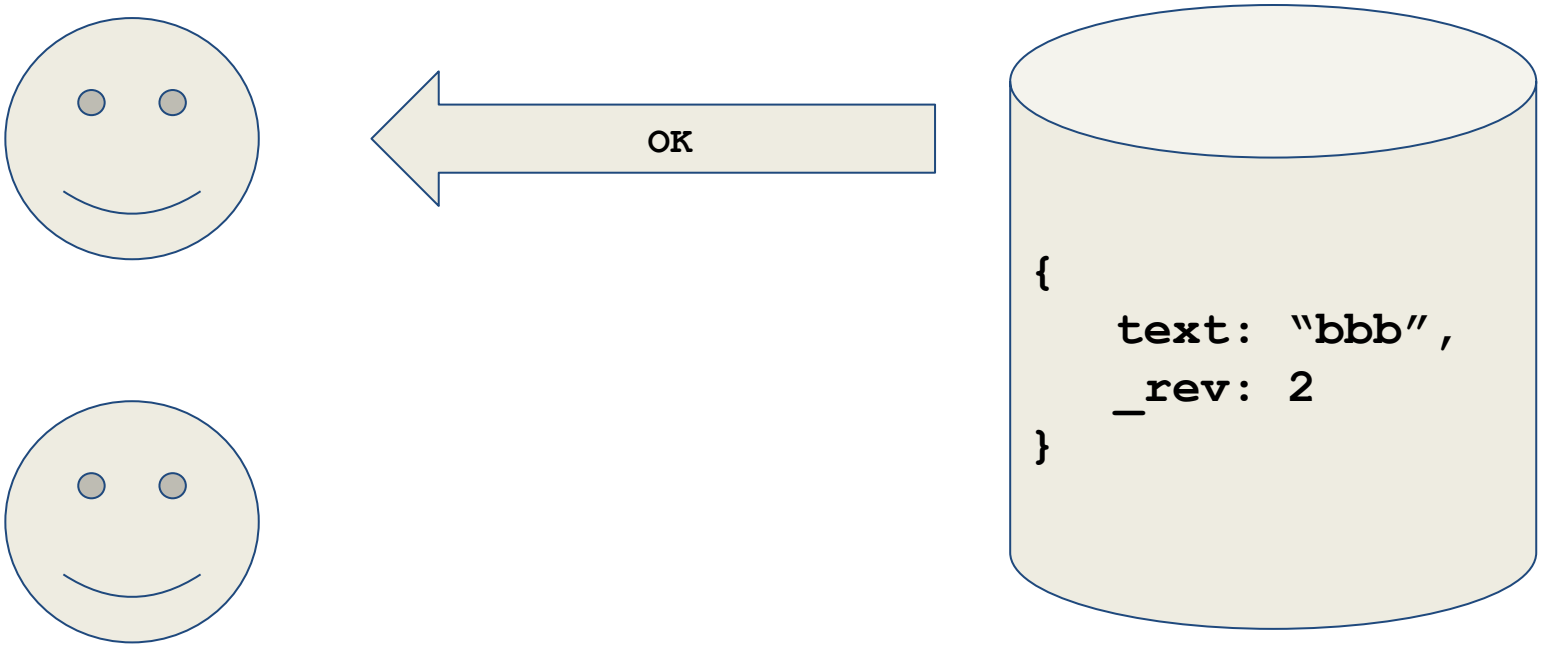
Оптимистическая блокировка



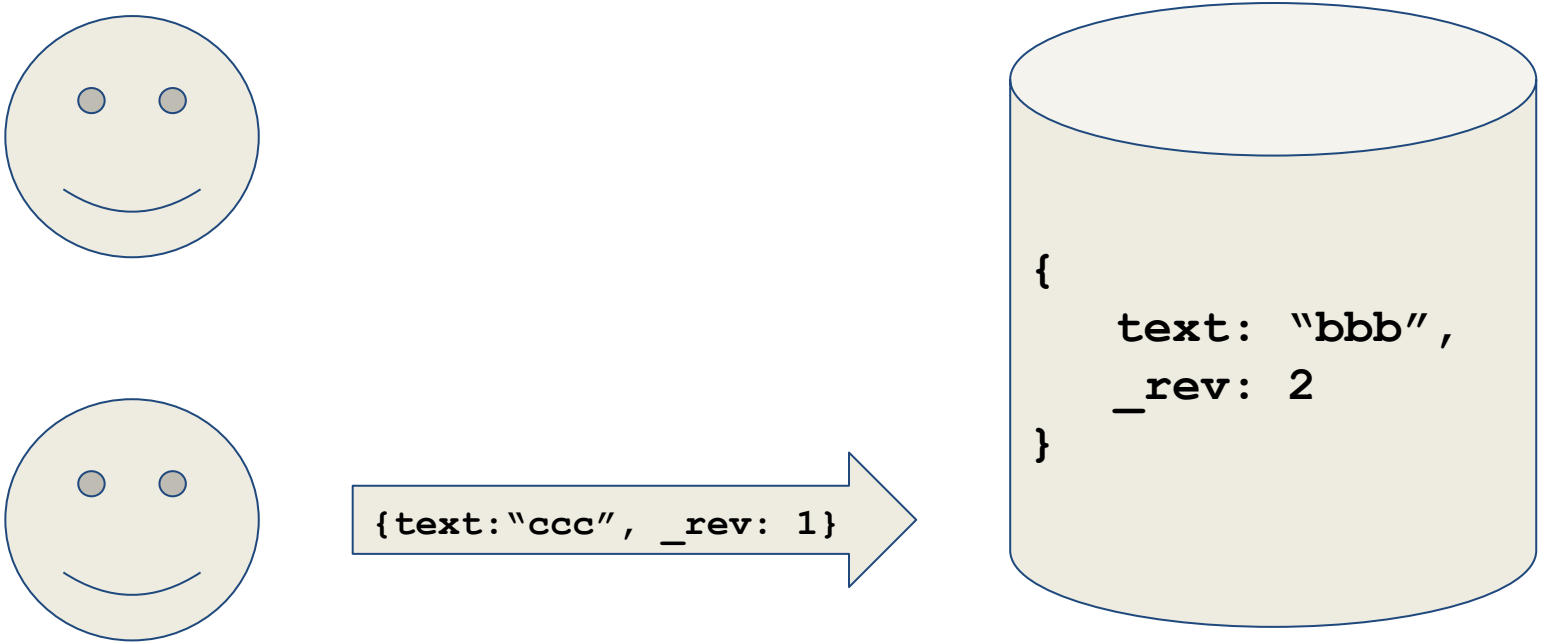
Оптимистическая блокировка



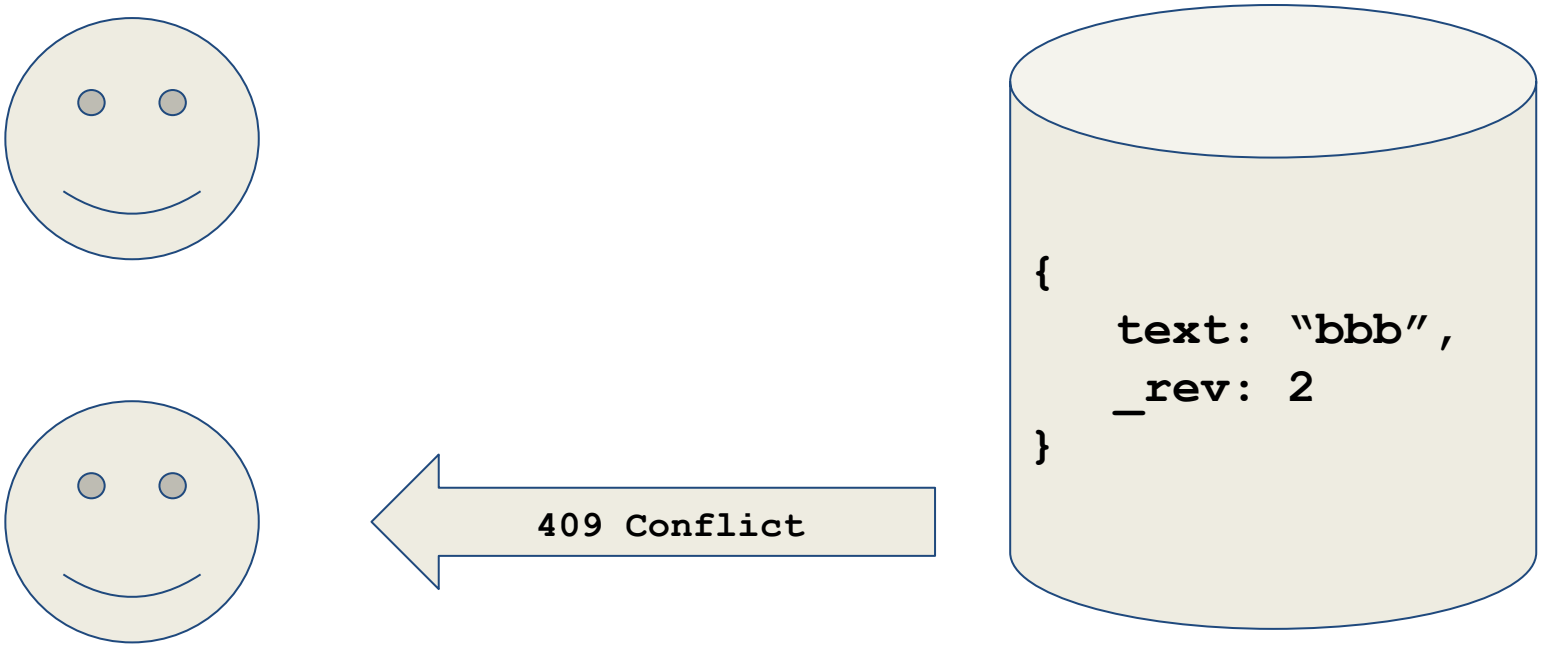
Оптимистическая блокировка



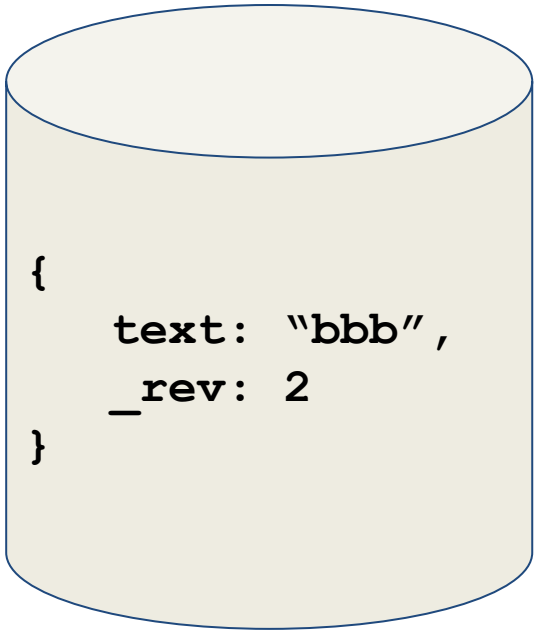
Оптимистическая блокировка



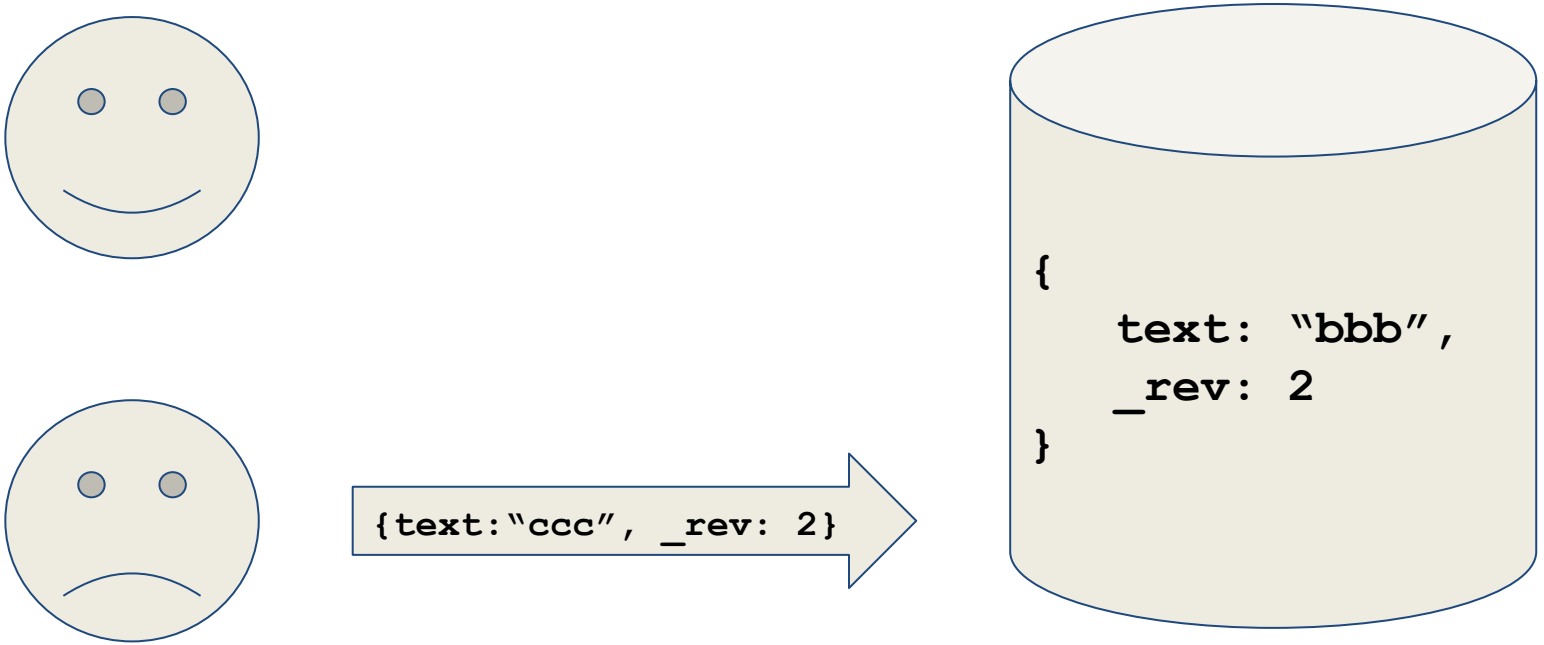
Оптимистическая блокировка



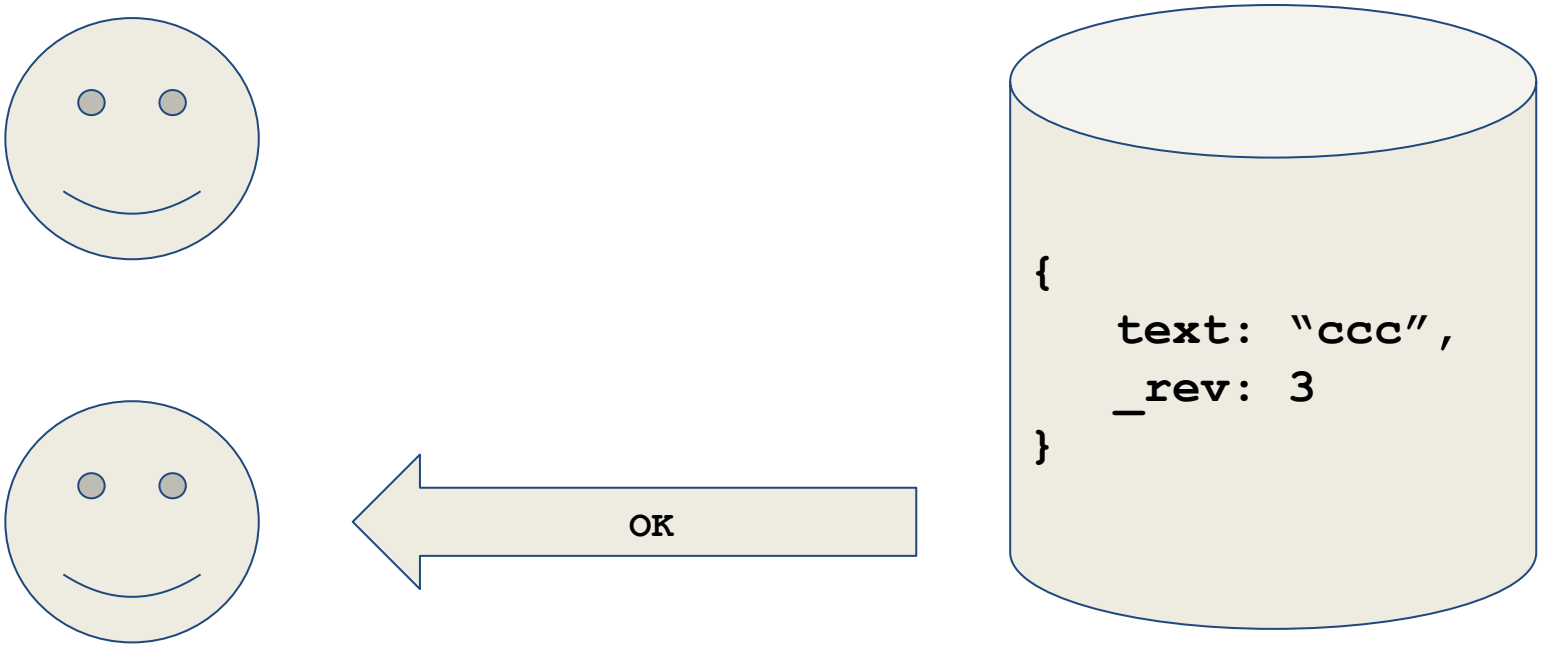
Оптимистическая блокировка



Оптимистическая блокировка

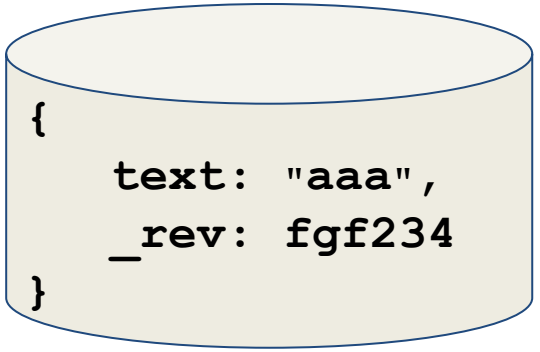


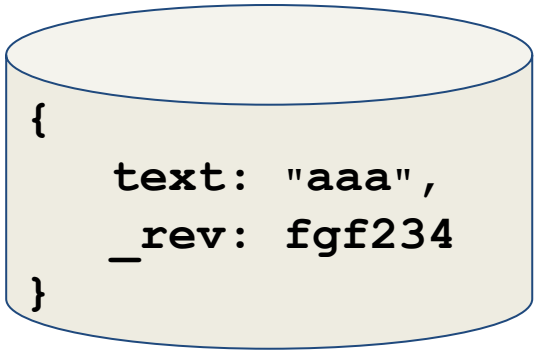
Оптимистическая блокировка



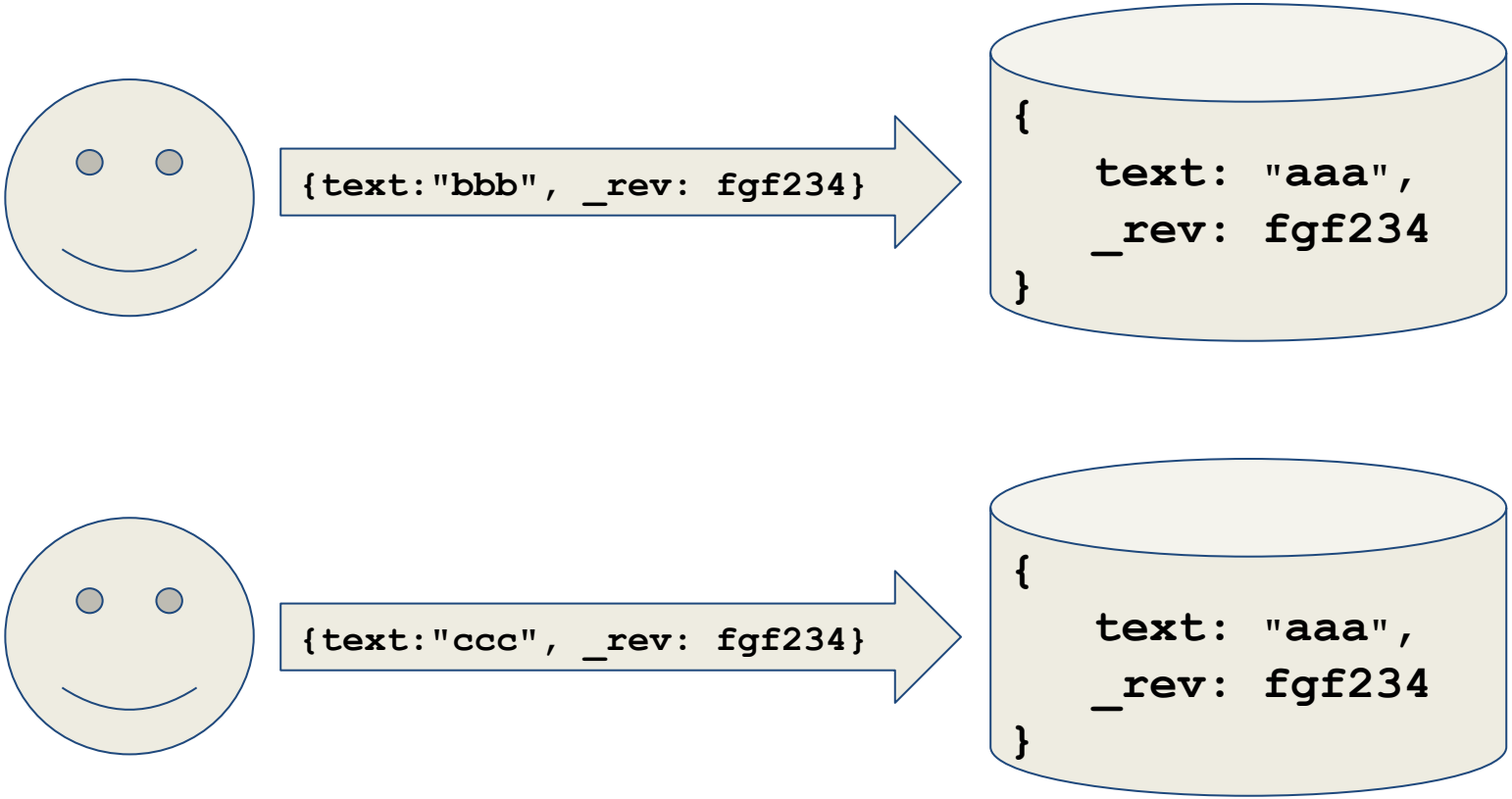
Возможные конфликты

Возможные конфликты

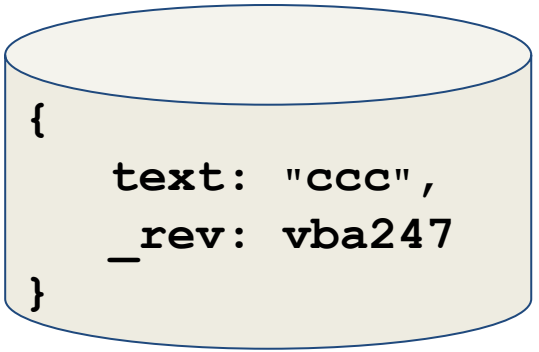




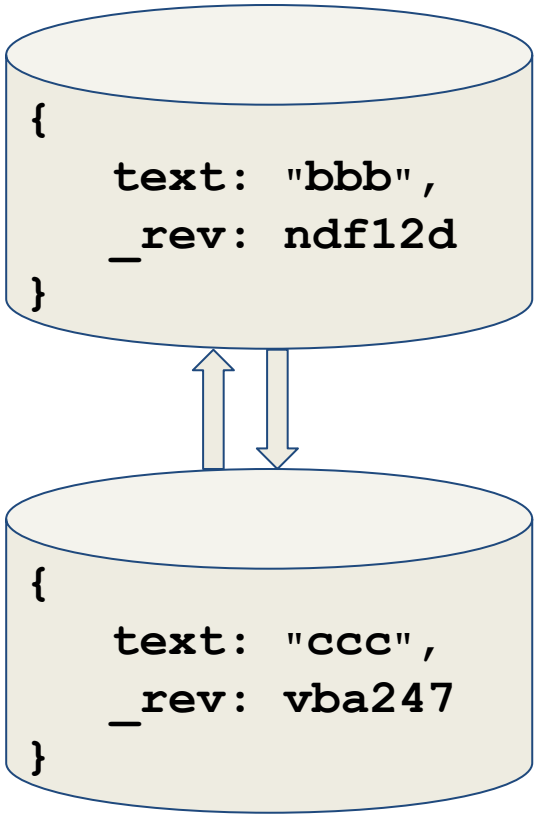
Возможные конфликты



Возможные конфликты



