RDBアンチパターン

PHPerに知ってほしい
リレーショナルデータベースのこと

What is it?

質問はTwitterのハッシュタグ

#phpcon_rdb_qa

でお願いします

What is it?

データベースの寿命は

アプリケーションよりも長い

What is it?

そんな長い付き合いになるデータベース

の大切な事をお伝えします

What is it?

対象のデータベースソフトウェア

What is it?

PostgreSQL 9.6とMySQL 5.7(InnoDB)に限る

他のRDBの話はしません

What is it?

RDBアンチパターン

What is it?

正規化して...

実行計画を見て...

SQLを書いて...

What is it?

良かれと思ってやったのに落ちてしまう罠

What is it?

アンチパターン

What is it?

アンチパターン

↓

後々に苦しみを生む

What is it?

アンチパターンを知る

What is it?

アンチパターンを知る

↓

同じ過ちを繰り返さない

What is it?


Avoiding the Pitfalls of Database Programming
SQL
アンチパターン
Bill Karwin 著
和田 卓人 監訳
和田 省二
児島 修 訳
O'REILLY
オライリー・ジャパン

What is it?

SQLアンチパターンは厳選された失敗集

What is it?

SQLアンチパターンは厳選された失敗集

↓

DBのノウハウが詰まった名著

What is it?

