SQL
アンチパターン

和田 卓人 (@t_wada)

Oct 15, 2013 @ JavaFesta

#javafesta #sqlap

和田 卓人

id: t-wada

@t_wada

github: twada

Avoiding the Pitfalls of Database Programming

SQL アンチパターン

Bill Karwin 著
和田 卓人
和田 省二 監訳
児島 修 訳

#sqlap

SQL アンチパターン

Bill Karwin 著
和田 卓人 監訳
和田 省二 監訳
児島 修 訳

愚者は経験に学び、賢者は歴史に学ぶ。

—オットー・フォン・ビスマルク

愚者は経験に学ぶ、賢者は歴史に学ぶ。

―オットー・エドゥアルト・レオポルト・フォン・ビスマルク

誤訳!? 造句!?

諸君は自らの経験からいくらか学ぶことができるという、全く愚かな考えであろうが、余はむしろ**他人の失敗を学ぶことで、自分の失敗を回避する**ことを好む。

—オットー・フォン・ビスマルク

Nur ein Idiot glaubt, aus den eigenen Erfahrungen zu lernen. Ich ziehe es vor, aus den Erfahrungen anderer zu lernen, um von vorneherein eigene Fehler zu vermeiden.

Agenda

1. アンチパターンとは
2. 本書のダイジェスト
3. おわりに

アンチパターンとは

べからず集

あるある集

だけでは無い

本書のアンチパターンの構成

0. 名前

名前重要!!

1. 目的

2. アンチパターン

3. アンチパターンの見つけ方

4. アンチパターンを用いても良い場合

5. 解決策

名付けの例

例: ナイーブツリー
(素朴な木)

なぜカタカナ!?

パターン名が英語そのままカタカナ表記であるのは、目次を見ただけではビックリするポイントですね。

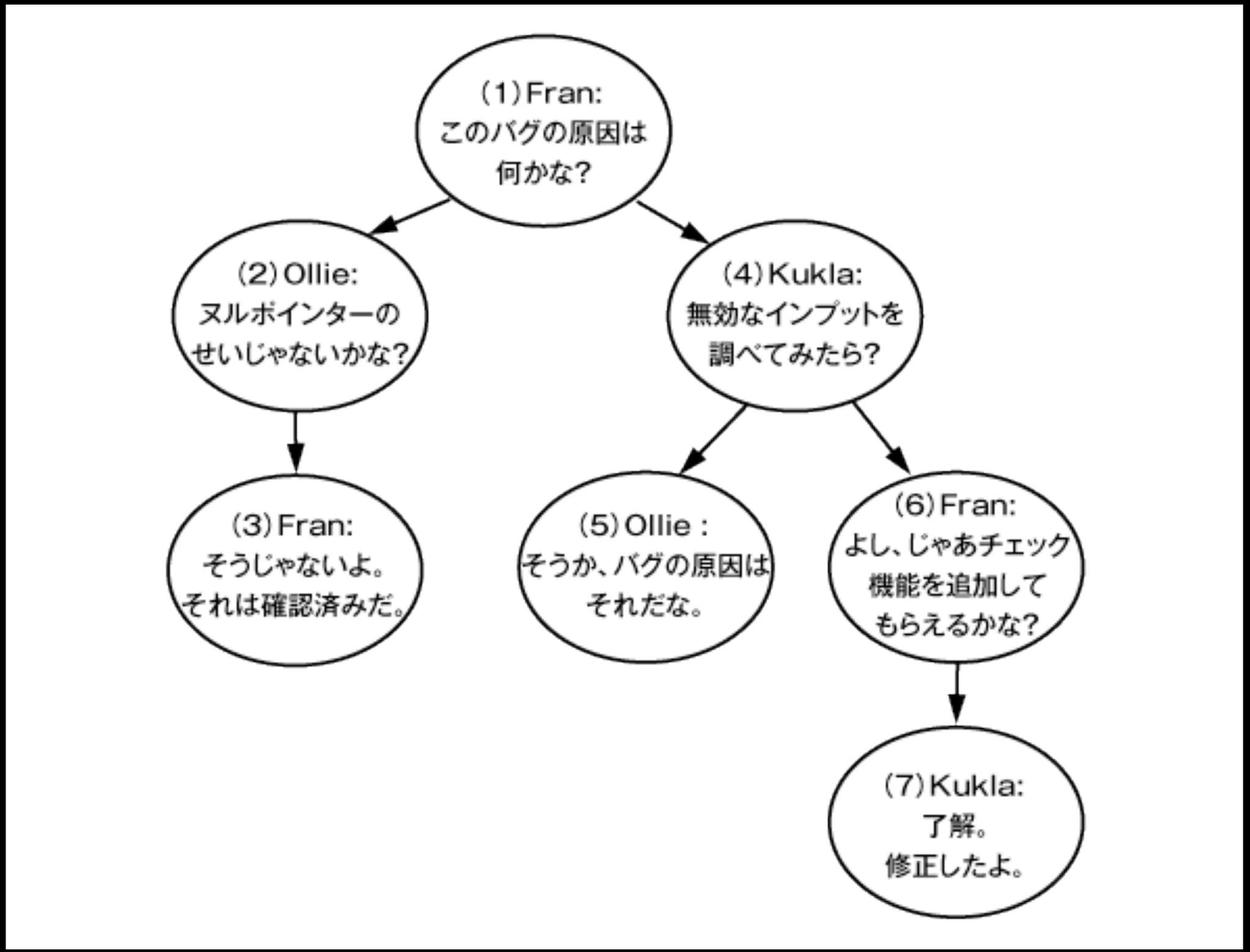
ただ、チーム内で相談するときなどに目立つ名前が付いているのはむしろありがたいですし、何よりなんかカッコよくておもしろかったです!

http://d.hatena.ne.jp/moro/20130205/1360044434

実例: ナイーブツリー(素朴な木)

0. 名前
1. 目的
2. アンチパターン
3. アンチパターンの見つけ方
4. アンチパターンを用いても良い場合
5. 解決策

目的: 階層構造を格納し、クエリを実行する



実例: ナイーブツリー(素朴な木)

0. 名前
1. 目的
2. アンチパターン
3. アンチパターンの見つけ方
4. アンチパターンを用いても良い場合
5. 解決策

アンチパターンとは何でしょうか。それは、**問題の解決を意図しながらも、しばしば他の問題を生じさせてしまうような技法**を指します。

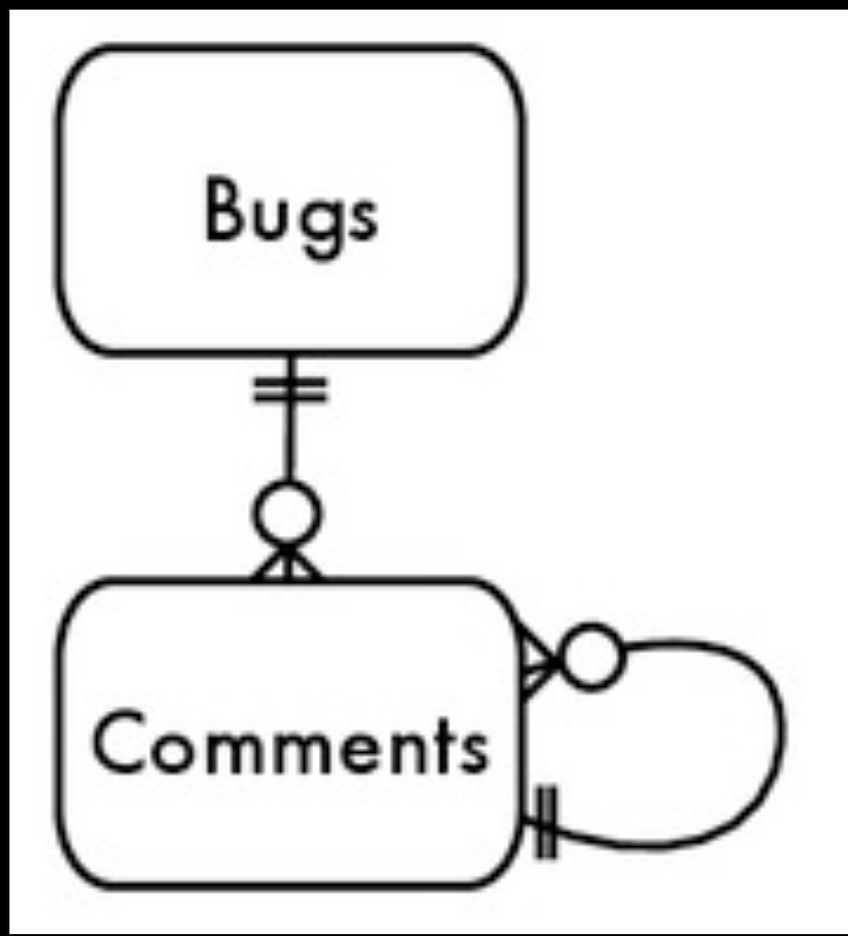
— Bill Karwin

よかれと思って裏目に出てしまうもの

アンチパターン: 常に親のみに依存する

```
CREATE TABLE Comments (
    comment_id    SERIAL PRIMARY KEY,
    parent_id     BIGINT UNSIGNED,
    comment       TEXT NOT NULL,
);
```

