

# DB와 ORM에 관한 7가지 고민

—

김찬우

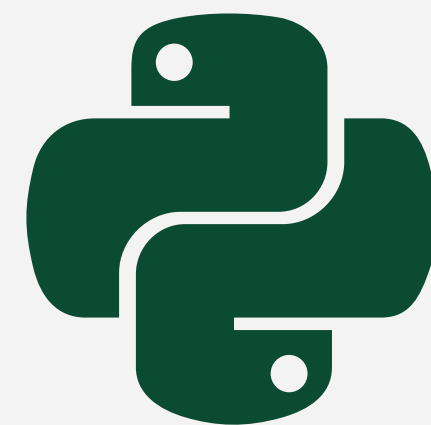
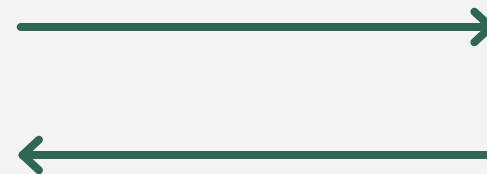
Internal Products Developer, **Toss**

# DB와 ORM에 관한 ≠ 6가지 고민

—

김찬우

Internal Products Developer, **Toss**



# Notes

- 1 슬라이드 주소 : [bit.ly/devdjango-2018-ed](http://bit.ly/devdjango-2018-ed)
- 2 특별한 언급이 없는 한, MySQL 환경을 기준으로 합니다.

Django at ***Toss***

django

Django at **Toss**



Django at **Toss**

***Toss***



**Viva Manager**

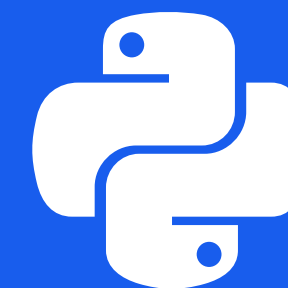
**toss internal**

# Django at **Toss**

**Toss**



**Viva Manager**  
**toss internal**





자주 묻는 질문

송금

이용방법

보안

거래내역 조회

신용등급 조회

카드조회서비스

비대면 계좌 개설

부동산 소액투자

P2P 분산투자

펀드 소액투자

해외 주식투자

기타

입금확인증은 어디에서 발급 받을 수 있나요?



은행의 자동이체 알림 메시지를 받았어요



토스 송금/결제 한도는 어떻게 되나요?



#### 1. 송금

- 일반 회원 : 일 50만원
- 토스 신한금융투자 계좌, 토스 KEB하나은행 계좌 개설 회원 : 일 200만원
- 미성년자 회원 : 일 30만원, 월 최대 100만원

채우기/옮기기와 송금 한도는 별도로 계산됩니다. 옮기기 횟수는 1일 30회까지 입니다.

#### 2. 결제

- 일반 회원 : 일 200만원
- 미성년자 회원 : 일 30만원, 월 최대 100만원

단, 일부 가맹점은 더 낮은 한도가 적용됩니다. 자세한 사항은 가맹점 홈페이지에서 확인하세요.

#### 3. 1999년생 출생자의 경우, 생일이 지난 다음달 1일에 '일반 회원'의 한도로 변경됩니다. (2018년 기준)

#### 4. 미성년자의 출금 한도는 송금액과 결제액이 합하여 계산됩니다.

예를 들어 이번 달에 30만원 송금하고 토스 가맹점에서 50만원 결제했다면, 다음 달이 되기 전엔 20만원이 넘는 송금이나 결제를 할 수 없습니다.

타인 명의 휴대폰으로도 토스를 사용할 수 있나요?



# 토스머니 대사 운영

BankingHistory2 - toss\_fb\_main 매칭 현황을 관리합니다.



지난 3일 간 이상 상황이 없습니다.