今日はRDBの失敗例をご紹介します

あじぇんだ

1　自己紹介
2　強すぎる依存
3　隠された状態
4　ロックの功罪
5　まとめ

あじぇんだ

1　自己紹介
2　強すぎる依存
3　隠された状態
4　ロックの功罪
5　まとめ

自己紹介

名前：曽根　壮大（そね　たけとも）

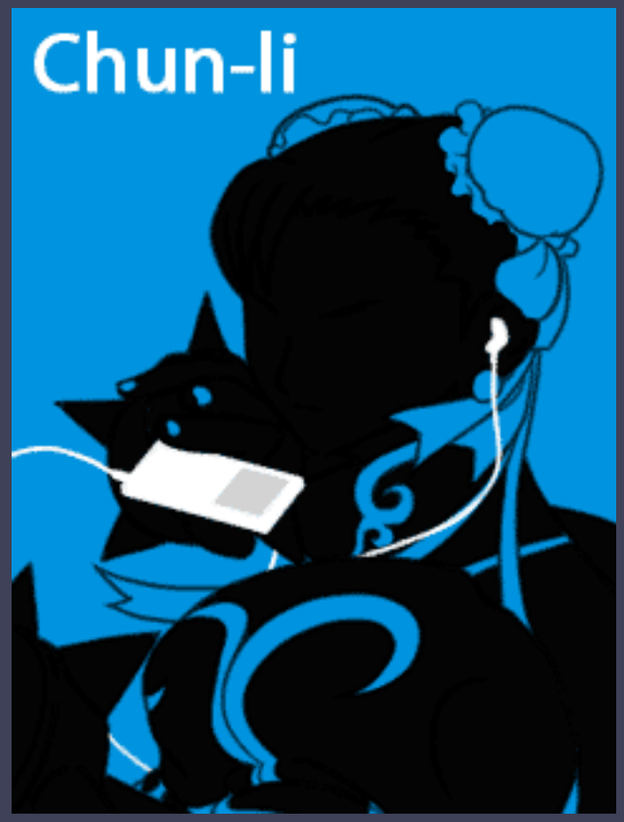
年齢：32歳（三人の子供がいます）

職業：Webエンジニア

所属：日本PostgreSQLユーザ会

　　中国支部 支部長

技術的にはLL系言語とかRDBが好きです



自己紹介

名前：曽根　壮大（そね　たけとも）

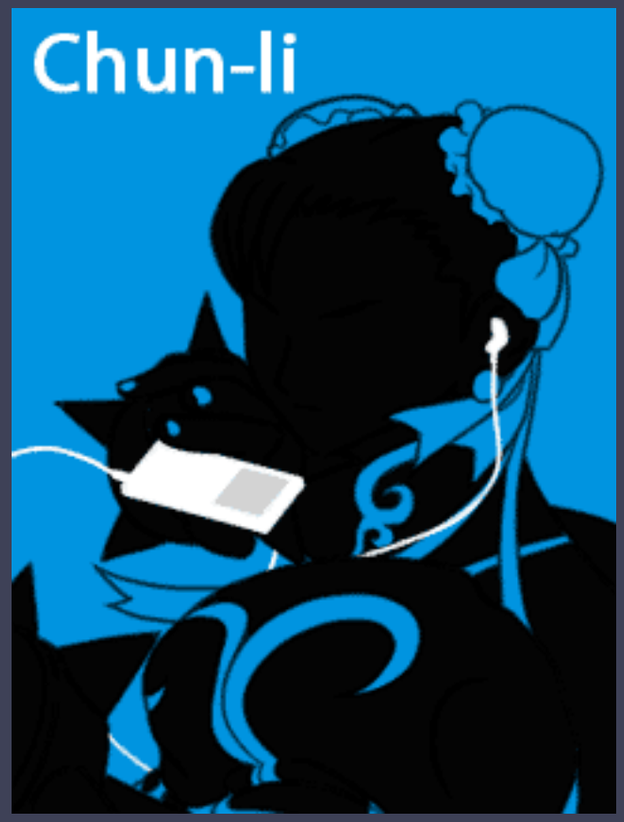
年齢：32歳（三人の子供がいます）

職業：Webエンジニア

所属：日本PostgreSQLユーザ会

中国支部 支部長

技術的にはLL系言語とかRDBが好きです



中国地方DB勉強会

https://dbstudychugoku.github.io/

中国地方DB勉強会

過去の資料が控えめに言っても

最高で最強なので是非チェック！

https://dbstudychugoku.github.io/

あじぇんだ

1　自己紹介
2　強すぎる依存
3　隠された状態
4　ロックの功罪
5　まとめ

強すぎる依存

データベースの抽象化

強すぎる依存

- Data Access Object（DAO）
- Object Relational Mapping(ORM)
- DB Migration

強すぎる依存

- Doctrine 2
- Eloquent ORM
- Idiorm
- Paris
- などなど…

強すぎる依存

これらの手法やツールはとても便利

強すぎる依存

これらの手法やツールはとても便利

↓

制約・誓約と引き換えに

高い生産性を与えてくれる

強すぎる依存

制約と誓約

強すぎる依存

制約

実装上のルールを強いる事で
コードの記述量を減らす

強すぎる依存

守らないとコンパイルエラー
実装や設計に直結

制約

実装上のルールを強いる事で
コードの記述量を減らす

強すぎる依存

守らないとコンパイルエラー
実装や設計に直結

制約

テーブルのidカラム必須
指定できる型
データアクセスの手段など

実装上のルール

コードの記述量を減らす

強すぎる依存

誓約

決まりごと(パターン)を守る事で
設計や実装の一貫性と再利用が可能

強すぎる依存

守らなくてもコードは動く
モラルや知識の不足などで無視される
ケースがある

誓約

決まりごと(パターン)を守る事で
設計や実装の一貫性と再利用が可能

強すぎる依存

守らなくてもコードは動く
モラルや知識の不足などで無視される
ケースがある

誓約

デザインパターン
設計思想
コード規約など

決まりごと(パター

設計や実装の一貫性と再利用が可能

強すぎる依存

データベースの抽象化は

ルール　マナー

制約と誓約の世界

強すぎる依存

制約と誓約と引き換えに

失っている事

強すぎる依存

- RDBの機能
- SQLの強力な文法
- RDBの本来の責務と設計

強すぎる依存

- RDBの機能
- SQLの強力な文法
- RDBの本来の責務と設計

強すぎる依存

型とかマテビューとかトリガーとかcheck制約とか

- RDBの機能
- SQLの強力な文法
- RDBの本来の責務と設計

強すぎる依存

- RDBの機能
- SQLの強力な文法
- RDBの本来の責務と設計

強すぎる依存

- RDBの
- SQLの強力な文法
- RDBの本来の責務と設計

Window関数とかWITH句とか

強すぎる依存

- RDBの機能
- SQLの強力な文法
- RDBの本来の責務と設計

強すぎる依存

- RDBの機能
- SQLの強力な文法
- RDBの本来の責務と設計

強すぎる依存

RDBの責務

- データを保存する
- データを取り出す
- データを守る

強すぎる依存

あなたのデータ

ちゃんと守れてますか？

強すぎる依存

DBの設計に強い制約が発生する

強すぎる依存

DBの設計に強い制約が発生する

↓

ORM を意識したスキーマ設計

RDBMS を意識したクラス設計

強すぎる依存

DBの設計に強い制

両輪がキレイに回っているときは
高い生産性を発揮する

↓

ORM を意識したスキーマ設計

RDBMS を意識したクラス設計

強すぎる依存