親idが入る



アンチパターンにより起こること

```
SELECT c1.*, c2.*, c3.*, c4.* -- 階層毎に列が増える
FROM Comments c1 -- 1階層目
  LEFT OUTER JOIN Comments c2
    ON c2.parent_id = c1.comment_id -- 2階層目
  LEFT OUTER JOIN Comments c3
    ON c3.parent_id = c2.comment_id -- 3階層目
LEFT OUTER JOIN Comments c4
  ON c4.parent_id = c3.comment_id -- 4階層目
```

素朴すぎる故に

アンチパターン

実例: ナイーブツリー(素朴な木)

0. 名前
1. 目的
2. アンチパターン
3. アンチパターンの見つけ方
4. アンチパターンを用いても良い場合
5. 解決策

直面している問題の種類や、メンバー間の会話での何気ない言葉が、そこにアンチパターンがあるかもしれないことに気づくヒントになります。

— Bill Karwin

アンチパターンの見つけ方

「このツリーでは、深さを何階層までサポートすればいい？」

「ツリー型のデータ構造を扱うコードなんて二度と書きたくないな」

「ツリーの中で孤児になった行をきれいにするために、定期的にスクリプトを実行しなければ」

実例: ナイーブツリー(素朴な木)

0. 名前
1. 目的
2. アンチパターン
3. アンチパターンの見つけ方
4. アンチパターンを用いても良い場合
5. 解決策

アンチパターンを用いても良い場合

共通テーブル式(CTE：common table expression)
を使って再帰クエリを書ける場合

```
WITH CommentTree
     (comment_id, bug_id, parent_id, author, comment, depth)
AS (
     SELECT *, 0 AS depth FROM Comments
     WHERE parent_id IS NULL
   UNION ALL
     SELECT c.*, ct.depth+1 AS depth FROM CommentTree ct
     JOIN Comments c ON ct.comment_id = c.parent_id
)
SELECT * FROM CommentTree WHERE bug_id = 1234;
```

アンチパターンを用いても良い場合

本書にはアンチパターンを適用しても良い状況の説明もあって好感が持てます。(略) この本は単なる「べからず集」ではなく「パターン本」だからです。コンテキストや制約が異なれば導かれる解法も異なるというわけです。

http://yojik.hatenablog.jp/entry/2013/02/13/235729

実例: ナイーブツリー(素朴な木)

0. 名前
1. 目的
2. アンチパターン
3. アンチパターンの見つけ方
4. アンチパターンを用いても良い場合
5. 解決策

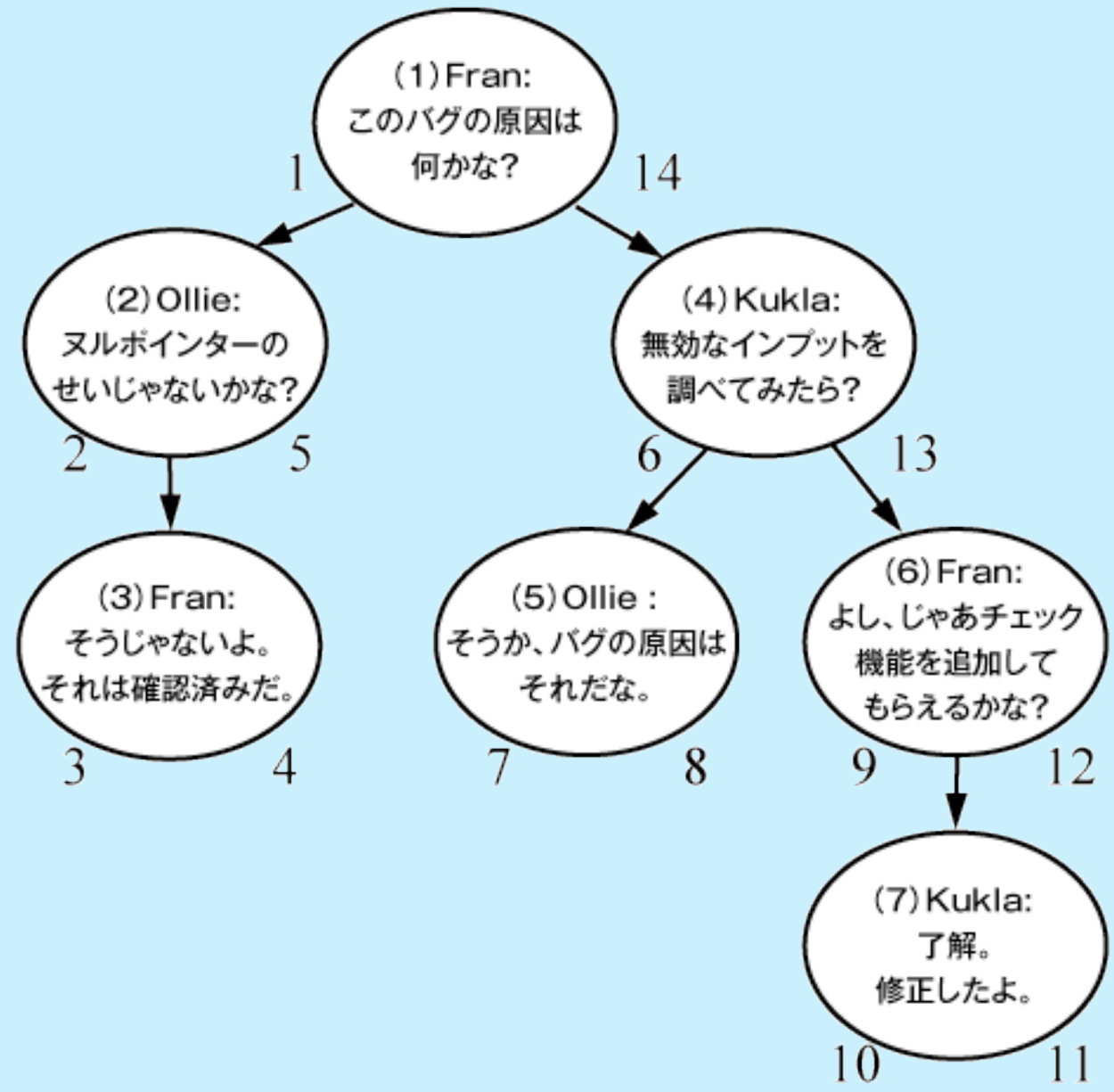
解決策: 代替ツリーモデルを使用する

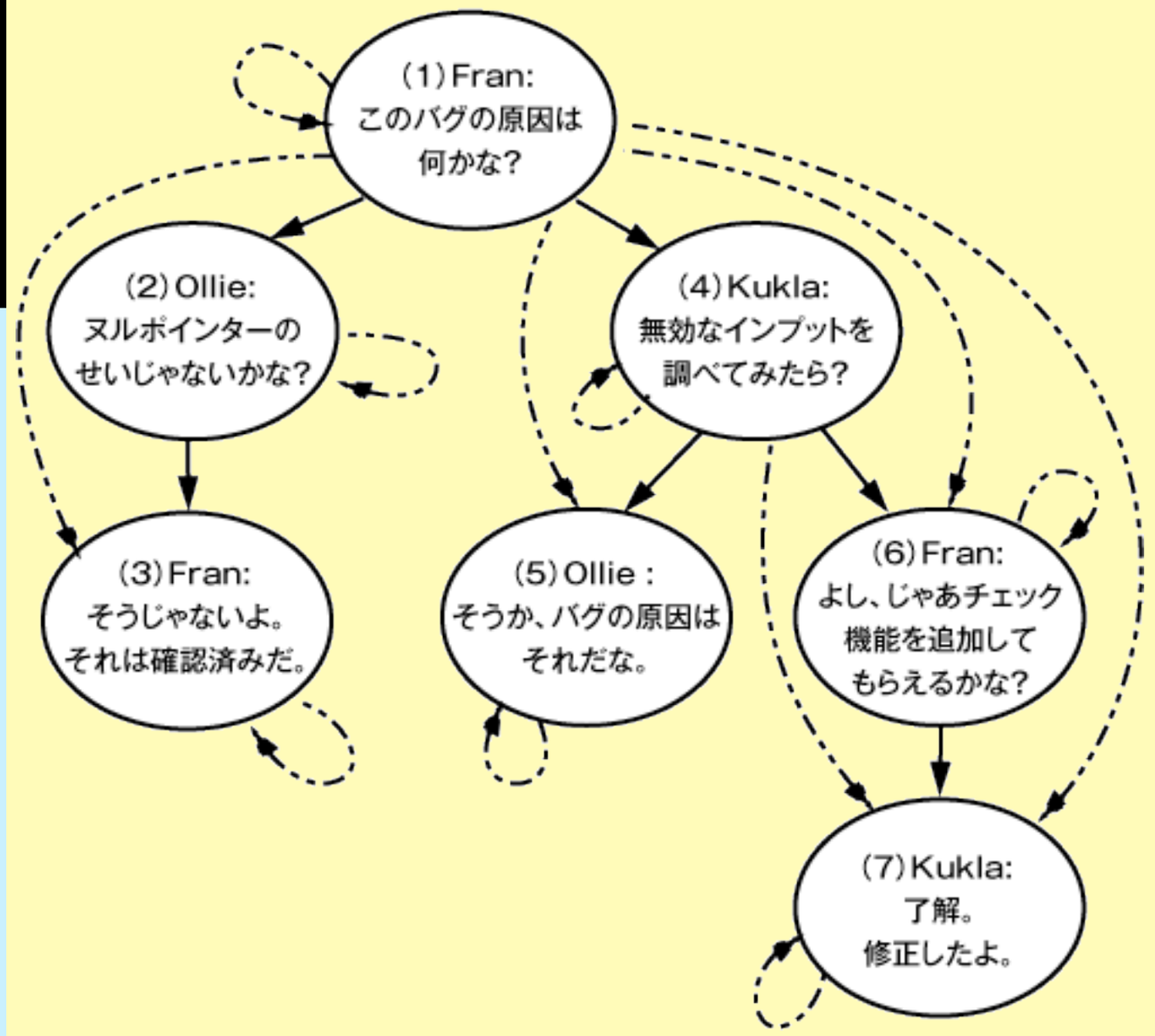
策1: 経路列挙 (Path Enumeration)