조회 시작 일시\*  
2018-08-15

조회 종료 일시\*  
2018-09-15

조회하기

✓ 9월 15일 (토) 법인계좌 잔여 0건 / 펌뱅킹 잔여 0건 ▼

✓ 9월 14일 (금) 법인계좌 잔여 0건 / 펌뱅킹 잔여 0건 ▼

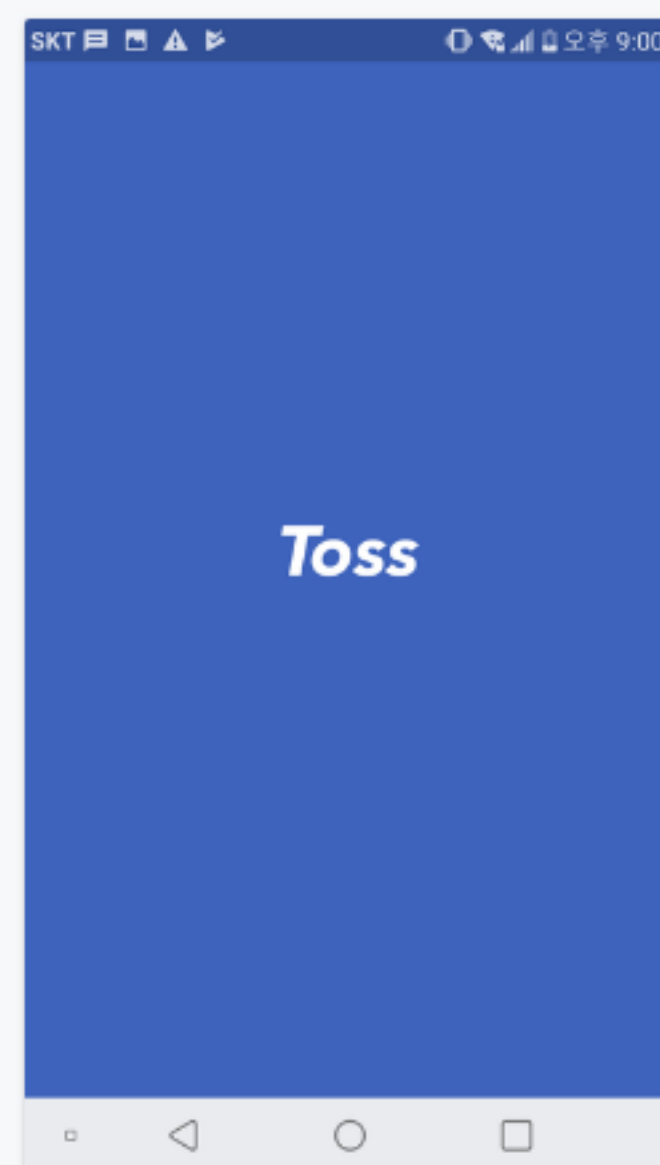
✓ 9월 13일 (목) 법인계좌 잔여 5건 / 펌뱅킹 잔여 1건 ▼

✓ 9월 12일 (수) 법인계좌 잔여 3건 / 펌뱅킹 잔여 1건 ▼

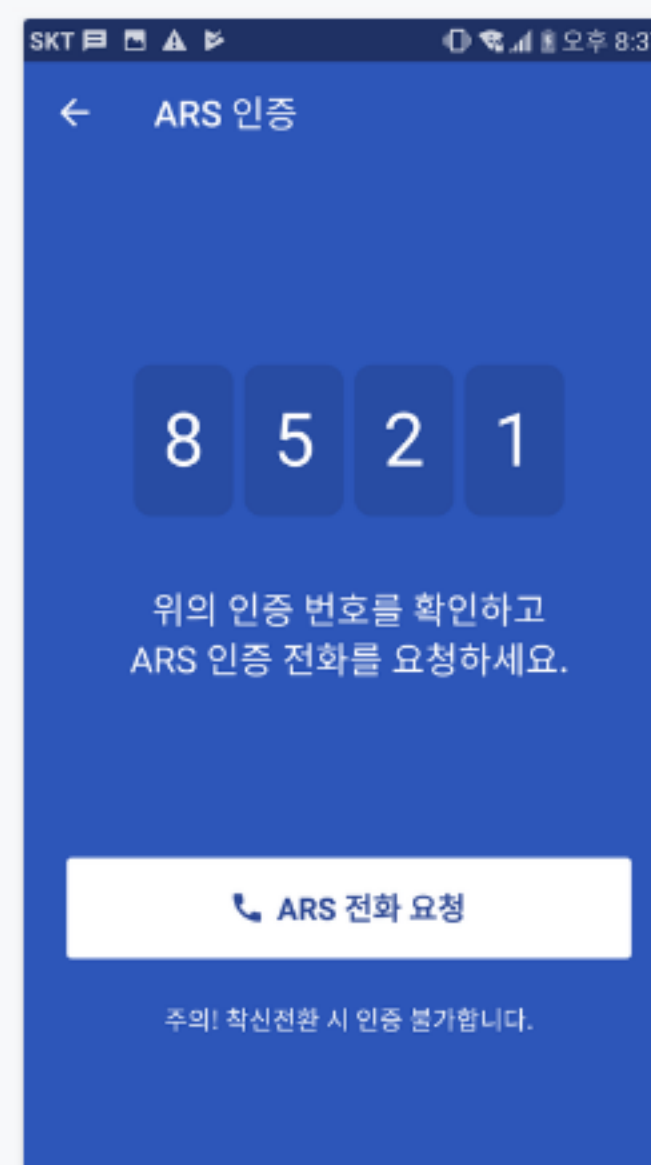
## ← 14 screens for 인트로

인트로 영역에서 필터 조건에 해당하는 스크린 14개가 검색되었습니다.