Возможные конфликты



Возможные конфликты

- Автоматическое решение

Возможные конфликты

- Автоматическое решение
- “handles it very elegantly”

Возможные конфликты

- Автоматическое решение
- “handles it very elegantly”
- Возможность решить вручную

Возможные конфликты

- Автоматическое решение
- “handles it very elegantly”
- Возможность решить вручную

```
1.  db.get(id, {conflicts: true})
```

Возможные конфликты

- Автоматическое решение
- “handles it very elegantly”
- Возможность решить вручную

```
1.  db.get(id, {conflicts: true})

2.  db.get(id, {rev: '8f9asd7s'})
```

Репликация

The way I like to think about CouchDB is this: CouchDB is bad at everything, ***except syncing****. And it turns out that's the most important feature you could ever ask for, for many types of software.*

Jason Smith

Apache CouchDB committer
Cloudant developer advocate

```
1.  const localDB = new PouchDB('databaseName');
2.  const remoteDB = new PouchDB('http://domain:5984/dbName');
```

```
1.  const localDB = new PouchDB('databaseName');
2.  const remoteDB = new PouchDB('http://domain:5984/dbName');


3.  localDB
4.    .replicate.to(remoteDB)
5.    .on('complete', () => {})
6.    .on('error', (err) => {});
```

```
1.  const localDB = new PouchDB('databaseName');
2.  const remoteDB = new PouchDB('http://domain:5984/dbName');
```

```
3.  localDB
4.    .replicate.to(remoteDB)
5.    .on('complete', () => {})
6.    .on('error', (err) => {});
```

```
7.  localDB
8.    .replicate.from(remoteDB)
9.    .on('complete', () => {})
10.   .on('error', (err) => {});