comment_id	path	発言者	コメント
1	1/	Fran	このバグの原因は何かな？
2	1/2/	Ollie	ヌルポインターのせいじゃないかな？
3	1/2/3/	Fran	そうじゃないよ。それは確認済みだ。
4	1/4/	Kukla	無効なインプットを調べてみたら？
5	1/4/5/	Ollie	そうか、バグの原因はそれだな。
6	1/4/6/	Fran	よし、じゃあチェック機能を追加してもらえるかな？
7	1/4/6/7/	Kukla	了解。修正したよ。

解決策: 代替ツリーモデルを使用する

2. 入れ子集合 NestedSet





3. 閉包テーブル Closure Table

解決策: 代替ツリーモデルを使用する

解決策の比較表

設計	テーブル数	子へのクエリ実行	ツリーへのクエリ実行	挿入	削除	参照整合性維持
隣接リスト	1	簡単	難しい	簡単	簡単	可能
再帰クエリ	1	簡単	簡単	簡単	簡単	可能
経路列挙	1	簡単	簡単	簡単	簡単	不可
入れ子集合	1	難しい	難しい	難しい	難しい	不可
閉包テーブル	2	簡単	簡単	簡単	簡単	可能

Agenda

1. アンチパターンとは
2. 本書のダイジェスト
3. おわりに

4つの部

25のパターン

を一巡り

1. 論理設計
2. 物理設計
3. クエリ
4. アプリケーション

論理設計のアンチパターン

1. ジェイウォーク（信号無視）
2. ナイーブツリー（素朴な木）
3. IDリクワイアド（とりあえずID）

4. キーレスエントリ（外部キー嫌い）
5. EAV（エンティティ・アトリビュート・バリュー）
6. ポリモーフィック関連
7. マルチカラムアトリビュート（複数列属性）
8. メタデータトリブル（メタデータ大増殖）

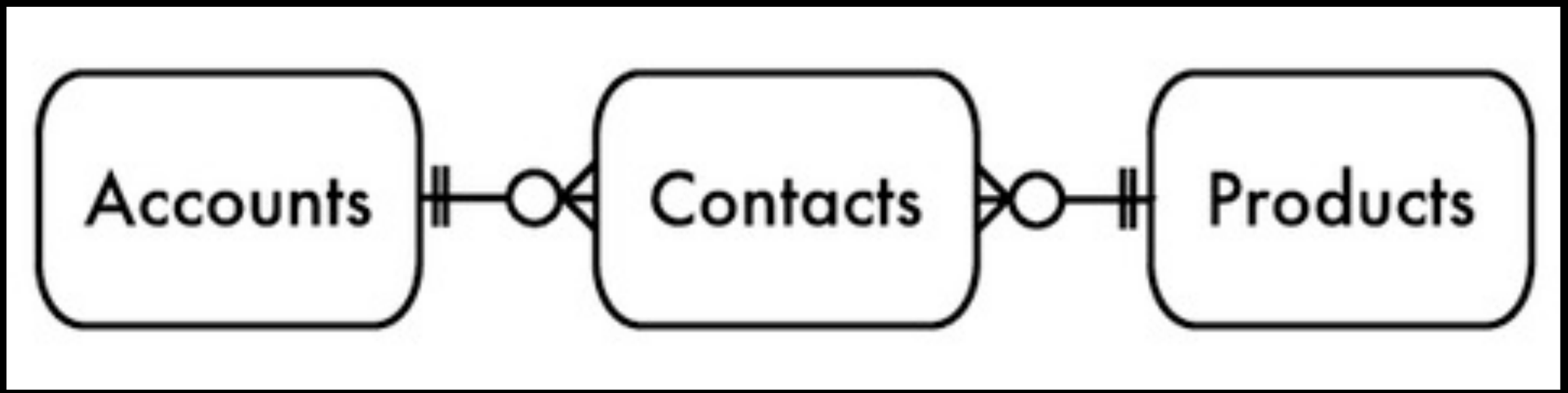
ジェイウォーク (信号無視)

```
CREATE TABLE Products (
   product_id    SERIAL PRIMARY KEY,
   product_name VARCHAR(1000),
   account_id    VARCHAR(100)
);
```

‘111,222,333’ カンマ区切りで
多対多関連を格納してしまう

解決策: 交差テーブルを作成する

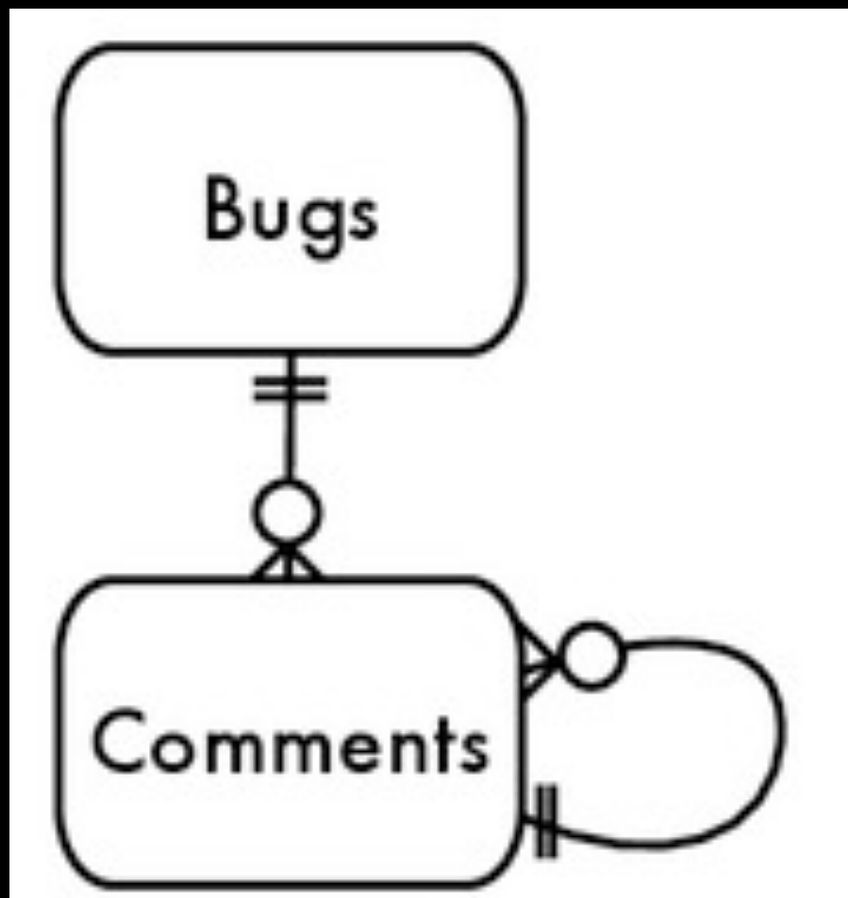
```
CREATE TABLE Contacts (
  product_id  BIGINT UNSIGNED NOT NULL,
  account_id  BIGINT UNSIGNED NOT NULL,
  PRIMARY KEY (product_id, account_id),
  FOREIGN KEY (product_id) REFERENCES Products(product_id),
  FOREIGN KEY (account_id) REFERENCES Accounts(account_id)
);
```



ナイーブツリー (素朴な木)

