Как испортить данные и как этого избежать. Введение в проблематику конкурентного доступа к данным.

Игра в футбол.

1. Играют две команды (зеленая и красная)
2. Команды забивают Голы
3. В Игре регистрируется гол и записывается команда, забившая последней

Начальная реализация Игры. Классы.

```
class Play {
  private readonly id: string;

  private goalsCount: number = 0;

  private lastScoredTeam: Team | null = null;

  public scoreGoal(team: Team) {
    this.lastScoredTeam = team;
    this.goalsCount++;
  }
}

class Goal {
  constructor(private readonly id: string, private readonly team: Team) {}
}
```

Начальная реализация Игры. База данных

```
CREATE TABLE `play` (
  `id` varchar(255),
  `goals_count` int(11) NOT NULL,
  `last_scored_team` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci


CREATE TABLE `goal` (
  `id` varchar(255),
  `team` varchar(255),
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
```

Начальная реализация Игры. Принцип сохранения данных.

```
BEGIN;

try {

UPDATE play SET values = ? WHERE id = ?;
INSERT INTO goal SET values = ?;

COMMIT;

} catch () {
  ROLLBACK;
}
```

Начальная реализация Игры. Запросы.

- POST http://play-server:3003/green
- POST http://play-server:3003/red

```
SELECT COUNT(*) as goals FROM goal;
+-------+
| goals |
+-------+
|   169 |
+-------+

SELECT * FROM play;
+--------------+--------------------+
| goals_count | last_scored_team |
+--------------+--------------------+
|         164 | green              |
+--------------+--------------------+
```

Где же голы!?

- Оба "запроса" получают копию текущих данных.
- Каждый "запрос" изменяет свою копию данных в соответствии с собственной бизнес-логикой.
- Каждый "запрос" обновляет текущие данные в базе данных.
- В базе данных остаются данные "запроса", выполнившего обновление последним.
- Данные испорчены.

Последующая реализация Игры. Классы

```
export class Play {
  private readonly dataVersion: number;

  private readonly id: string;

  private goalsCount: number = 0;

  private lastScoredTeam: Teams | null = null;

  public scoreGoal(team: Teams) {
    this.lastScoredTeam = team;
    this.goalsCount++;
  }
}

export class Goal {
  constructor(private readonly id: string, private readonly team: Team) {}
}
```

Последующая реализация Игры. База данных.

```
CREATE TABLE `play` (
  `data_version` int(11) DEFAULT 1,
  `id` varchar(255),
  `goals_count` int(11) NOT NULL,
  `last_scored_team` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci


CREATE TABLE `goal` (
  `id` varchar(255),
  `team` varchar(255),
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
```

Последующая реализация Игры. Принцип сохранения данных.

```
BEGIN;

try {

UPDATE play SET values = ?, data_version = data_version + 1
            WHERE id = ? AND data_version = ?;

// Если число обновленных строк === 0, выбросить Исключение

INSERT INTO goal SET values = ?;

COMMIT;

} catch () {
  ROLLBACK;
}
```

Последующая реализация Игры. Запросы

- POST http://play-server:3003/green
- POST http://play-server:3003/red

173 2xx responses, **3 non 2xx** responses

```
SELECT COUNT(*) as goals FROM goal;
+-------+
| goals |
+-------+
|   283 |
+-------+

SELECT * FROM play;
+--------------+--------------------+---------------+
| goals_count | last_scored_team | data_version |
+--------------+--------------------+---------------+
|         283 | green              |           284 |
+--------------+--------------------+---------------+
```