制約と誓約が崩れるとき

強すぎる依存

データが成長した時

強すぎる依存

サービスが成長した時

データが成長した時

強すぎる依存

サービスが成長した時

データが成長した時

↓

アプリケーション側だけで
カバー出来なくなった時

強すぎる依存

制約と誓約が崩れる時の例

- クエリが捌けなくなった
- シャーディングが必要
- INDEXが必要だが利用できない
- ALTER文が怖くて流せない

強すぎる依存

パフォーマンス問題によって制約と誓約が崩れる

制約と誓約が崩れる時の例

- クエリが捌けなくなった
- シャーディングが必要
- INDEXが必要だが利用できない
- ALTER文が怖くて流せない

強すぎる依存

現状ではORMは

インピーダンス・ミスマッチ
を解決するための存在

であってRDBの完全な抽象化ではない

強すぎる依存

処方箋

強すぎる依存

処方箋

制約と誓約によって

失っている存在を忘れないこと

あじぇんだ

1　自己紹介
2　強すぎる依存
3　隠された状態
4　ロックの功罪
5　まとめ

隠された状態

データベースは

データ保存するただの箱では無い

隠された状態

リレーショナルモデル

隠された状態

リレーショナルモデル

↓

事実のみを保存する

隠された状態

事実と状態

隠された状態

状態の例

- ユーザの状態(statusカラム)
- 課金状態（与信、請求済み）
- カート情報（カゴの中、購入済み）

隠された状態

レコードに状態を持たせると危険

id	name	age	delete_flag
1	sone	31	1
2	soudai	33	0
3	taketomo	32	0
4	sone	31	0

id	name	age	delete_flag
1	sone	31	1
2			0
3	taketomo	32	0
4	sone	31	0

削除と言う状態をフラグで表現

隠された状態

```
$user = $this->get_user($user_id);

if ($user->get('delete_falg') == 1)
{
    // 削除済みのユーザの処理
    return
}
```

隠された状態

```
$user = $this->get_user($user_id);

if ($user->get('delete_falg') == 1)
{
    // 削
    return
}
```

データの中の状態はアプリケーション
は取り出すまでわからない

隠された状態

- Unique制約が死んだり
- INDEXが死んだり
- 外部キー制約が死んだり

隠された状態

レコードから

状態が見えない時は

もっと危険

売上

売上	売上金額	売上日	配送状態
1	29,522	2014-03-31 23:59:59	配送済
2	6480	2014-04-01 00:00:00	発注中
⋮	⋮	⋮	

カート

売上id	商品id	個数	購入者
1	1	3	sone
1	2	3	sone
1	3	3	sone
2	4	2	sone

商品

商品id	商品名	価格
1	SQL実践入門	2,580
2	リーダブルコード	2,592
3	プログラマのためのSQL	4,200
4	データベース・リファクタリング	3,000

売上

売上	売上金額	売上日	配送状態
1	29,522	2014-03-31 23:59:59	配送済
2	6480	2014-04-01 00:00:00	発注中
⋮	⋮	⋮	

カート

売上id	商品id	個数	購入者
1	1	3	sone
1	2	3	sone
1	3	3	sone
2	4	2	sone

商品

商品id	商品名	価格
1	SQL実践入門	2,580
2	リーダブルコード	2,592
3	プログラマのためのSQL	4,200
4	データベース・リファクタリング	3,000

売上

売上	売上金額	売上日	配送状態
1	29,522	2014-03-31 23:59:59	
2	6480	2014-04-01 00:00:00	
⋮	⋮	⋮	

消費税の状態が隠れてる
5%と8%が日付で変わる

カート

売上id	商品id	個数	購入者
1	1	3	sone
1	2	3	sone
1	3	3	sone
2	4	2	sone

商品

商品id	商品名	価格
1	SQL実践入門	2,580
2	リーダブルコード	2,592
3	プログラマのためのSQL	4,200
4	データベース・リファクタリング	3,000