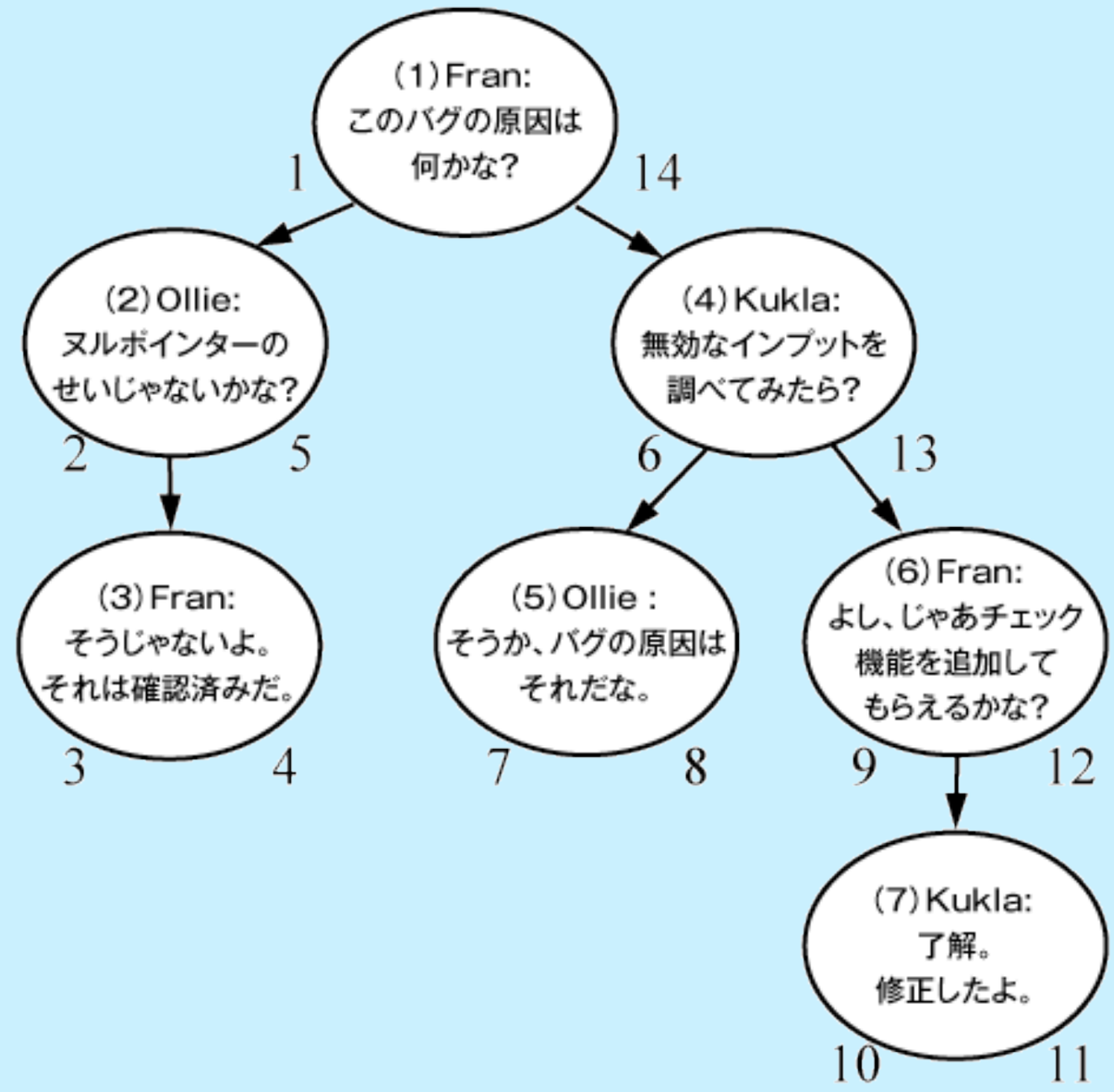
```
CREATE TABLE Comments (
    comment_id    SERIAL PRIMARY KEY,
    parent_id     BIGINT UNSIGNED,
    comment       TEXT NOT NULL,
);
```

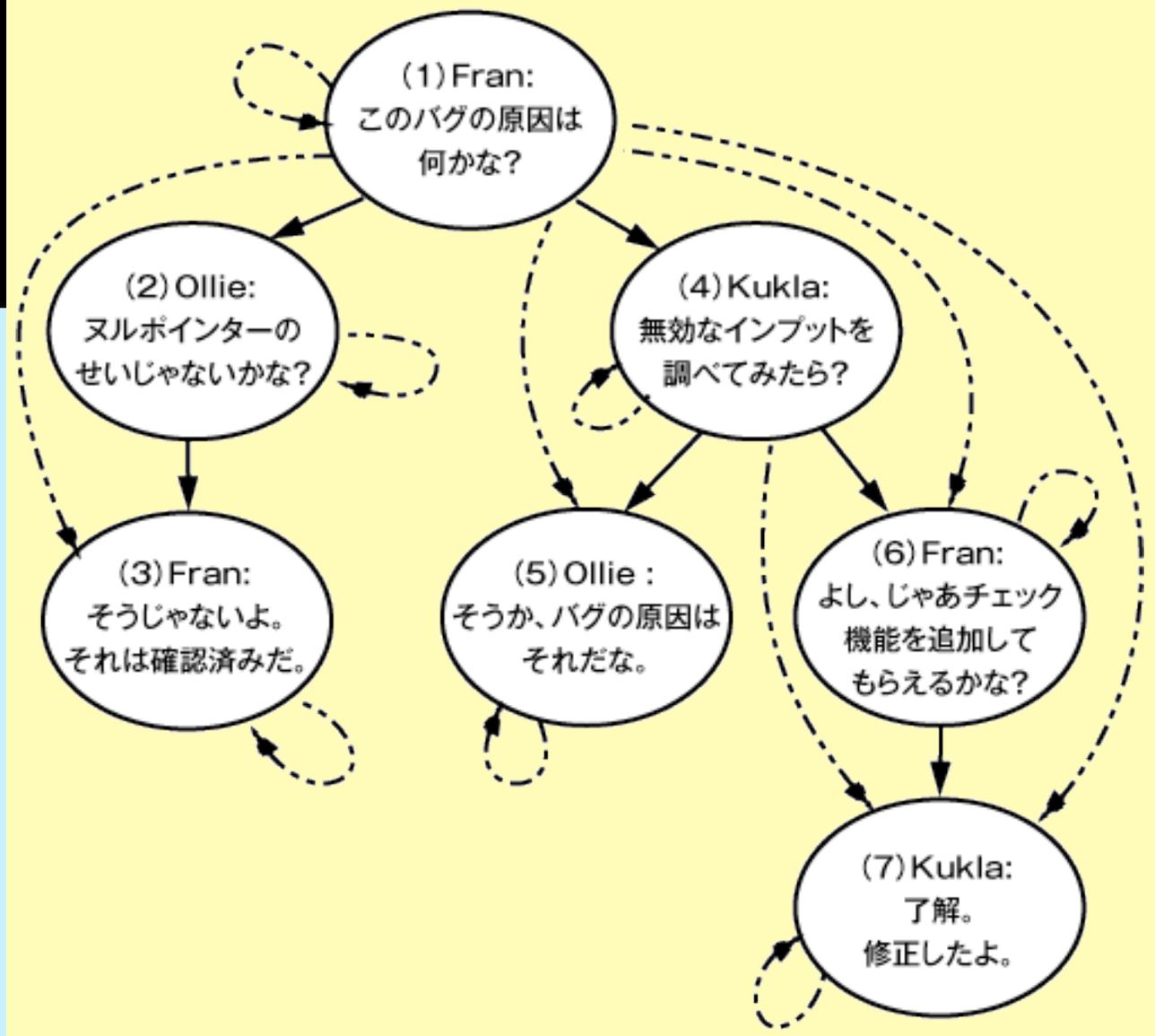
親idが入る



解決策: 代替ツリーモデルを使用する

2. 入れ子集合 NestedSet





3. 閉包テーブル Closure Table

IDリクワイアド (とりあえずID)

考え無しにとりあえず主キー
を“id”にしてしまう

```
CREATE TABLE BugsProducts (
  id          SERIAL PRIMARY KEY,
  bug_id      BIGINT UNSIGNED NOT NULL,
  product_id  BIGINT UNSIGNED NOT NULL,
  UNIQUE KEY  (bug_id, product_id),
  FOREIGN KEY (bug_id) REFERENCES Bugs(bug_id),
  FOREIGN KEY (product_id) REFERENCES Products(product_id)
);
```

解決策: 状況に応じて適切に調整する

わかりやすい列名 (id より bug_id)

```
SELECT * FROM Bugs INNER JOIN BugsProducts USING (bug_id);
```

同じ名前なら
USINGが使える

規約に縛られない
自然キーと複合キーの活用

考えた結果の“id”ならそれで良し

キーレスエントリ(外部キー嫌い)

```
CREATE TABLE Bugs (
  -- 他の列．．．
  reported_by         BIGINT UNSIGNED NOT NULL,
  status              VARCHAR(20) NOT NULL DEFAULT 'NEW',
  FOREIGN KEY (reported_by) REFERENCES Accounts(account_id),
  FOREIGN KEY (status) REFERENCES BugStatus(status)
);
```