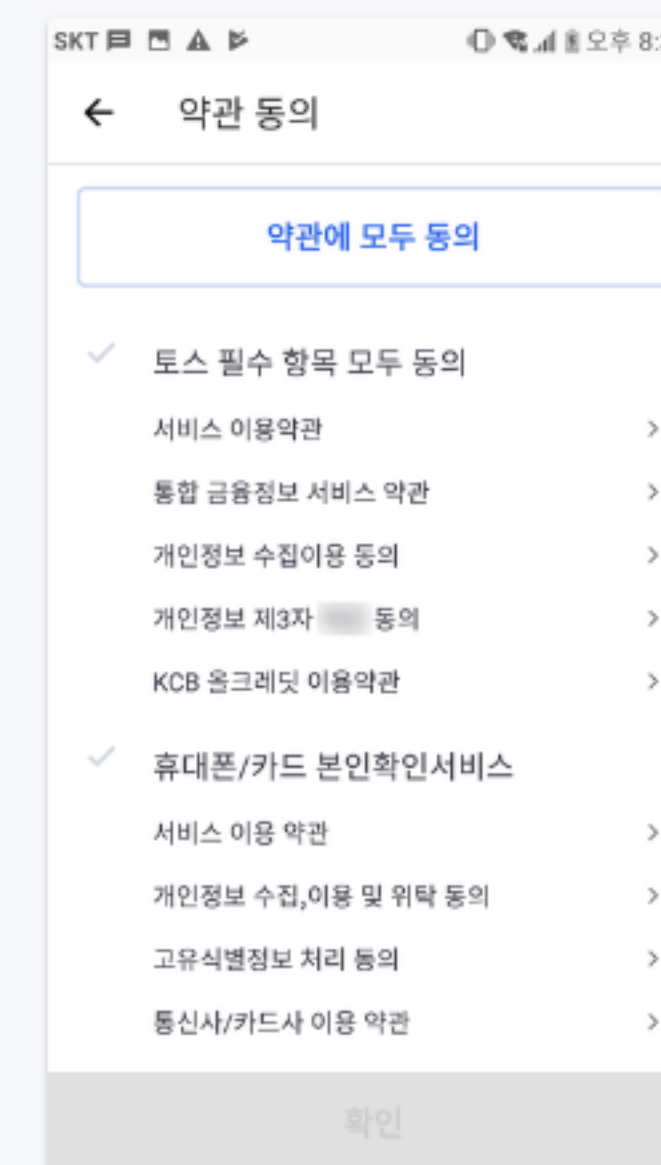
splash



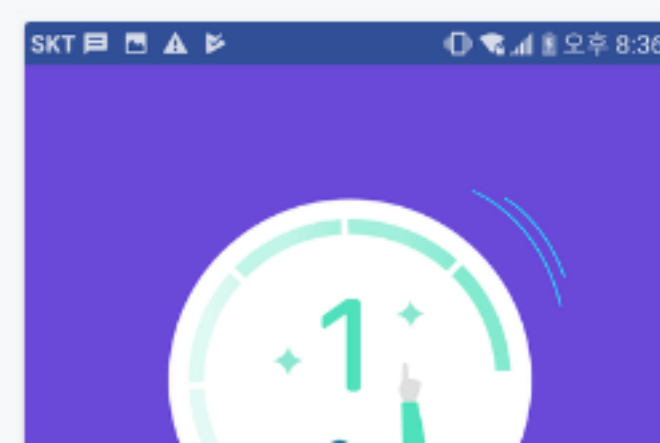
auth\_ars 6



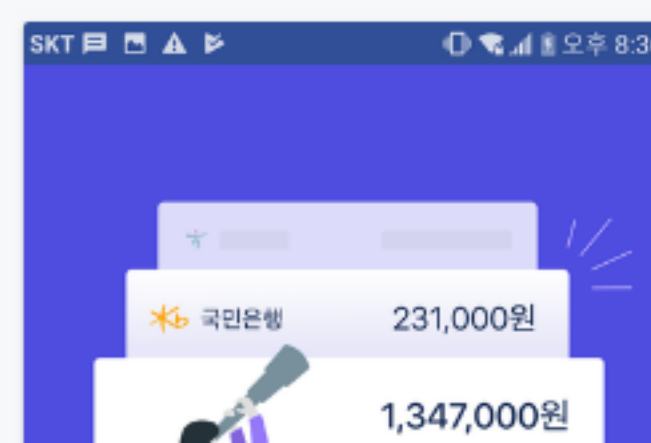
init\_guide\_4 2



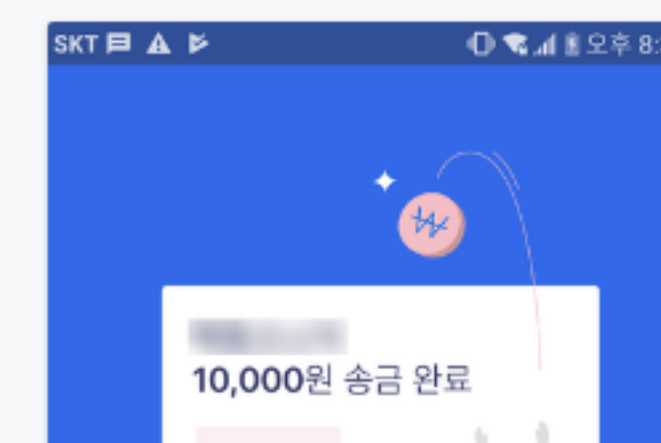
init\_guide\_3



init\_guide\_2



init\_guide\_1



1  
2  
3  
4  
5  
6  
7

# Why do we use ORM in the first place?

#ORM #왜쓸까

1. Why do we use ORM in the first place?

??? : “SQL 몰라도 된다고 하는 사람과는 일 안 한다”

1. Why do we use ORM in the first place?

??? : “SQL 몰라도 된다고 하는 사람과는 일 안 한다”



SQL queries from 1 connection

DO  
1183.69 ms (12 queries )