```

```
1.  const localDB = new PouchDB('databaseName');
2.  const remoteDB = new PouchDB('http://domain:5984/dbName');
```

```
3.  localDB
4.    .replicate.to(remoteDB)
5.    .on('complete', () => {})
6.    .on('error', (err) => {});
```

```
7.  localDB
8.    .replicate.from(remoteDB)
9.    .on('complete', () => {})
10.   .on('error', (err) => {});
```

```
11. localDB.sync(remoteDB);
```

```
1.  localDB
2.    .sync(remoteDB, { live: true, retry: true })
```

```
1.  localDB
2.    .sync(remoteDB, { live: true, retry: true })
3.    .on('change', (info) => {
4.          /* database content changed within replication */
5.    })
```

```
1.  localDB
2.    .sync(remoteDB, { live: true, retry: true })
3.    .on('change', (info) => {
4.          /* database content changed within replication */
5.    })
6.    .on('paused', (err) => { /* replication paused */ })
```

```
localDB
  .sync(remoteDB, { live: true, retry: true })
  .on('change', (info) => {
        /* database content changed within replication */
  })
  .on('paused', (err) => { /* replication paused */ })
  .on('active', () => { /* replication resumed */ })
```

```
localDB
  .sync(remoteDB, { live: true, retry: true })
  .on('change', (info) => {
       /* database content changed within replication */
  })
  .on('paused', (err) => { /* replication paused */ })
  .on('active', () => { /* replication resumed */ })
  .on('denied', (err) => {
       /* a document failed to replicate */
  })
```

```
localDB
  .sync(remoteDB, { live: true, retry: true })
  .on('change', (info) => {
       /* database content changed within replication */
  })
  .on('paused', (err) => { /* replication paused */ })
  .on('active', () => { /* replication resumed */ })
  .on('denied', (err) => {
       /* a document failed to replicate */
  })
  .on('complete', (info) => { /* replication complete*/ })
```

```
localDB
  .sync(remoteDB, { live: true, retry: true })
  .on('change', (info) => {
       /* database content changed within replication */
  })
  .on('paused', (err) => { /* replication paused */ })
  .on('active', () => { /* replication resumed */ })
  .on('denied', (err) => {
       /* a document failed to replicate */
  })
  .on('complete', (info) => { /* replication complete*/ })
  .on('error', (err) => { /* errors handling */ });
```