外部キーを定義せず，
整合性をアプリで保つ
→アプリ以外からゴミデータが入る

解決策: 外部キー制約を宣言する

```
CREATE TABLE Bugs (
  -- 他の列. . .
  reported_by       BIGINT UNSIGNED NOT NULL,
  status            VARCHAR(20) NOT NULL DEFAULT 'NEW',
  FOREIGN KEY (reported_by) REFERENCES Accounts(account_id),
  FOREIGN KEY (status) REFERENCES BugStatus(status)
);
```

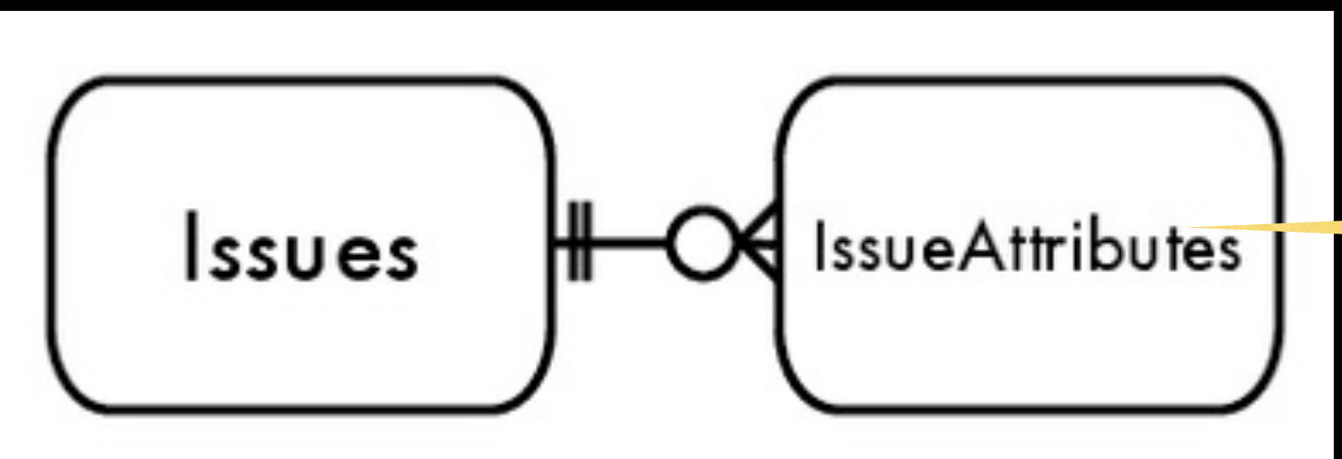
きちんと外部キーを定義する

EAV（エンティティ・アトリビュート・バリュー）

```
CREATE TABLE Issues (
  issue_id     SERIAL PRIMARY KEY
);

CREATE TABLE IssueAttributes (
  issue_id     BIGINT UNSIGNED NOT NULL,
  attr_name    VARCHAR(100) NOT NULL,
  attr_value   VARCHAR(100),
  PRIMARY KEY (issue_id, attr_name),
  FOREIGN KEY (issue_id) REFERENCES Issues(issue_id)
);
```