売上

売上	売上金額		配送状態
1	29,52		配送済
2	6480	2014-04-01 00:00:00	発注中
⋮	⋮	⋮	

いつ配送状態が
変わったかわからない

カート

売上id	商品id	個数	購入者
1	1	3	sone
1	2	3	sone
1	3	3	sone
2	4	2	sone

商品

商品id	商品名	価格
1	SQL実践入門	2,580
2	リーダブルコード	2,592
3	プログラマのためのSQL	4,200
4	データベース・リファクタリング	3,000

売上

売上	売上金額	売上日	配送状態
1	29,522	2014-03-31 23:59:59	配送済
2	6480	2014-04-01 00:00:00	発注中
⋮	⋮	⋮	

カート

売上id	商品id	個数	購入者
1	1	3	sone
1	2	3	sone
1	3	3	sone
2	4	2	sone

商品

商品id	商品名	価格
1	SQL実践入門	2,580
2	リーダブルコード	2,592
3	プログラマのためのSQL	4,200
4	データベース・リファクタリング	3,000

売上

売上	売上金額	売上日	配送状態
1	29,522	2014-03-31 23:59:59	配送済
2	6480	2014-04-01 00:00:00	発注中
⋮	⋮	⋮	

カート

売上id	商品id	個数	購入者
1	1	3	sone
1	2	3	sone
1	3	3	sone
2	4	2	sone

商品名が変わったら？
価格を変えたら？
売上の事実と不整合が生まれる

商品id	商品名	価格
1	SQL実践入門	2,580
2	リーダブルコード	2,592
3	プログラマのためのSQL	4,200
4	データベース・リファクタリング	3,000

隠された状態

隠された状態の例

- 配送状態の履歴
- 購入履歴の消費税の内訳
- 削除や更新の履歴

隠された状態

事実のみを保存する

隠された状態

事実のみを保存する

↓

事実の履歴を保存する

売上

売	売上金	消費税	消費税額	売上日
1	29,522	5%	1406	2014-03-31 23:59:59
2	6480	8%	480	2014-04-01 00:00:00
⋮	⋮			⋮

消費税

消費税	有効日	失効日
5%	1997-04-01	2014-03-31
8%	2014-04-01	null

配送状況

売上id	配送状態	作成日時
1	発注中	2014-03-31 11:59:59
1	配送中	2014-03-31 15:59:59
1	納品済	2014-03-31 18:59:59
2	発注中	2014-03-31 23:59:59

売上

売	売上金	消費税	消費税額	売上日
1	29,522	5%	1406	2014-03-31 23:59:59
2	6480	8%	480	2014-04-01 00:00:00
⋮	⋮			⋮

消費税

消費税	有効日	失効日
5%	1997-04-01	2014-03-31
8%	2014-04-01	null

配送状況

売上id	配送状態	作成日時
1	発注中	2014-03-31 1
1	配送中	2014-03-31 1
1	納品済	2014-03-31 18:59:59
2	発注中	2014-03-31 23:59:59

配送状況の履歴を持つ
更新では無く追加で歴史を作る

売上

売	売上金	消費税	消費税額	売上日
1	29,522	5%	1406	2014-03-31 23:59:59
2	6480	8%	480	2014-04-01 00:00:00
⋮	⋮			⋮

消費税

消費税	有効日	失効日
5%	1997-04-01	2014-03-31
8%	2014-04-01	null

同様に消費税の歴史を持つ
返品時の払い戻しの処理の時など
で持ってないと死ぬ

配送状況

売上id	配送状態	作成日時
1	発注中	2014-03-31 11:59:59
1	配送中	2014-03-31 15:59:59
1	納品済	2014-03-31 18:59:59
2	発注中	2014-03-31 23:59:59

売上

売	売上金	消費税	消費税額	売上日
1	29,522	5%	1406	2014-03-31 23:59:59
2	6480	8%	480	2014-04-01 00:00:00
⋮	⋮			⋮

消費税

消費税	有効日	失効日
5%	1997-04-01	2014-03-31
8%	2014-04-01	null

同様に消費税の歴史を持つ
返品時の払い戻しの処理の時など
で持ってないと死ぬ

PostgreSQLの範囲型を使うと有効範囲
をスムーズに処理出来る

売上id		
1		9:59
1	配送中	2014-03-31 15:59:59
1	納品済	2014-03-31 18:59:59
2	発注中	2014-03-31 23:59:59