- In-memory ⇄ Local (кэш)

- In-memory ⇄ Local (кэш)
- Local ⇄ Local (без сервера)

- In-memory ⇄ Local (кэш)
- Local ⇄ Local (без сервера)
- Remote ⇄ Remote (можно, но не нужно)

- In-memory ⇄ Local (кэш)
- Local ⇄ Local (без сервера)
- Remote ⇄ Remote (можно, но не нужно)
- Local ⇄ Remote ⇄ Local (бэкап на сервере)

Map/Reduce функции

Что это?

Что это?

- Аналог индексов в SQL базах данных

Что это?

- Аналог индексов в SQL базах данных
- Параллельное выполнение на нескольких узлах

Что это?

- Аналог индексов в SQL базах данных
- Параллельное выполнение на нескольких узлах
- Разделяются на два типа:

Что это?

- Аналог индексов в SQL базах данных
- Параллельное выполнение на нескольких узлах
- Разделяются на два типа:
  1. Временные запросы (Temporary queries)

Что это?

- Аналог индексов в SQL базах данных
- Параллельное выполнение на нескольких узлах
- Разделяются на два типа:
  1. Временные запросы (Temporary queries)
  2. Постоянные запросы (Persistent queries)

```
1.  localDB
2.    .query(
3.      (doc, emit) => emit(doc.name),
4.      {key: 'foo'}
5.    )
6.    .then((result) => {
7.         /* documents with name === 'foo' */
8.    })
9.    .catch((err) => { /* errors handling */ });
```

```
1.  localDB
2.    .query(
3.      (doc, emit) => emit(doc.name),
4.      {key: 'foo'}
5.    )
6.    .then((result) => {
7.          /* documents with name === 'foo' */
8.    })
9.    .catch((err) => { /* errors handling */ });
```

- Очень медленные запросы

```
1.  localDB
2.    .query(
3.      (doc, emit) => emit(doc.name),
4.      {key: 'foo'}
5.    )
6.    .then((result) => {
7.         /* documents with name === 'foo' */
8.    })
9.    .catch((err) => { /* errors handling */ });
```

- Очень медленные запросы
- Рекомендуется использовать только для дебага

```
1.  const ddoc = {
2.    _id: '_design/my_index',
3.    views: {
4.      by_name: {
5.        map: function (doc) {
6.                emit(doc.name);
7.              }.toString()
8.      }
9.    }
10. };
```

```
const ddoc = {
  _id: '_design/my_index',
  views: {
    by_name: {
      map: function (doc) {
             emit(doc.name);
           }.toString()
    }
  }
};

pouch
  .put(ddoc)
  .then(() => { /* success */ })
  .catch((err) => { /* errors handling */ });
```

```
1.  localDB
2.    .query('my_index/by_name', {key: 'foo'})
3.    .then((res) => { /* handle result */ })
4.    .catch((err) => { /* handle error */ });
```

```
1.  localDB
2.    .query('my_index/by_name', {key: 'foo'})
3.    .then((res) => { /* handle result */ })
4.    .catch((err) => { /* handle error */ });
```

Индексируется только при первом выполнении

```
1.  localDB
2.    .query('my_index/by_name', {key: 'foo'})
3.    .then((res) => { /* handle result */ })
4.    .catch((err) => { /* handle error */ });
```