動的な項目を
格納したい



FK,名前,値
FK,名前,値
……

解決策: サブタイプのモデリングを行う

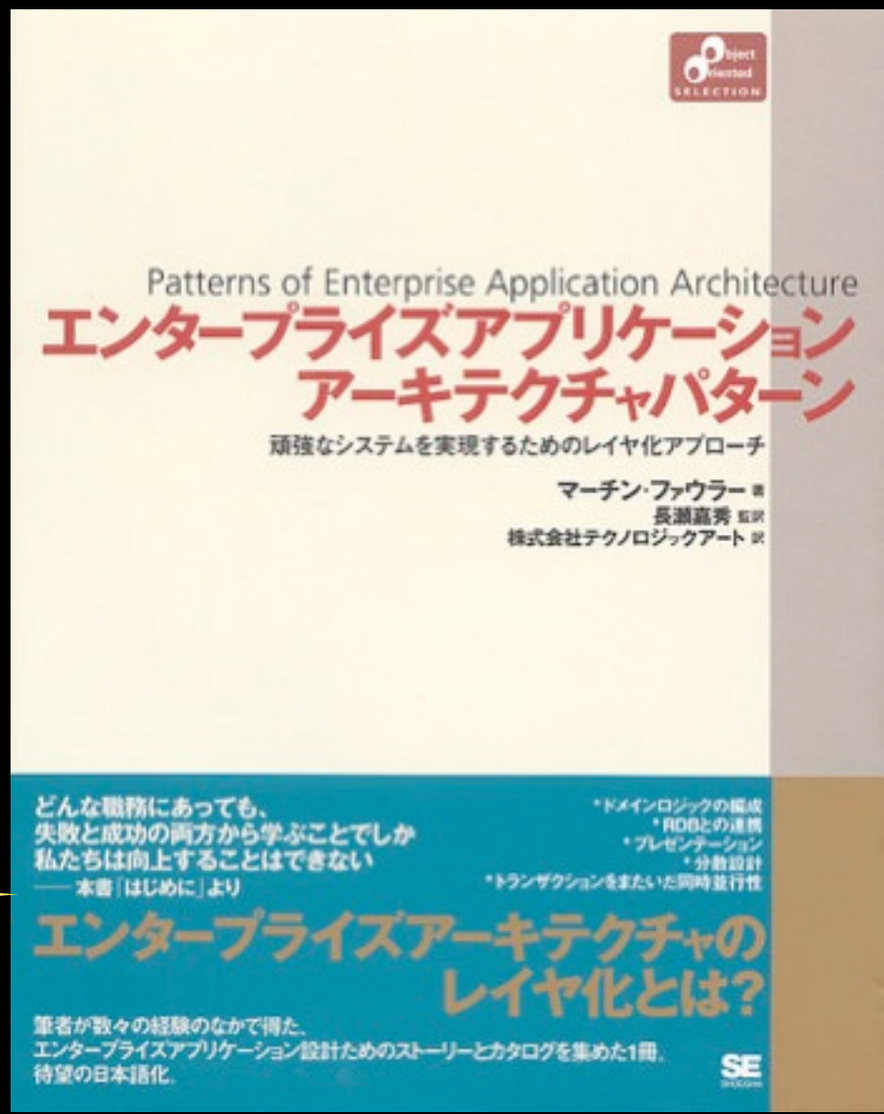
シングルテーブル継承

具象テーブル継承

クラステーブル継承

シリアライズ LOB

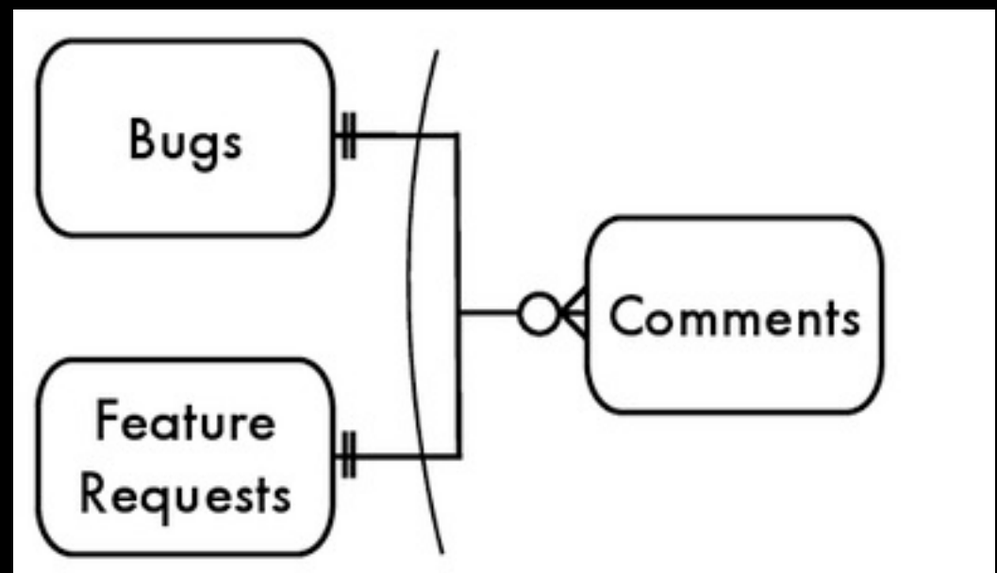
PofEAA 読むべし



ポリモーフィック関連

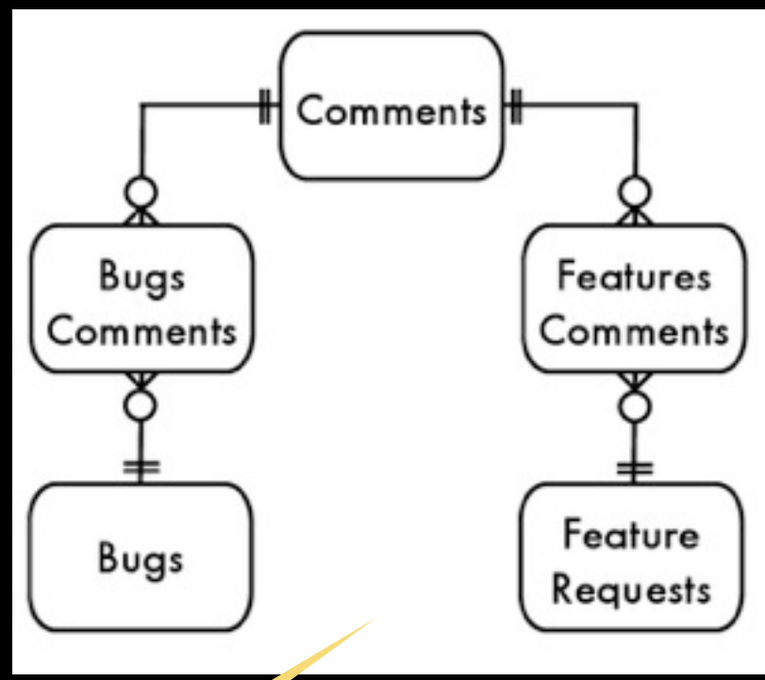
```
CREATE TABLE Comments (
  comment_id    SERIAL PRIMARY KEY,
  issue_type    VARCHAR(20),
  issue_id      BIGINT UNSIGNED NOT NULL,
  author        BIGINT UNSIGNED NOT NULL,
  comment_date  DATETIME,
  comment       TEXT,
  FOREIGN KEY   (author) REFERENCES Accounts(account_id)
);
```

'Bugs' または
'FeatureRequests' が入る



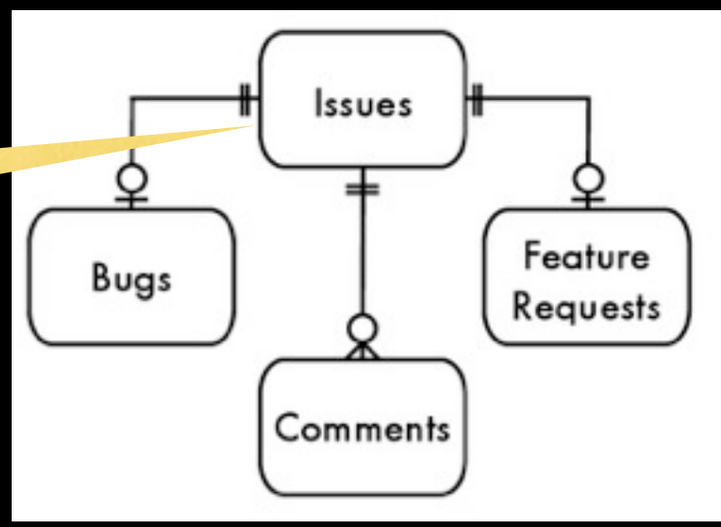
解決策:関連(リレーションシップ)を単純化する

```
CREATE TABLE BugsComments (
  issue_id    BIGINT UNSIGNED NOT NULL,
  comment_id  BIGINT UNSIGNED NOT NULL,
  PRIMARY KEY (issue_id, comment_id),
  FOREIGN KEY (issue_id) REFERENCES Bugs(issue_id),
  FOREIGN KEY (comment_id) REFERENCES Comments(comment_id)
);
CREATE TABLE FeaturesComments (
  issue_id    BIGINT UNSIGNED NOT NULL,
  comment_id  BIGINT UNSIGNED NOT NULL,
  PRIMARY KEY (issue_id, comment_id),
  FOREIGN KEY (issue_id) REFERENCES FeatureRequests(issue_id),
  FOREIGN KEY (comment_id) REFERENCES Comments(comment_id)
);
```



普通に交差テーブル

基底テーブルが必要なら
PofEAA の継承マッピング



マルチカラムアトリビュート（複数列属性）

```
CREATE TABLE Bugs (
  bug_id       SERIAL PRIMARY KEY,
  description  VARCHAR(1000),
  tag1         VARCHAR(20),
  tag2         VARCHAR(20),
  tag3         VARCHAR(20)
);
```

空いている列に値が入る

解決策: 従属テーブルを作成する

```
CREATE TABLE Tags (
  bug_id        BIGINT UNSIGNED NOT NULL,
  tag           VARCHAR(20),
  PRIMARY KEY   (bug_id, tag),
  FOREIGN KEY   (bug_id) REFERENCES Bugs(bug_id)
);
```

複数の列ではなく
複数の行に格納すること

```
SELECT * FROM Bugs
  INNER JOIN Tags AS t1 USING (bug_id)
  INNER JOIN Tags AS t2 USING (bug_id)
WHERE t1.tag = 'printing' AND t2.tag = 'performance';
```

メタデータトリブル（メタデータ大増殖）

```
CREATE TABLE Bugs_2008 ( . . . );
CREATE TABLE Bugs_2009 ( . . . );
CREATE TABLE Bugs_2010 ( . . . );
```

年が変わる毎にテーブル
や列を追加している

解決策: パーティショニングと正規化を行う

```
CREATE TABLE Bugs (
  bug_id          SERIAL PRIMARY KEY,
  -- 他の列. . .
  date_reported ĐATE
) PARTITION BY HASH ( YEAR(date_reported) )
PARTITIONS 4;
```

水平パーティショニング
(シャーディング)

1. 論理設計

2. 物理設計

3. クエリ

4. アプリケーション

物理設計のアンチパターン

9. ラウンディングエラー（丸め誤差）

10. サーティワンフレーバー（31のフレーバー）

11. ファントムファイル（幻のファイル）

否定意見多し

12. インデックスショットガン（闇雲インデックス）

ラウンディングエラー（丸め誤差）

少数値を表すために
浮動小数点数を使ってしまう

```
ALTER TABLE Bugs ADD COLUMN hours FLOAT;

ALTER TABLE Accounts ADD COLUMN hourly_rate FLOAT;


SELECT hourly_rate * 1000000000 FROM Accounts WHERE
account_id = 123;
```

結果: 599500000762.939

丸め誤差発生

解決策: NUMERIC データ型を使用する

精度(桁の総数)
スケール(小数点以下に格納できる桁数)

```
ALTER TABLE Bugs ADD COLUMN hours NUMERIC(9,2);

ALTER TABLE Accounts ADD COLUMN hourly_rate NUMERIC(9,2);


SELECT hourly_rate * 1000000000 FROM Accounts WHERE
account_id = 123;
```

結果: 59950000000

誤差は出ない

サーティワンフレーバー（31のフレーバー）

```
CREATE TABLE Bugs (
  -- 他の列．．．
  status    VARCHAR(20) CHECK (status IN
                ('NEW','IN PROGRESS','FIXED'))
);
```

許可する値を列定義で指定する
→ 許可値を取得できない

解決策: 限定する値をデータで指定する

```
CREATE TABLE BugStatus (
   status  VARCHAR(20) PRIMARY KEY
);

INSERT INTO BugStatus (status) VALUES
   ('NEW'), ('IN PROGRESS'), ('FIXED');
```

```
CREATE TABLE Bugs (
   -- 他の列. . .
   status  VARCHAR(20),
   FOREIGN KEY (status) REFERENCES BugStatus(status)
      ON UPDATE CASCADE
);
```

外部キーで整合性保証

ファントムファイル（幻のファイル）

```
CREATE TABLE Screenshots (
  bug_id          BIGINT UNSIGNED NOT NULL,
  image_id        BIGINT UNSIGNED NOT NULL,
  screenshot_path   VARCHAR(100),
  caption           VARCHAR(100),
  PRIMARY KEY (bug_id, image_id),
  FOREIGN KEY (bug_id) REFERENCES Bugs(bug_id)
);
```

物理ファイルの使用を必須と思い込む

解決策: 必要に応じてBLOB型を採用する

```
CREATE TABLE Screenshots (
  bug_id           BIGINT UNSIGNED NOT NULL,
  image_id         BIGINT UNSIGNED NOT NULL,
  screenshot_image  BLOB,
  caption          VARCHAR(100),
  PRIMARY KEY (bug_id, image_id),
  FOREIGN KEY (bug_id) REFERENCES Bugs(bug_id)
);
```

整合性、トランザクション、
ロック、権限管理がメリット

インデックスショットガン（闇雲インデックス）

```
CREATE TABLE Bugs (
  bug_id          SERIAL PRIMARY KEY,
  date_reported ĐATE NOT NULL,
  summary         VARCHAR(80) NOT NULL,
  status          VARCHAR(10) NOT NULL,
  hours           NUMERIC(9,2),
  INĐEX (bug_id),
  INĐEX (summary),
  INĐEX (hours),
  INĐEX (bug_id, date_reported, status)
);
```