売上

売	売上金	消費税	消費税額	
1	29,522	5%	1406	20
2	6480	8%	480	2014-04-01 00:00:00
⋮	⋮			⋮

計算したら得る事が出来る
非正規化の一つ
パフォーマンスと相談

消費税

消費税	有効日	失効日
5%	1997-04-01	2014-03-31
8%	2014-04-01	null

配送状況

売上id	配送状態	作成日時
1	発注中	2014-03-31 11:59:59
1	配送中	2014-03-31 15:59:59
1	納品済	2014-03-31 18:59:59
2	発注中	2014-03-31 23:59:59

売上

売	売上金	消費税	消費税額	
1	29,522	5%	1406	20
2	6480	8%	480	20
⋮	⋮			

計算したら得る事が出来る
非正規化の一つ
パフォーマンスと相談

MySQLの仮想列を利用すると
よりキレイに実装出来る

消費税

消費税	有効日	失効日
5%	1997-04-01	2014-03-31
8%	2014-04-01	null

配送状況

売上id	配送状態	作成日時
1	発注中	2014-03-31 11:59:59
1	配送中	2014-03-31 15:59:59
1	納品済	2014-03-31 18:59:59
2	発注中	2014-03-31 23:59:59

隠された状態

データベース上に無いデータは

アプリ側で作り出せない

隠された状態

隠された状態があると

例外対応の時に死ぬ

隠された状態

処方箋

隠された状態

処方箋

適正な形でデータを保存する

隠された状態

処方箋

適正な形でデータを保存する

↓

ちゃんとリスクヘッジをする

あじぇんだ

1　自己紹介
2　強すぎる依存
3　隠された状態
4　ロックの功罪
5　まとめ

ロックの功罪

トランザクションとデータ整合性

ロックの功罪

トランザクション分離レベル

ロックの功罪

トランザクション分離レベル

名前	分離レベル	説明
SERIALIZABLE	さいつよ	直列的に処理
REPEATABLE READ	強い	読み取り対象のデータを常に読み取る
READ COMMITTED	まぁま強い	確定した最新データを常に読み取る
READ UNCOMMITTED	弱い	他の処理によって行われている、書きかけのデータまで読み取る。

ロックの功罪

トランザクション分離レベル

名前	分離レベル	説明
SERIAL		直列的に処理
REPEATABLE READ	強い	読み取り対象のデータを常に読み取る
READ COMMITTED	まぁま強い	確定した最新データを常に読み取る
READ UNCOMMITTED	弱い	他の処理によって行われている、書きかけのデータまで読み取る。

MySQLのデフォルトはここ

ロックの功罪

トランザクション分離レベル

名前	分離レベル	説明
SERIALIZABLE	さいつよ	直列的に処理
REPEATA		対象のデータ 読み取る
READ COMMITTED	まぁま強い	確定した最新データを常に読み取る
READ UNCOMMITTED	弱い	他の処理によって行われている、書きかけのデータまで読み取る。

PostgreSQL, OracleDB, SQLServer のデフォルトはここ

ロックの功罪

トランザクション分離レベル

名前	分離レベル	説明
SERIALIZABLE	さいつよ	直列的に処理
REPEATA		対象のデータ 読み取る
READ CON		
READ UNCOMMITTED	弱い	れている、書きかけのデータまで読み取る。