Индексируется только при первом выполнении

```
5.  localDB
6.    .query('my_index/by_name', {limit: 0})
7.    .then((res) => { /* handle result */ })
8.    .catch((err) => { /* handle error */ });
```

```
1.  function mapFunction(doc) {
2.      emit(doc.name);
3.  }
```

```
1.  function mapFunction(doc) {
2.      emit(doc.name);
3.  }
```

```
4.  function mapFunction(doc) {
5.      if (doc.type === 'brick') {
6.          if (doc.color === 'blue') {
7.              emit('Wow! Blue brick!');
8.          } else {
9.              emit(doc.color);
10.         }
11.     }
12. }
```

```
1.  pouch
2.    .query(mapFunction, {key: 'yellow'})
3.    .then((result) => { /* handle result */ })
4.    .catch((err) => { /* handle errors */ });
```

```
pouch
   .query(mapFunction, {key: 'yellow'})
   .then((result) => { /* handle result */ })
   .catch((err) => { /* handle errors */ });


pouch
   .query(mapFunction, {
      startkey: 'C',
      endkey: 'C\uffff',
      limit: 5,
      include_docs: true
   })
   .then((result) => { /* handle result */ })
   .catch((err) => { /* handle errors */ });
```

```
1.  const mapReduceFun = {
2.      map: (doc) => emit(doc.name.charAt(0)),
3.      reduce: '_count'
4.  };
```

```
1.  const mapReduceFun = {
2.      map: (doc) => emit(doc.name.charAt(0)),
3.      reduce: '_count'
4.  };



5.  pouch.query(mapReduceFun, {
6.      key: 'P',
7.      reduce: true,
8.      group: true
9.  })
10.  .then((result) => { /* handle result */ })
11.  .catch((err) => { /* handle errors */ });
```

Итого

Итого

NoSQL база в браузере, в памяти, на сервере, везде и всюду

Итого

NoSQL база в браузере, в памяти, на сервере, везде и всюду

Автоматическая синхронизация из коробки

Итого

NoSQL база в браузере, в памяти, на сервере, везде и всюду

Автоматическая синхронизация из коробки

Репликация во все стороны из коробки

Итого

NoSQL база в браузере, в памяти, на сервере, везде и всюду

Автоматическая синхронизация из коробки

Репликация во все стороны из коробки

Интеграция с CouchDB и другими базами данных

Итого

NoSQL база в браузере, в памяти, на сервере, везде и всюду

Автоматическая синхронизация из коробки

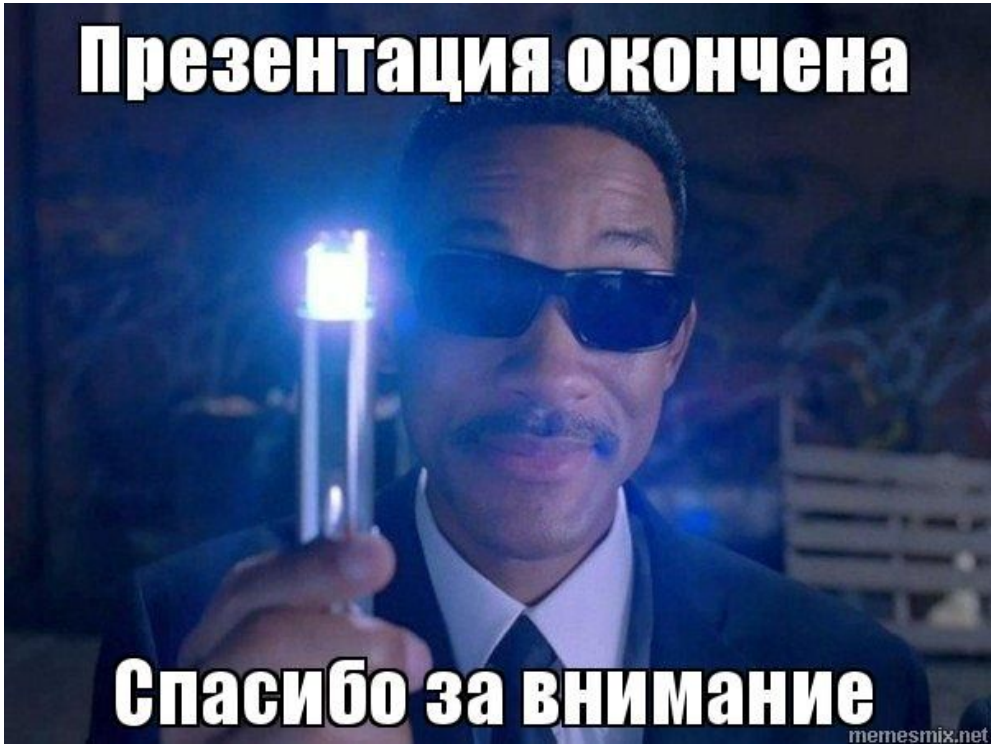
Репликация во все стороны из коробки

Интеграция с CouchDB и другими базами данных

Куча плагинов на все случаи жизни

Спасибо за внимание!

Вопросы?



@ZurabBelyi

BelyiZ

BelyiZ