闇雲にインデックスを定義してしまう

解決策:「MENTOR」の原則に基づいて効果的なインデックス管理を行う

Measure（測定）
Explain（解析）
Nominate（指名）
Test（テスト）
Optimize（最適化）
Rebuild（再構築）

1. 論理設計
2. 物理設計
3. クエリ
4. アプリケーション

クエリのアンチパターン

13. フィア・オブ・ジ・アンノウン（恐怖のunknown）

14. アンビギュアスグループ（曖昧なグループ）

15. ランダムセレクション

様々な解決策

16. プアマンズ・サーチエンジン（貧者のサーチエンジン）

17. スパゲッティクエリ

18. インプリシットカラム（暗黙の列）

フィア・オブ・ジ・アンノウン（恐怖のunknown）

```sql
CREATE TABLE Bugs (
  bug_id              SERIAL PRIMARY KEY,
  -- 他の列．．．
  assigned_to         BIGINT UNSIGNED NOT NULL,
  hours               NUMERIC(9,2) NOT NULL,
  FOREIGN KEY (assigned_to) REFERENCES Accounts(account_id)
);
```

NULLを嫌う

```sql
INSERT INTO Bugs (assigned_to, hours) VALUES (-1, -1);

SELECT AVG( hours ) AS average_hours_per_bug FROM Bugs
WHERE hours <> -1;
```

NULL代わりの値を使い、
おかしなことに

解決策: NULL を一意な値として使う

```
SELECT * FROM Bugs WHERE assigned_to IS NULL;
SELECT * FROM Bugs WHERE assigned_to IS NOT NULL;
```

IS [NOT] NULL 述語を使う

```
SELECT * FROM Bugs WHERE assigned_to IS NULL OR assigned_to <> 1;

SELECT * FROM Bugs WHERE assigned_to IS ÐISTINCT FROM 1;
```

二つのクエリは等価。IS ÐISTINCT FROM なら
プリペアドステートメントにも使える!

アンビギュアスグループ（曖昧なグループ）

```
SELECT p.product_id,
       MAX(b.date_reported) AS latest,
       b.bug_id
FROM Bugs b INNER JOIN BugsProducts p USING (bug_id)
GROUP BY p.product_id;
```

MAX(date_reported)のbug_idが
返るとは限らない

解決策: 曖昧でない列を使う

```
SELECT m.product_id, m.latest,
  MAX(b1.bug_id) AS latest_bug_id
FROM Bugs b1 INNER JOIN
  (
   SELECT product_id, MAX(date_reported) AS latest
   FROM Bugs b2 INNER JOIN BugsProducts USING (bug_id)
   GROUP BY product_id
  ) m ON b1.date_reported = m.latest
GROUP BY m.product_id, m.latest;
```

相関サブクエリ
導出テーブル
JOIN の使用
などの手段がある
(例は導出テーブル)

ランダムセレクション

```
SELECT * FROM Bugs ORDER BY RAND() LIMIT 1;
```

非決定性を持つ式によって
ソートを行ってしまう

解決策: 特定の順番に依存しない

```
SELECT b1.*
FROM Bugs AS b1
INNER JOIN (
  SELECT CEIL(
    RAND() * (SELECT MAX(bug_id) FROM Bugs)
  ) AS bug_id
) AS b2 ON b1.bug_id >= b2.bug_id
ORDER BY b1.bug_id
LIMIT 1;
```

様々なテクニック
(例は欠番の穴の後にある番号を取得するSQL)

プアマンズ・サーチエンジン（貧者のサーチエンジン）

```
SELECT * FROM Bugs
WHERE description LIKE '%one%';
```

あいまい検索に
パターンマッチ述語を使用してしまう

解決策: 適切なツールを使用する

1. ベンダー拡張の全文検索機能

2. サードパーティーの全文検索エンジン

3. 転置インデックスの自作

SQL だけでがんばらない

スパゲッティクエリ

```
SELECT p.product_id,
  COUNT(f.bug_id) AS count_fixed,
  COUNT(o.bug_id) AS count_open
FROM BugsProducts p
INNER JOIN Bugs f ON p.bug_id = f.bug_id AND f.status = 'FIXED'
INNER JOIN BugsProducts p2 USING (product_id)
INNER JOIN Bugs o ON p2.bug_id = o.bug_id AND o.status = 'OPEN'
WHERE p.product_id = 1
GROUP BY p.product_id;
```

複雑な問題をワンステップで解決しようとする

解決策: 分割統治を行う

```
(SELECT p.product_id, 'FIXED' AS status, COUNT(f.bug_id) AS bug_count
 FROM BugsProducts p
 LEFT OUTER JOIN Bugs f ON p.bug_id = f.bug_id AND f.status = 'FIXED'
 WHERE p.product_id = 1
 GROUP BY p.product_id)

UNION ALL

(SELECT p.product_id, 'OPEN' AS status, COUNT(o.bug_id) AS bug_count
 FROM BugsProducts p
 LEFT OUTER JOIN Bugs o ON p.bug_id = o.bug_id AND o.status = 'OPEN'
 WHERE p.product_id = 1
 GROUP BY p.product_id)

ORDER BY bug_count DESC;
```

ワンステップずつ解く
必要なら UNION 等

インプリシットカラム（暗黙の列）

```
SELECT * FROM Bugs;
```

タイプ数を減らしたい欲望に負ける

→不要な列まで取得したり、定義変更がバグの元になる

解決策: 列名を明示的に指定する

```
SELECT bug_id, date_reported, summary, description, resolution,
  reported_by, assigned_to, verified_by, status, priority, hours
FROM Bugs;

INSERT INTO Accounts (account_name, first_name, last_name, email,
  password_hash, portrait_image, hourly_rate)
VALUES ('bkarwin', 'Bill', 'Karwin', 'bill@example.com',
SHA2('xyzzy', 256), NULL, 49.95);
```

誤りの防止
（フェイルファースト）
必要な列だけ指定

1. 論理設計
2. 物理設計
3. クエリ
4. アプリケーション

アプリケーションのアンチパターン

19. リーダブルパスワード（読み取り可能パスワード）

20. SQLインジェクション

21. シュードキー・ニートフリーク（疑似キー潔癖症）

22. シー・ノー・エビル（臭いものに蓋）

23. ディプロマティック・イミュニティ（外交特権）

24. マジックビーンズ（魔法の豆）

25. 砂の城

奥野さん書き下ろし！

リーダブルパスワード（読み取り可能パスワード）

```
CREATE TABLE Accounts (
  account_id     SERIAL PRIMARY KEY,
  account_name   VARCHAR(20) NOT NULL,
  email          VARCHAR(100) NOT NULL,
  password       VARCHAR(30) NOT NULL
);

INSERT INTO Accounts (account_id, account_name, email, password)
VALUES (123, 'billkarwin', 'bill@example.com', 'xyzzy');
```

パスワードを平文で格納してしまう

解決策: ソルトをつけてパスワードハッシュを格納する

```php
<?php
$stmt = $pdo->query(
    "SELECT salt
     FROM Accounts
     WHERE account_name = 'bill'");

$row = $stmt->fetch();
$salt = $row[0];

$hash = hash('sha256', $password . $salt);
```

ソルトとハッシュ

```php
$stmt = $pdo->query("
  SELECT (password_hash = '$hash') AS password_matches
  FROM Accounts AS a
  WHERE a.account_name = 'bill'");

$row = $stmt->fetch();
if ($row === false) {
  // アカウント'bill'は存在しない
} else {
  $password_matches = $row[0];
  if (!$password_matches) {
    // パスワードが間違っている
  }
}
```

SQLインジェクション

```php
<?php
$project_name = $_REQUEST["name"];
$sql = "SELECT * FROM Projects WHERE project_name = '$project_name'";
```

↓

```
http://bugs.example.com/project/view.php?name=O'Hare
```

↓

```
SELECT * FROM Projects WHERE project_name = 'O'Hare';
```

未検証の入力をクエリにつなげて実行してしまう

解決策: 誰も信用してはならない

あえて言うなら、第一に
プリペアドステートメントを
使うべし

```php
<?php
$sql = "UPĐATE Accounts
  SET password_hash = SHA2(?, 256)
  WHERE account_id = ?";
$stmt = $pdo->prepare($sql);
$stmt->bindValue(1, $_REQUEST["password"], PĐO::PARAM_STR);
$stmt->bindValue(2, $_REQUEST["userid"], PĐO::PARAM_INT);
$stmt->execute();
```

シュードキー・ニートフリーク（疑似キー潔癖症）

```
bug_id   status   product_name
-------------------------------
       1   OPEN     Open RoundFile
       2   FIXED    ReConsider
       4   OPEN     ReConsider
```

```
UPDATE Bugs SET bug_id = 3 WHERE bug_id = 4;
```

気になるので欠番の隙間を埋めてしまう

解決策: 疑似キーの欠番は埋めない

解決すべき大きな問題が残っています。それは、疑似キーの欠番を埋めろという上司からの要求を、どうやって断るのかという問題です。これは技術ではなく、コミュニケーションの問題です