PostgreSQL, OracleDB, SQLServer のデフォルトはここ

ファントムリードやダーティリードなどの話は今日はしません
「え？なにそれ」って人は今すぐググッて勉強した方がいいです

ロックの功罪

なぜトランザクションが必要か

ロックの功罪

ユーザA

ユーザB

同時に購入

商品A

ロックの功罪

ユーザA

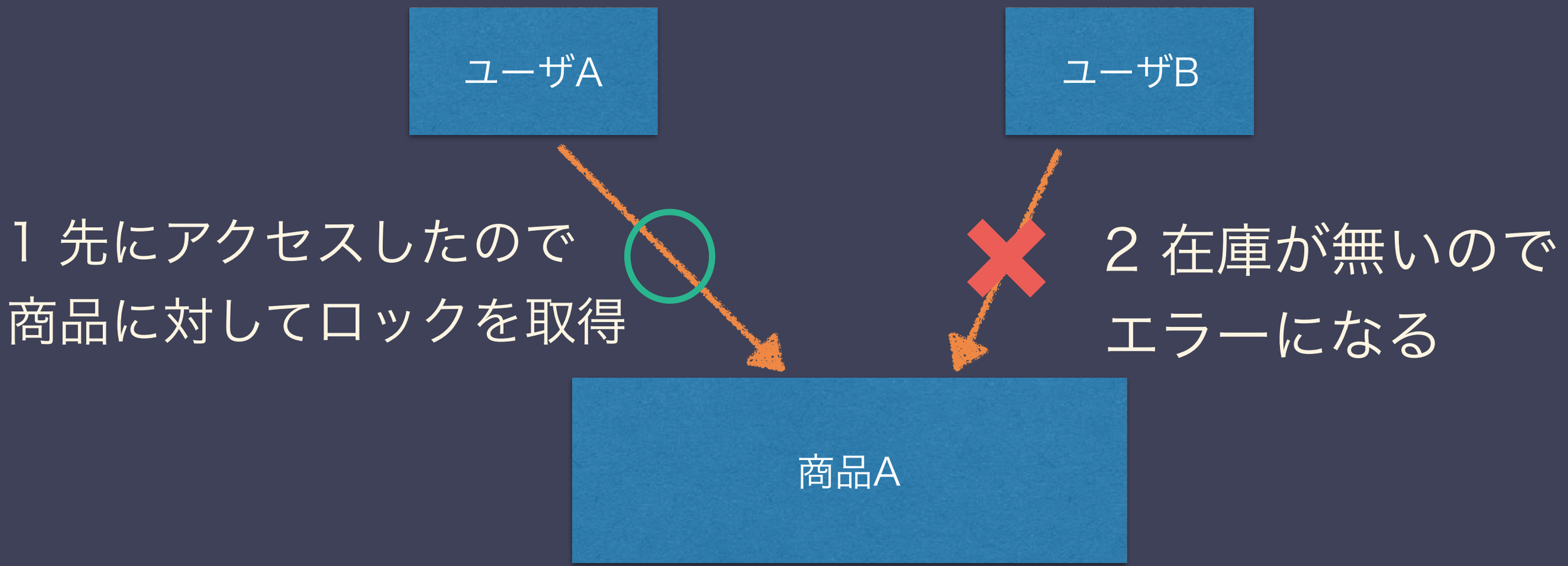
ユーザB

同時に購入

在庫が最後の1個だったとき、
どちらを優先するか？

商品A

ロックの功罪



ロックの功罪

並列処理のデータの不整合を防ぐ

ロックの功罪

あなたの処理は大丈夫？

- 複数のブラウザを開いて同時アクセス
- バズった時などの大量の同時アクセス時
- 並列処理を書いた時

ロックの功罪

あなたの処理は大丈夫？

- 複数のブラウザを開いて同時アクセス
- バズった時などの大量の同時アクセス時
- 並列処理を書いた時

ロックの功罪

あなたの処理は大丈夫？

例えばデイリーの来店ポイントの処理とか
商品購入時のポイント減算の処理とか

- 複数のブラウザを開いて同時アクセス
- バズった時などの大量の同時アクセス時
- 並列処理を書いた時

ロックの功罪

あなたの処理は大丈夫？

- 複数のブラウザを開いて同時アクセス
- バズった時などの大量の同時アクセス時
- 並列処理を書いた時

ロックの功罪

あなたの処理は大丈夫？

- 複数の 時アクセス
- バズった時などの大量の同時アクセス時
- 並列処理を書いた時

○○限定商品の在庫処理とか
一人あたりの購入限界数の処理とか

ロックの功罪

あなたの処理は大丈夫？

- 複数のブラウザを開いて同時アクセス
- バズった時などの大量の同時アクセス時
- 並列処理を書いた時

ロックの功罪

あなたの処理は大丈夫？

- 複数のブラウザを開いて同時アクセス
- バズ
- 並列処理を書いた時

一つのデータに対して複数のサービスがアクセスする時とか
高速化のために非同期処理やマルチスレッド処理を作った時とか

ロックの功罪

ロックの責務

事実に基いて時系列に処理して
データを適切に守る

ロックの功罪

ここまではロックの功罪の功の部分

ロックの功罪

ロックの罪

ロックの功罪

ロックによる性能遅延

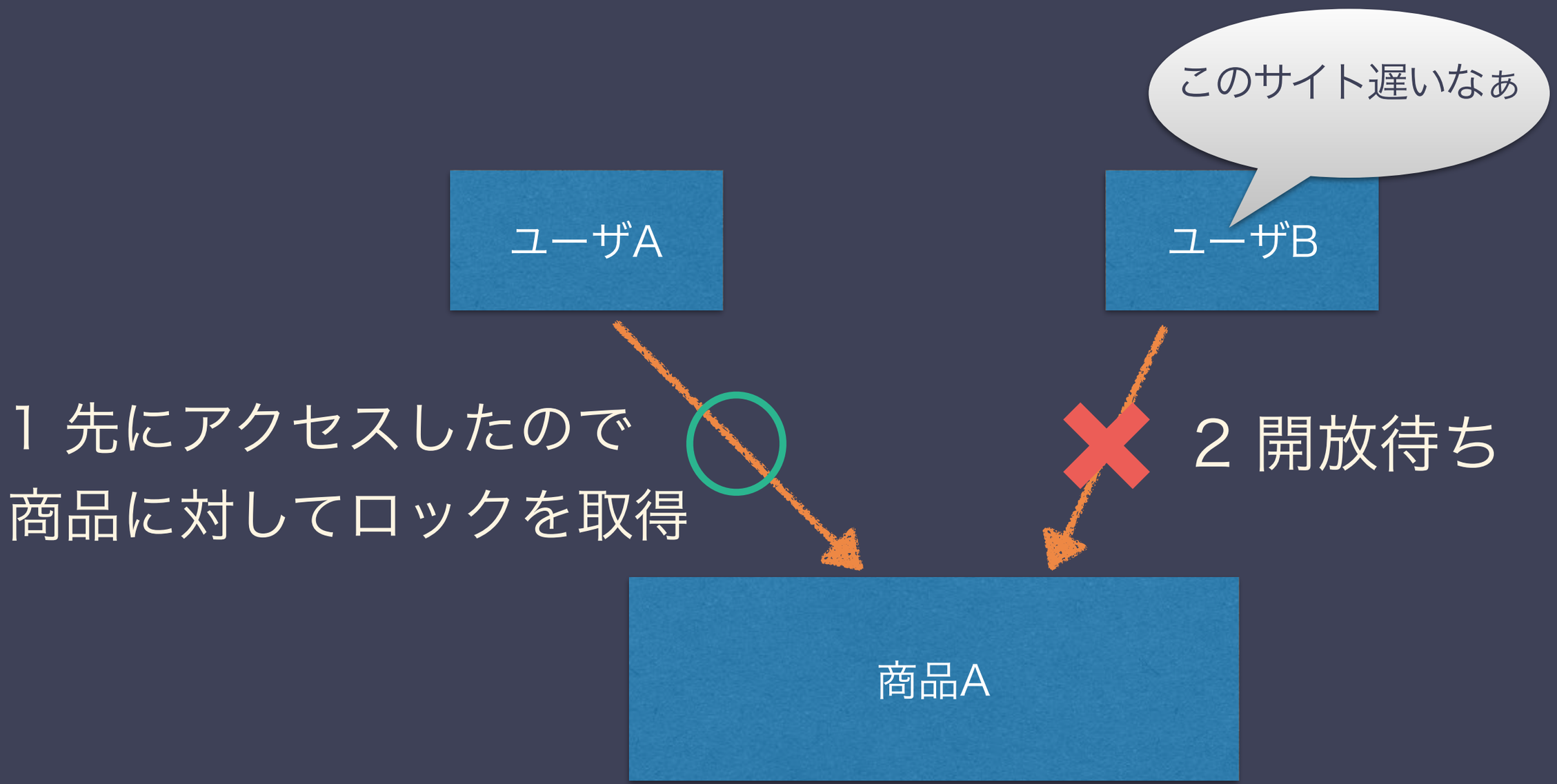
