cafeMuji 注文管理システム データベース設計書

1. データベース概要

1.1 データベースエンジン

- **開発環境**: SQLite 3.x
- **本番環境**: SQLite（PostgreSQL推奨）
- **ORM**: Django ORM

1.2 設計方針

- **正規化**: 第3正規形を基本とする
- **拡張性**: 将来の機能追加に対応できる柔軟な設計
- **パフォーマンス**: 適切なインデックス設計
- **整合性**: 外部キー制約によるデータ整合性の確保

2. ER図

2.1 エンティティ関係図

```
[FoodOrder] ----< [注文グループ] >---- [Order]
     |                                    |
     |                                    |
[フード注文]                          [アイスクリーム注文]
     |                                    |
     |                                    |
[クリップ情報]                        [フレーバー情報]
     |                                    |
     |                                    |
[注文状態]                            [注文状態]

[ShavedIceOrder] ----< [注文グループ] >---- [共通属性]
     |                                    |
     |                                    |
[かき氷注文]                          [時刻管理]
     |                                    |
     |                                    |
[フレーバー]                          [完了状態]
```

3. テーブル設計詳細

3.1 フード注文テーブル（FoodOrder）

テーブル概要

フード注文（からあげ丼、ルーロー飯）の全情報を管理するテーブル

フィールド定義

フィールド名	データ型	制約	説明
id	AutoField	Primary Key	自動採番ID
menu	CharField(20)	Not Null	メニュー名
quantity	PositiveIntegerField	Default: 1	注文数量
eat_in	BooleanField	Default: True	店内/テイクアウト
clip_color	CharField(10)	Not Null	クリップ色
clip_number	IntegerField	Not Null	クリップ番号
group_id	CharField(50)	Not Null	グループID
status	CharField(10)	Default: ‘ok’	注文状態
is_completed	BooleanField	Default: False	完了フラグ
timestamp	DateTimeField	Auto Now Add	受注時刻
completed_at	DateTimeField	Nullable	完了時刻
note	TextField	Nullable	備考

インデックス設計

```
-- 主要検索用インデックス
CREATE INDEX idx_food_order_group_id ON food_foodorder(group_id);
CREATE INDEX idx_food_order_status ON food_foodorder(status);
CREATE INDEX idx_food_order_completed ON food_foodorder(is_completed);
CREATE INDEX idx_food_order_timestamp ON food_foodorder(timestamp);

-- 複合インデックス
CREATE INDEX idx_food_order_clip ON food_foodorder(clip_color, clip_number);
```

制約

- `quantity` > 0
- `clip_number` > 0
- `status` IN (‘ok’, ‘stop’)

3.2 アイスクリーム注文テーブル（Order）

テーブル概要

アイスクリーム注文の全情報を管理するテーブル

フィールド定義

フィールド名	データ型	制約	説明
id	AutoField	Primary Key	自動採番ID

フィールド名	データ型	制約	説明
group_id	CharField(20)	Not Null	グループID
size	CharField(2)	Choices	サイズ（S/W）
container	CharField(10)	Choices	容器（cup/cone）
flavor1	CharField(50)	Choices	フレーバー1
flavor2	CharField(50)	Nullable	フレーバー2
is_completed	BooleanField	Default: False	完了フラグ
timestamp	DateTimeField	Auto Now Add	受注時刻
status	CharField(10)	Default: ‘ok’	注文状態
clip_color	CharField(10)	Choices	クリップ色
clip_number	IntegerField	Not Null	クリップ番号
completed_at	DateTimeField	Nullable	完了時刻
is_auto_stopped	BooleanField	Default: False	自動STOP
note	TextField	Nullable	備考
is_pudding	BooleanField	Default: False	アフォガードプリン

選択肢定義

```
# サイズ選択肢
SIZE_CHOICES = [
    ('S', 'シングル'),
    ('W', 'ダブル'),
]

# 容器選択肢
CONTAINER_CHOICES = [
    ('cup', 'カップ'),
    ('cone', 'コーン'),
]

# フレーバー選択肢
FLAVOR_CHOICES = [
    ('jersey', 'ジャージー牛乳'),
    ('ocha', 'お茶'),
    ('mango', 'マンゴー'),
    ('mint', 'チョコミント'),
    ('caramel', 'キャラメル'),
    ('strawberry', 'いちご'),
    ('tachibana', 'たちばな'),
    ('idashio', '井田塩'),
    ('cassis', 'カシス'),
    ('chocolate', 'ショコラ'),
    ('coffee', 'コーヒー'),
    ('lemon', 'レモン'),
]

# 注文状態選択肢
ORDER_STATUS_CHOICES = [
    ('ok', 'アイス作成OK'),
    ('stop', 'アイス作成STOP'),
    ('hold', '保留（非表示待ち）'),
]

# クリップ色選択肢
CLIP_COLOR_CHOICES = [
    ('yellow', '黄色'),
```

```
    ('white', '白色'),
]
```