Query	Timeline	Time (ms)	Action
<div><div></div><div>+</div><div>SELECT @@SQL_AUTO_IS_NULL</div></div>	<div></div>	20.71	
<div><div></div><div>+</div><div>SELECT VERSION()</div></div>	<div></div>	21.33	
<div><div></div><div>+</div><div>SELECT . . . FROM `T0` INNER JOIN `T1` ON (`T0`.`account_no` = `T1`.`no`) WHERE (`T0`.`target_date` &lt;= '2018-10-06' AND `T0`.`target_date` &gt;= '2018-09-06' AND `T1`.`toss_money` = 1) GROUP BY `T0`.`target_date`, `T0`.`category_id` ORDER BY NULL</div></div>	<div></div>	302.85	<div>Sel</div> <div>Expl</div>
<div><div></div><div>+</div><div>SELECT . . . FROM `T2` INNER JOIN `T1` ON (`T2`.`account_no` = `T1`.`no`) WHERE (`T2`.`target_date` &lt;= '2018-10-06' AND `T2`.`target_date` &gt;= '2018-09-06' AND `T1`.`toss_money` = 1) GROUP BY `T2`.`target_date` ORDER BY NULL</div></div>	<div></div>	71.67	<div>Sel</div> <div>Expl</div>
<div><div></div><div>+</div><div>SELECT . . . FROM `T0` INNER JOIN `T1` ON (`T0`.`account_no` = `T1`.`no`) WHERE (`T0`.`target_date` &lt;= '2018-10-06' AND `T0`.`target_date` &gt;= '2018-09-06' AND `T1`.`toss_money` = 1)</div></div>	<div></div>	522.04	<div>Sel</div> <div>Expl</div>
<div><div></div><div>+</div><div>SELECT . . . FROM `T3` LEFT OUTER JOIN `T3` A ON (`T3`.`parent_id` = A.`id`) LEFT OUTER JOIN `T3` AA ON (A.`parent_id` = AA.`id`) LEFT OUTER JOIN `T3` AAA ON (AA.`parent_id` = AAA.`id`) WHERE `T3`.`id` IN (0, 257, 155, 307, 317, 407, 435, 271, 273, 167, 147, 149, 151, 409, 431, 285, 287, 289, 197, 419, 293, 423, 171, 159, 173, 157, 177, 179, 415, 437, 417, 183, 283, 185, 187, 411, 189, 309, 279, 175, 267, 325, 327, 329, 331, 319, 305, 433, 291, 215, 217, 221, 315, 351, 227, 195, 401, 233, NULL, 235, 237, 239, 241, 339, 349, 247, 343, 249, 251, 281, 277) ORDER BY `T3`.`ordering` ASC, `T3`.`codename` ASC</div></div>	<div></div>	36.98	<div>Sel</div> <div>Expl</div>
<div><div></div><div>+</div><div>SELECT . . . FROM `T1` INNER JOIN `T4` ON (`T1`.`bank_no` = `T4`.`no`)</div></div>	<div></div>	62.87	<div>Sel</div> <div>Expl</div>
<div><div></div><div>+</div><div>SELECT . . . FROM `T3` WHERE `T3`.`parent_id` IS NULL ORDER BY `T3`.`ordering` ASC, `T3`.`codename` ASC</div></div>	<div></div>	24.64	<div>Sel</div> <div>Expl</div>
<div><div></div><div>+</div><div>SELECT . . . FROM `T3` WHERE (`T3`.`id` IN (257, 299, 321, 307, 279, 349, 435, 271, 273, 147, 277, 151, 327, 409, 155, 157, 287, 289, 283, 293, 325, 167, 169, 227, 171, 159, 173, 175, 177, 179, 437, 183, 185, 415, 411, 317, 319, 193, 343, 195, 407, 197, 417, 329, 339, 331, 305, 269, 285, 433, 255, 291, 213, 303, 215, 267, 217, 309, 219, 221, 351, 225, 419, 315, 165, 401, 259, 233, 431, 423, 237, 239, 241, 275, 235, 187, 189, 247, 153, 249, 251, 281, 149) AND `T3`.`parent_id` IN (299, 303)) ORDER BY `T3`.`ordering` ASC, `T3`.`codename` ASC</div></div>	<div></div>	28.77	<div>Sel</div> <div>Expl</div>
<div><div></div><div>+</div><div>SELECT . . . FROM `T3` WHERE (`T3`.`id` IN (257, 299, 321, 307, 279, 349, 435, 271, 273, 147, 277, 151, 327, 409, 155, 157, 287, 289, 283, 293, 325, 167, 169, 227, 171, 159, 173, 175, 177, 179, 437, 183, 185, 415, 411, 317, 319, 193, 343, 195, 407, 197, 417, 329, 339, 331, 305, 269, 285, 433, 255, 291, 213, 303, 215, 267, 217, 309, 219, 221, 351, 225, 419, 315, 165, 401, 259, 233, 431, 423, 237, 239, 241, 275, 235, 187, 189, 247, 153, 249, 251, 281, 149) AND `T3`.`parent_id` IN (259, 321, 269, 147, 149, 153, 409, 411, 287, 289, 165, 169, 173, 431, 433, 293, 319, 193, 343, 195, 407, 197, 417, 329, 339, 331, 305, 269, 285, 433, 255, 291, 213, 303, 215, 267, 217, 309, 219, 221, 351, 225, 419, 315, 165, 401, 259, 233, 431, 423, 237, 239, 241, 275, 235, 187, 189, 247, 153, 249, 251, 281, 149) AND `T3`.`parent_id` IN (259, 321, 269, 147, 149, 153, 409, 411, 287, 289, 165, 169, 173, 431, 433, 293, 319, 193, 343, 195, 407, 197, 417, 329, 339, 331, 305, 269, 285, 433, 255, 291, 213, 303, 215, 267, 217, 309, 219, 221, 351, 225, 419, 315, 165, 401, 259, 233, 431, 423, 237, 239, 241, 275, 235, 187, 189, 247, 153, 249, 251, 281, 149) ORDER BY `T3`.`ordering` ASC, `T3`.`codename` ASC</div></div>	<div></div>	30.48	<div>Sel</div> <div>Expl</div>
<div><div></div><div>+</div><div>SELECT . . . FROM `T3` WHERE (`T3`.`id` IN (257, 299, 321, 307, 279, 349, 435, 271, 273, 147, 277, 151, 327, 409, 155, 157, 287, 289, 283, 293, 325, 167, 169, 227, 171, 159, 173, 175, 177, 179, 437, 183, 185, 415, 411, 317, 319, 193, 343, 195, 407, 197, 417, 329, 339, 331, 305, 269, 285, 433, 255, 291, 213, 303, 215, 267, 217, 309, 219, 221, 351, 225, 419, 315, 165, 401, 259, 233, 431, 423, 237, 239, 241, 275, 235, 187, 189, 247, 153, 249, 251, 281, 149) AND `T3`.`parent_id` IN (257, 279, 175, 271, 273, 277, 151, 283, 157, 415, 417, 419, 293, 167, 291, 171, 285, 177, 307, 155, 437, 183, 185, 423, 187, 317, 435, 195, 407, 197, 327, 331, 339, 215, 267, 217, 221, 351, 227, 233, 235, 237, 189, 241, 349, 249, 251)) ORDER BY `T3`.`ordering` ASC, `T3`.`codename` ASC</div></div>	<div></div>	35.57	<div>Sel</div> <div>Expl</div>
<div><div></div><div>+</div><div>SELECT . . . FROM `T3` WHERE (`T3`.`id` IN (257, 299, 321, 307, 279, 349, 435, 271, 273, 147, 277, 151, 327, 409, 155, 157, 287, 289, 283, 293, 325, 167, 169, 227, 171, 159, 173, 175, 177, 179, 437, 183, 185, 415, 411, 317, 319, 193, 343, 195, 407, 197, 417, 329, 339, 331, 305, 269, 285, 433, 255, 291, 213, 303, 215, 267, 217, 309, 219, 221, 351, 225, 419, 315, 165, 401, 259, 233, 431, 423, 237, 239, 241, 275, 235, 187, 189, 247, 153, 249, 251, 281, 149) AND `T3`.`parent_id` IN (305, 239, 401, 309, 343, 281, 315, 159)) ORDER BY `T3`.`ordering` ASC, `T3`.`codename` ASC</div></div>	<div></div>	25.77	<div>Sel</div> <div>Expl</div>