ロックの功罪

ロックによる性能遅延

↓

ロック待ちによる処理遅延

ロックの功罪



ロックの功罪

両方のユーザがそれぞれのロックで
待たされた場合はデットロック

このサイト遅いなぁ

ユーザA

ユーザB

1 先にアクセスしたので
商品に対してロックを取得

2 開放待ち

商品A

ロックの功罪

両方のユーザがそれぞれのロックで
待たされた場合はデットロック

このサイト遅いなぁ

ユーザA

ユーザB

1 先にアクセスしたので
商品に対してロックを取得

2 開放待ち

商品A

ロックの功罪

主なロックの種類

- 排他(eXcluded)ロック
- 共有(Shared)ロック

ロックの功罪

- 排他(eXcluded)ロック
- 共有(Shared)ロック

ロック対象に対して、他のアクセスを一切禁止する
他のアクセスは更新・削除・参照全て出来ない
書き込みロックと呼ばれる事もある

ロックの功罪

主なロックの種類

- 排他
- 共有(Shared)ロック

ロック対象に対して、参照以外の処理を禁止する
他のアクセスは参照(SELECT)をすることが出来る
読み込みロックと呼ばれる事もある

ロックの功罪

ロックを知らない所で取っている

ロックの功罪

ロックを知らない所で取っている

↓

暗黙的ロック

ロックの功罪

MySQLの場合

ロックの功罪

PostgreSQLの場合

ロックの功罪

特に共有ロックは頻繁に取っている

ロックを知らない所で取っている

↓

暗黙的ロック

ロックの功罪

ロックはデータを守るために必要

ロックの功罪

ロックはデータを守るために必要

↓

パフォーマンスとトレードオフ

ロックの功罪

- ロック待ちで処理が詰まる
- 遅いSELECTによって更新処理が詰まる
- 更新対象がコンフリクトして待たされる
- デットロック、トランザクションは死ぬ

ロックの功罪

サービスのメインテーブルは危険

- 多くの参照で利用している
- 更新も頻繁にある
- JOINやサブクエリの対象になっている

ロックの功罪

そもそもロックの処理は

RDBによってかなり違う

ロックの功罪

そもそもロックの処理は

RDBによってかなり違う

処方箋を一括りで言えない

ロックの功罪

ロックはデータを守るために必要

ロックの功罪

ロックはデータを守るために必要

↓

しかしトラブルの時の根は深い