シー・ノー・エビル（臭いものに蓋）

```php
<?php
$pdo = new PĐO("mysql:dbname=test;host=db.example.com",
               "dbuser", "dbpassword");
$sql = "SELECT bug_id, summary, date_reported FROM Bugs
         WHERE assigned_to = ? ANĐ status = ?";
$stmt = $pdo->prepare($sql);
$stmt->execute(array(1, "OPEN"));
$bug = $stmt->fetch();
```

戻り値や例外を無視する

解決策: エラーから優雅に回復する

```
    <?php
    try {
        $pdo = new PÐO("mysql:dbname=test;host=localhost",
            "dbuser", "dbpassword");
①   } catch (PÐOException $e) {
        report_error($e->getMessage());
        return;
    }
    $sql = "SELECT bug_id, summary, date_reported FROM Bugs
        WHERE assigned_to = ? AND status = ?";
②   if (($stmt = $pdo->prepare($sql)) === false) {
        $error = $pdo->errorInfo();
        report_error($error[2]);
        return;
    }
③   if ($stmt->execute(array(1, "OPEN")) === false) {
        $error = $stmt->errorInfo();
        report_error($error[2]);
        return;
    }
④   if (($bug = $stmt->fetch()) === false) {
        $error = $stmt->errorInfo();
        report_error($error[2]);
        return;
    }
```

戻り値や例外をチェック

ディプロマティック・イミュニティ（外交特権）

文書化

バージョン管理

テスティング

DBだけは特別扱いし、
やらなくて良いと考えがち

解決策: 包括的に品質問題に取り組む

文書化

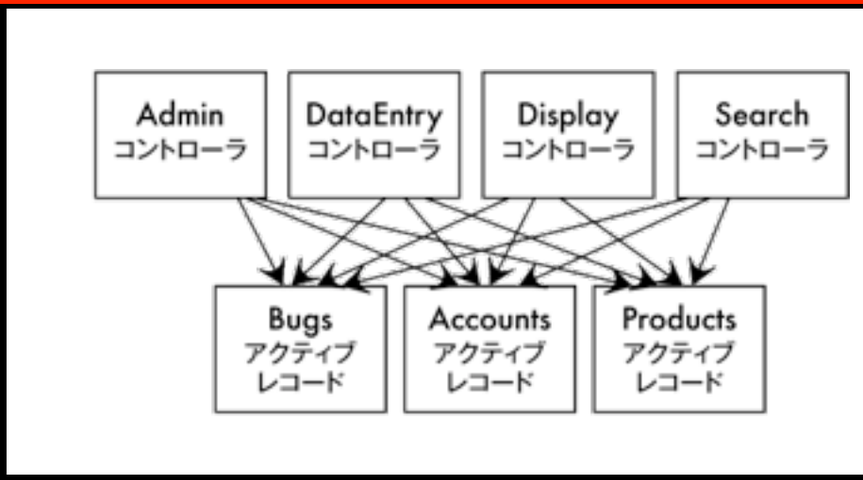
バージョン管理

テスティング

例外は無い!!
テーブル定義のバージョン管理には
migration が便利

マジックビーンズ（魔法の豆）

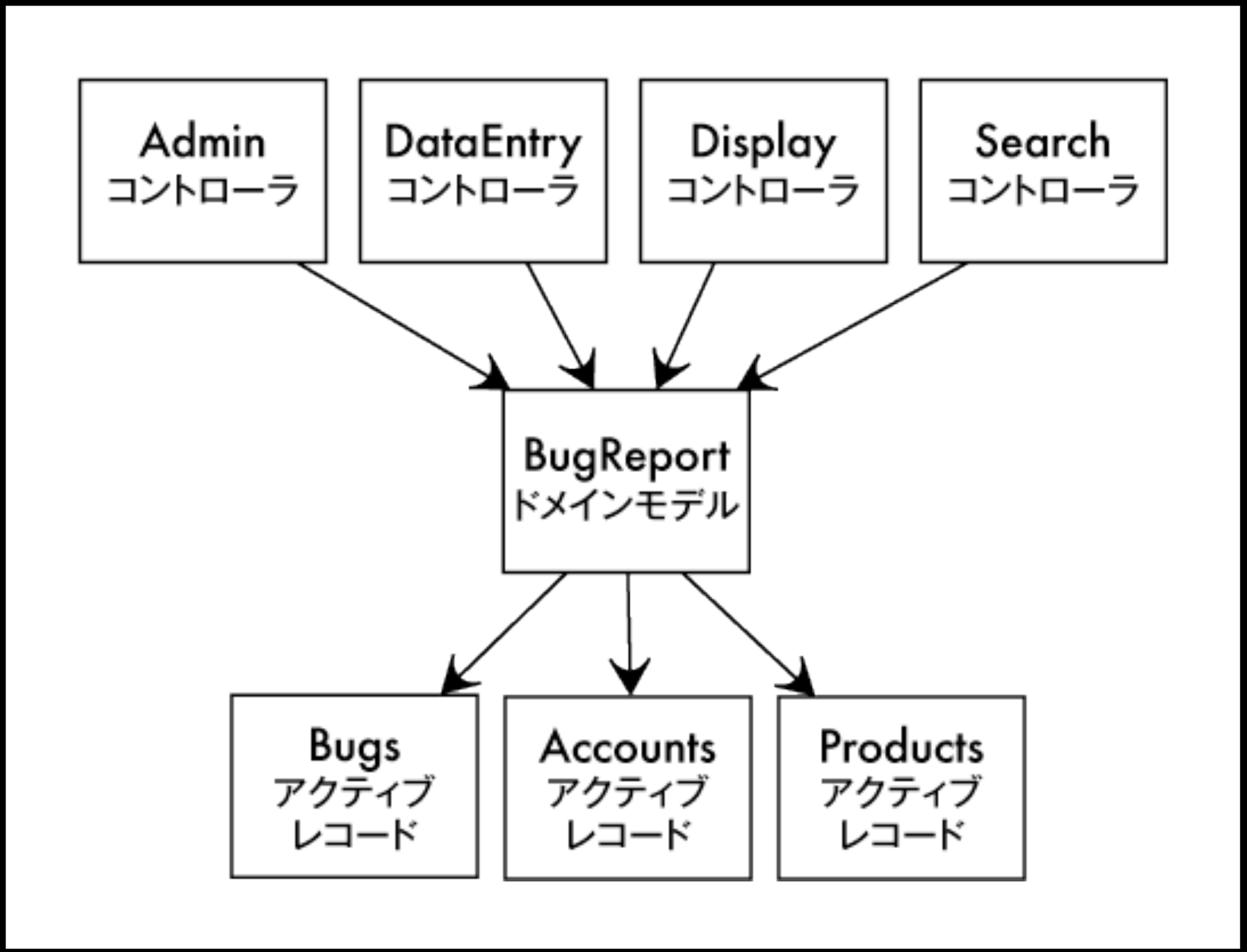
```php
<?php
class CustomBugs extends BaseBugs
{
  public function assignUser(Accounts $a)
  {
    $this->assigned_to = $a->account_id;
    $this->save();
    mail($a->email, "バグ担当に任命されました",
      "あなたはバグ #{$this->bug_id} の修正担当に任命されました。");
  }
}
```




```php
$bugsTable = Doctrine_Core::getTable('Bugs');
$bugsTable->find(1234);
$bug->assigned_to = $user->account_id;
$bug->save();
```

モデルがアクティブレコードそのものなので、ドメインロジックを迂回できる

解決策: モデルがアクティブレコードを「持つ」ようにする



砂の城 (奥野さん書き下ろし)

「データサイズが一ヶ月で3倍になった」

「データベースへの更新でデッドロックが起きる。製品のバグじゃないか？」

「問題を解決するためにベンダーが要求しているデータは本番環境の負荷が高すぎるので採取できない」

「最近マシンをアップグレードした。だから性能の問題とは無縁だろう」

想定不足

解決策: インシデントを想定し、備える

ベンチマーク

テスト環境の構築

例外処理

バックアップ

高可用性

ディザスタリカバリ

運用ポリシーの策定

Agenda

1. アンチパターンとは
2. 本書のダイジェスト
3. おわりに

「悪いこと」をまとめた意義

この本の素晴らしいところは、よく見る「悪い」方法を「悪いこと」としてまとめてくれたことです。

http://bleis-tift.hatenablog.com/entry/2013/02/14/SQLアンチパターン

アンチパターン名で議論できるようになる

「”**マルチカラムアトリビュート**”とか 10 年前に通ったわー」

「あーはいはい”**インデックスショットガン**”乙」
Explain の結果も見ないでインデックス貼りまくる奴いるよねーーー

http://yoshiori.github.com/blog/2013/02/10/sql-antipatterns/

アンチパターンを共有しよう!

この問題！

進研ゼミでやったところだ！

http://yojik.hatenablog.jp/entry/2013/02/13/235729

ご清聴ありがとうございました

本書のハッシュタグは
#sqlap です!
ぜひ読んでみてください