퍼포먼스 병목의 요인

DB I/O

1. Why do we use ORM in the first place?

1.

Joins, Aggregation, Indexes... 쿼리셋으로 돌아돌아 공부하면  
장님 코끼리 만지는 느낌을 벗어날 수 없다.



1. Why do we use ORM in the first place?

2.

데이터 엔지니어와 소통하기

다른 스택 개발자들과 소통하기

1. Why do we use ORM in the first place?

**쉽게 해 준다고 해서 애초에 고민해야 하는 전통적인 질문들이 사라지지 않는다는.**

1. Why do we use ORM in the first place?

**쉽게 해 준다고 해서 애초에 고민해야 하는 전통적인 질문들이 사라지지 않는다는.**

오히려...

1. Why do we use ORM in the first place?

**쉽게 해 준다고 해서 애초에 고민해야 하는 전통적인 질문들이 사라지지 않는다는.**

오히려...

DRY, object-oriented code 

1. Why do we use ORM in the first place?

**쉽게 해 준다고 해서 애초에 고민해야 하는 전통적인 질문들이 사라지지 않는다는.**

오히려...

DRY, object-oriented code 

SQL injection 

1. Why do we use ORM in the first place?

**결론: SQL 중요하다.**

1  
2  
3  
4  
5  
6  
7

# Is the ORM-generated schema good enough?

#ORM #스키마 #관찰을까

2. Is the ORM-generated schema good enough?

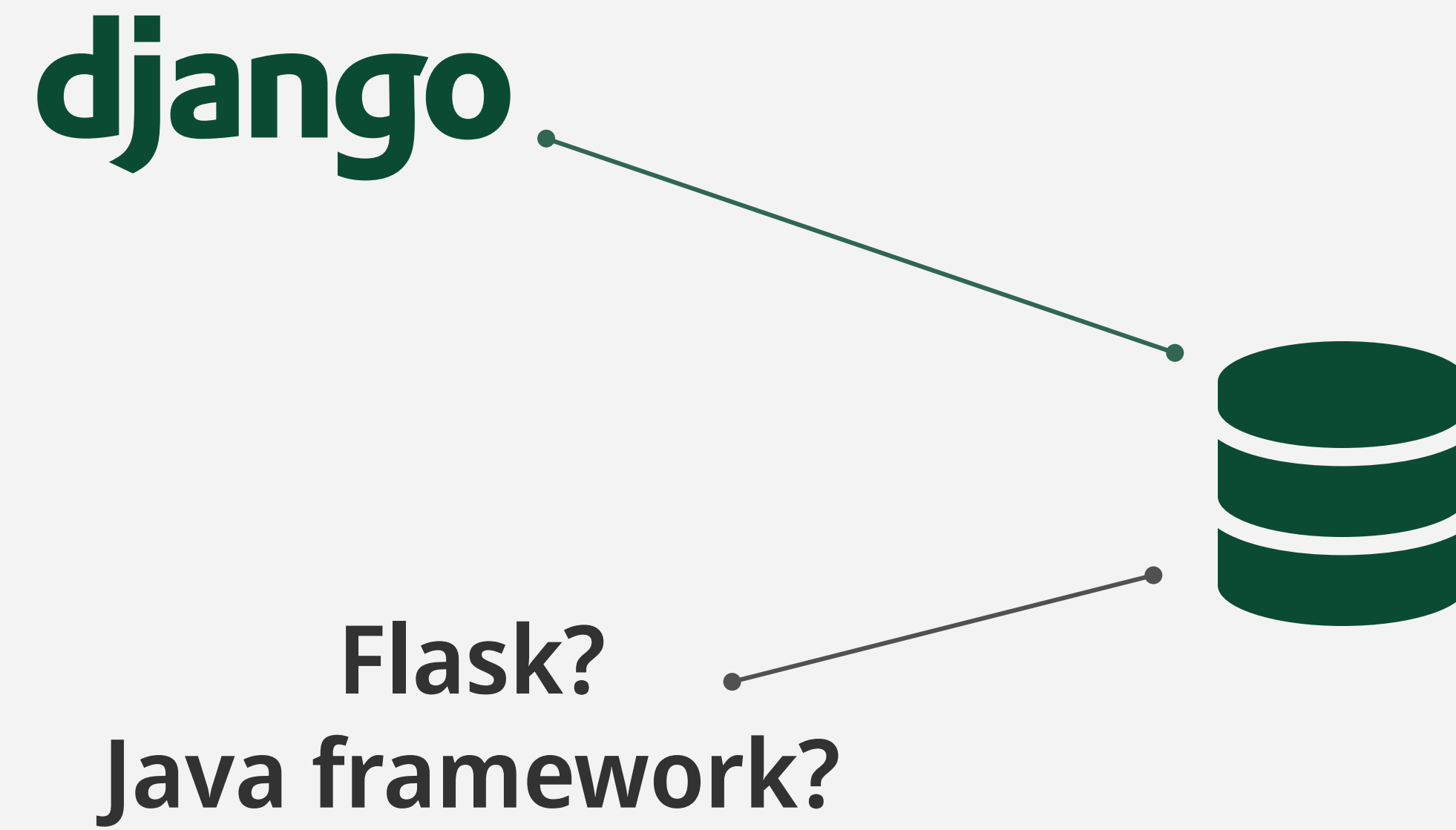




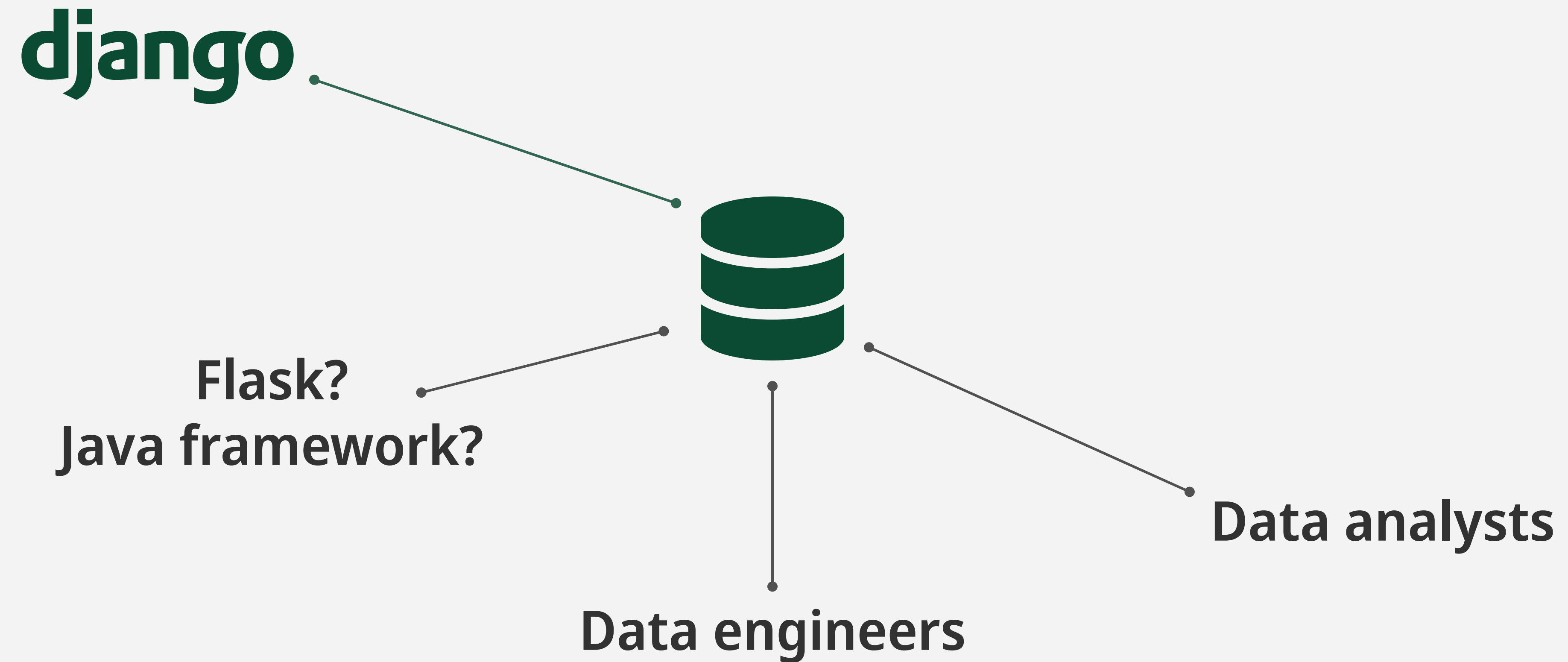
2. Is the ORM-generated schema good enough?