ロックの功罪

処方箋

ロックの功罪

処方箋

ロックを知る

ロックの功罪

未知なる仕組みは怖い

ロックの功罪

未知なる仕組みは怖い

↓

しっかりと理解する事

ロックの功罪

ロックを見る

ロックの功罪

ロックを見る

↓

各DBに用意されている仕組みを使う

ロックの功罪

ムーアの法則の限界

ロックの功罪

ムーアの法則の限界

↓

時代は並列・分散処理時代へ

ロックの功罪

ムーアの法則の限界

↓

時代は並列・分散処理時代へ

多くのソフトウェアエンジニアは並列処理が苦手
だからこそ守りの仕組みしっかり知る

あじぇんだ

1　自己紹介
2　強すぎる依存
3　隠された状態
4　ロックの功罪
5　まとめ

まとめ

機能やツールに依存し過ぎない

まとめ

機能やツールに依存し過ぎない

↓

シンプルな設計は全てに勝る

まとめ

データの寿命はコードよりも長い

まとめ

一度作ったDBは消せない

まとめ

一度作ったDBは消せない

↓

設計が大事

まとめ

データベースの死はサービスの死

まとめ

データベースの死はサービスの死

↓

解決できる人は英雄

まとめ

DBの問題は忘れた頃にやってくる

まとめ

そーだい@初代ALF
@soudai1025

DBの問題は忘れた頃にやってくる。
サーバーの問題は休日にやってくる。
アプリケーションの問題は納期前にやってくる。
脆弱性の発見は連休中にやってくる←New！！

23 リツイート　18 いいね

1:42 - 2016年5月6日

まとめ

サービスやチームを守るために

無知を既知する

まとめ

愚者は経験に学ぶ

賢者は過去に学ぶ

まとめ

周囲の経験談から学ぶ

まとめ

周囲の経験談から学ぶ

↓

積極的にコミュニティを利用する

参考資料

・postgresql-jp Slack(チャットルーム)

https://postgresql-hackers-jp.herokuapp.com/

・mysql-casual Slack(チャットルーム)

https://mysql-casual-slackin.herokuapp.com/

まとめ

RDBの知識は寿命が長い

まとめ

RDBの知識は寿命が長い

↓

覚えれば仕事で長い間役に立つ

まとめ

より良い設計を

一緒に考えて行きましょう

まとめ

RDBはいいぞ。

ご清聴ありがとうございました。