インデックス設計

```
-- 主要検索用インデックス
CREATE INDEX idx_ice_order_group_id ON ice_order(group_id);
CREATE INDEX idx_ice_order_status ON ice_order(status);
CREATE INDEX idx_ice_order_completed ON ice_order(is_completed);
CREATE INDEX idx_ice_order_timestamp ON ice_order(timestamp);

-- 複合インデックス
CREATE INDEX idx_ice_order_clip ON ice_order(clip_color, clip_number);
CREATE INDEX idx_ice_order_flavor ON ice_order(flavor1, flavor2);
```

3.3 かき氷注文テーブル（ShavedIceOrder）

テーブル概要

かき氷注文の全情報を管理するテーブル

フィールド定義

フィールド名	データ型	制約	説明
id	AutoField	Primary Key	自動採番ID
flavor	CharField(50)	Choices	フレーバー
group_id	CharField(50)	Default: ’’	グループID
is_completed	BooleanField	Default: False	完了フラグ
clip_color	CharField(10)	Default: ‘white’	クリップ色
clip_number	IntegerField	Default: 0	クリップ番号
timestamp	DateTimeField	Auto Now Add	受注時刻
completed_at	DateTimeField	Nullable	完了時刻
status	CharField(10)	Default: ‘ok’	注文状態
is_auto_stopped	BooleanField	Default: False	自動STOP
note	TextField	Nullable	備考

選択肢定義

```
# フレーバー選択肢
FLAVOR_CHOICES = [
    ('抹茶', '抹茶'),
    ('いちご', 'いちご'),
    ('ゆず', 'ゆず'),
]

# 注文状態選択肢
ORDER_STATUS_CHOICES = [
    ('ok', 'アイス作成OK'),
    ('stop', 'アイス作成STOP'),
    ('hold', '保留（非表示待ち）'),
```

```
]

# クリップ色選択肢
CLIP_COLOR_CHOICES = [
    ('yellow', '黄色'),
    ('white', '白色'),
]
```

4. データ整合性

4.1 外部キー制約

現在の設計では、各テーブルは独立しており、外部キー制約は設定されていません。これは以下の理由によります：

- **柔軟性**: 各注文タイプを独立して管理
- **拡張性**: 将来の機能追加に対応
- **パフォーマンス**: 結合クエリの回避

4.2 ビジネスルール

```
# 注文状態の管理
def update_order_status(order, new_status):
    if new_status == 'completed':
        order.is_completed = True
        order.completed_at = timezone.now()
    order.status = new_status
    order.save()

# クリップ番号の重複チェック
def check_clip_availability(color, number):
    existing_orders = FoodOrder.objects.filter(
        clip_color=color,
        clip_number=number,
        is_completed=False
    )
    return existing_orders.count() == 0
```

5. データ移行

5.1 マイグレーションファイル

各アプリケーションで独立したマイグレーションファイルを管理：

- `food/migrations/`: フード注文関連のマイグレーション
- `ice/migrations/`: アイスクリーム注文関連のマイグレーション
- `shavedice/migrations/`: かき氷注文関連のマイグレーション

5.2 マイグレーション履歴

```
# マイグレーションの確認
python manage.py showmigrations

# マイグレーションの実行
```

```
python manage.py migrate

# マイグレーションファイルの作成
python manage.py makemigrations
```

6. バックアップ・復旧

6.1 バックアップ戦略

```
# データベースのバックアップ
python manage.py dumpdata > backup.json

# 特定アプリのバックアップ
python manage.py dumpdata food > food_backup.json
python manage.py dumpdata ice > ice_backup.json
python manage.py dumpdata shavedice > shavedice_backup.json
```

6.2 復旧手順

```
# データベースの復旧
python manage.py loaddata backup.json

# 特定アプリの復旧
python manage.py loaddata food_backup.json
```

7. パフォーマンス最適化

7.1 クエリ最適化

```
# 効率的なクエリ例
def get_active_orders():
    return FoodOrder.objects.filter(
        is_completed=False
    ).select_related().order_by('timestamp')

def get_orders_by_clip(color, number):
    return FoodOrder.objects.filter(
        clip_color=color,
        clip_number=number,
        is_completed=False
    ).first()
```

7.2 インデックス戦略

- **検索頻度の高いフィールド**: プライマリキー、グループID、状態
- **結合に使用されるフィールド**: クリップ情報、完了フラグ
- **ソートに使用されるフィールド**: タイムスタンプ

8. 監視・ログ

8.1 データベース監視

```
# 注文数の監視
def get_order_statistics():
    total_orders = FoodOrder.objects.count()
    completed_orders = FoodOrder.objects.filter(is_completed=True).count()
```

```
    pending_orders = total_orders - completed_orders

    return {
        'total': total_orders,
        'completed': completed_orders,
        'pending': pending_orders
    }
```

8.2 ログ設定

```
# settings.py
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'file': {
            'level': 'INFO',
            'class': 'logging.FileHandler',
            'filename': 'debug.log',
        },
    },
    'loggers': {
        'django.db.backends': {
            'handlers': ['file'],
            'level': 'INFO',
        },
    },
}
```

作成日: 2025年8月 **作成者**: 村岡 優次郎 **バージョン**: 1.0