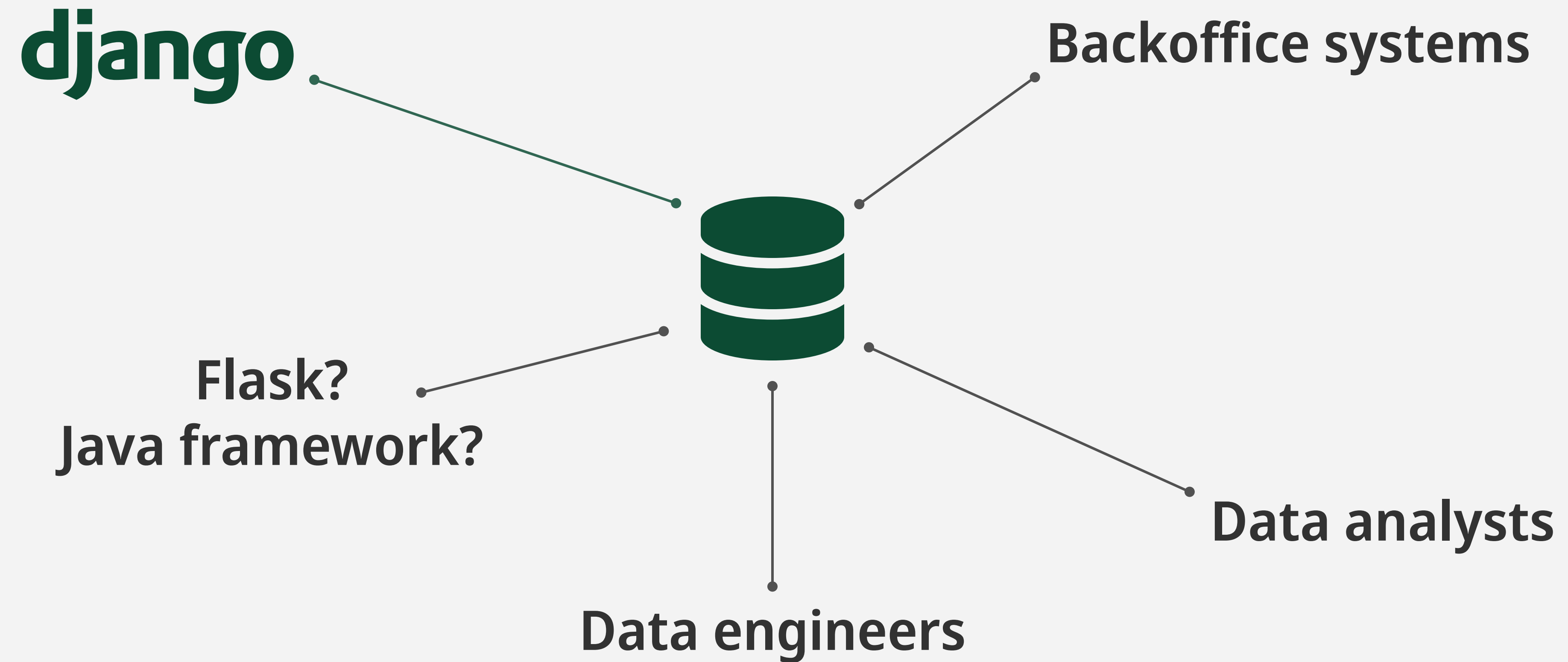
2. Is the ORM-generated schema good enough?



2. Is the ORM-generated schema good enough?



2. Is the ORM-generated schema good enough?



2. Is the ORM-generated schema good enough?

**DB는 영원히 장고만의 것이 아니다.**

**그 자체로 완결성 있어야 한다.**

2. Is the ORM-generated schema good enough?

# DDL

이름 짓기.

2. Is the ORM-generated schema good enough?

```
# banking/models.py
```

```
class BankingHistoryCategorizationManualLogEntry(models.Model):
```

```
    ...
```

```
-- SQL
```

```
CREATE TABLE banking_bankinghistorycategorizationmanuallogentry (
```

```
    ...
```

2. Is the ORM-generated schema good enough?

```
# banking/models.py
```

```
class BankingHistoryCategorizationManualLogEntry(models.Model):
```

```
...
```

```
-- SQL 🤔
```

```
CREATE TABLE banking_bankinghistorycategorizationmanuallogentry (
```

```
...
```



2. Is the ORM-generated schema good enough?

```
# banking/models.py
```

```
action_time = models.DateTimeField(db_index=True)  
slug = models.SlugField(unique=True)
```

```
-- SQL
```

```
ALTER TABLE banking_bankinghistorycategorizationmanuallogentry  
  ADD INDEX banking_bankinghistorycateg_action_time_fe6002e8 (action_time);  
ALTER TABLE banking_bankinghistorycategorizationmanuallogentry  
  ADD CONSTRAINT banking_bankinghistoryca_slug_b6ea4771_uniq UNIQUE (slug);
```

2. Is the ORM-generated schema good enough?

```
# banking/models.py
```

```
action_time = models.DateTimeField(db_index=True)
slug = models.SlugField(unique=True)
```

```
-- SQL 🤔
```

```
ALTER TABLE banking_bankinghistorycategorizationmanuallogentry
ADD INDEX banking_bankinghistorycateg_action_time_fe6002e8 (action_time);
ALTER TABLE banking_bankinghistorycategorizationmanuallogentry
ADD CONSTRAINT banking_bankinghistoryca_slug_b6ea4771_uniq UNIQUE (slug);
```

2. Is the ORM-generated schema good enough?

```
# banking/models.py
```

```
action_type = models.CharField(max_length=30, choices=ACTION_TYPE_CHOICES)
reason = models.CharField(max_length=255)
```

```
-- SQL
```

```
ALTER TABLE banking_bankinghistorycategorizationmanuallogentry
ADD COLUMN action_type varchar(30) NOT NULL;
ALTER TABLE banking_bankinghistorycategorizationmanuallogentry
ADD COLUMN reason varchar(255) NOT NULL;
```

2. Is the ORM-generated schema good enough?

```
# banking/models.py
```

```
action_type = models.CharField(max_length=30, choices=ACTION_TYPE_CHOICES)
reason = models.CharField(max_length=255)
```

```
-- SQL 🤔
```

```
ALTER TABLE banking_bankinghistorycategorizationmanuallogentry
ADD COLUMN action_type varchar(30) CHARACTER SET utf8 NOT NULL;
ALTER TABLE banking_bankinghistorycategorizationmanuallogentry
ADD COLUMN reason varchar(255) CHARACTER SET utf8 NOT NULL;
```

2. Is the ORM-generated schema good enough?

```
# banking/models.py
```

```
category = models.ForeignKey('BankingHistoryCategory', models.CASCADE)
```

```
-- SQL
```

```
ALTER TABLE banking_bankinghistorycategorizationmanuallogentry
```

```
  ADD COLUMN category_id varchar(30) NOT NULL;
```

```
ALTER TABLE banking_bankinghistorycategorizationmanuallogentry
```

```
  ADD CONSTRAINT banking_bankinghisto_category_id_ec0fe47a_fk_banking_b
```

```
  FOREIGN KEY (category_id) REFERENCES banking_bankinghistorycategory (slug);
```

2. Is the ORM-generated schema good enough?

```
# banking/models.py
```

```
category = models.ForeignKey('BankingHistoryCategory', models.CASCADE)
```

```
-- SQL 🤔
```

```
ALTER TABLE banking_bankinghistorycategorizationmanuallogentry
```

```
  ADD COLUMN category_id varchar(30) NOT NULL;
```

```
ALTER TABLE banking_bankinghistorycategorizationmanuallogentry
```

```
  ADD CONSTRAINT banking_bankinghisto_category_id_ec0fe47a_fk_banking_b
```

```
  FOREIGN KEY (category_id) REFERENCES banking_bankinghistorycategory (slug);
```

2. Is the ORM-generated schema good enough?

```
# banking/models.py
```

```
categories = models.ManyToManyField('BankingHistoryCategory')
```

```
-- SQL 😊
```

```
CREATE TABLE banking_bankinghistorycategorizationmanuallogentry_categories (  
    id int(11) AUTO_INCREMENT NOT NULL PRIMARY KEY,  
    bankinghistorycategorizationmanuallogentry_id int(11) NOT NULL,  
    bankinghistorycategory_id varchar(30) NOT NULL);  
ALTER TABLE banking_bankinghistorycategorizationmanuallogentry_categories  
    ADD CONSTRAINT banking_bankinghistoryca_bankinghistorycategoriza_773edd06_uniq  
    UNIQUE (bankinghistorycategorizationmanuallogentry_id, bankinghistorycategory_id);  
ALTER TABLE ...
```

## 2. Is the ORM-generated schema good enough?

- ✅ **항상** 테이블 & 인덱스 이름 지정하기
- ✅ **항상** M2M 중간 테이블 지정하기
- ✅ **스tring** 인코딩, 테이블 & 칼럼 코멘트, ...



2. Is the ORM-generated schema good enough?

+ ) 진지하게 오래 가는 DB를 만들 거면 migration은 끄자.

1  
2  
3  
4  
5  
6  
7

# What really is a ForeignKey?

#ForeignKey #대체될까

### 3. What really is a ForeignKey?

```
class User(models.Model):  
    id = models.IntegerField(  
        primary_key=True,  
    )  
  
class Post(models.Model):  
    id = models.IntegerField(  
        primary_key=True,  
    )  
    user = models.ForeignKey(  
        User,  
        on_delete=models.CASCADE,  
    )
```

### 3. What really is a ForeignKey?

```
class User(models.Model):  
    id = models.IntegerField(  
        primary_key=True,  
    )  
  
class Post(models.Model):  
    id = models.IntegerField(  
        primary_key=True,  
    )  
    user = models.ForeignKey(  
        User,  
        on_delete=models.CASCADE,  
    )
```

```
>>> post.user  
<User: User object (1)>
```

### 3. What really is a ForeignKey?

```
class User(models.Model):  
    id = models.IntegerField(  
        primary_key=True,  
    )  
  
class Post(models.Model):  
    id = models.IntegerField(  
        primary_key=True,  
    )  
    user = models.ForeignKey(  
        User,  
        on_delete=models.CASCADE,  
    )
```

```
>>> post.user  
<User: User object (1)>  
  
>>> post.user_id  
1
```

### 3. What really is a ForeignKey?

```
class User(models.Model):  
    id = models.IntegerField(  
        primary_key=True,  
    )  
  
class Post(models.Model):  
    id = models.IntegerField(  
        primary_key=True,  
    )  
    user = models.ForeignKey(  
        User,  
        on_delete=models.CASCADE,  
    )
```

```
>>> post.user  
<User: User object (1)>
```

```
>>> post.user_id  
1
```

```
>>> user.post_set  
<django.db.models.fields.related_descriptors.  
create_reverse_many_to_one_manager.<locals>.  
RelatedManager object at 0x10e082a90>
```

### 3. What really is a ForeignKey?

```
class User(models.Model):  
    id = models.IntegerField(  
        primary_key=True,  
    )  
  
class Post(models.Model):  
    id = models.IntegerField(  
        primary_key=True,  
    )  
    user = models.ForeignKey(  
        User,  
        on_delete=models.CASCADE,  
    )
```

```
CREATE TABLE user (  
    id int(11) NOT NULL,  
    PRIMARY KEY (id)  
);  
  
CREATE TABLE post (  
    id int(11) NOT NULL,  
    user_id int(11) NOT NULL,  
    PRIMARY KEY (id),  
    KEY c564eba6 (user_id),  
    CONSTRAINT c564eba6 FOREIGN KEY (user_id)  
        REFERENCES user (id)  
);
```

### 3. What really is a ForeignKey?

```
class User(models.Model):  
    id = models.IntegerField(  
        primary_key=True,  
    )  
  
class Post(models.Model):  
    id = models.IntegerField(  
        primary_key=True,  
    )  
    user = models.ForeignKey(  
        User,  
        on_delete=models.CASCADE,  
    )
```

```
CREATE TABLE user (  
    id int(11) NOT NULL,  
    PRIMARY KEY (id)  
);  
  
CREATE TABLE post (  
    id int(11) NOT NULL,  
    user_id int(11) NOT NULL,  
    PRIMARY KEY (id),  
    KEY c564eba6 (user_id),  
    CONSTRAINT c564eba6 FOREIGN KEY (user_id)  
        REFERENCES user (id)  
);
```



### 3. What really is a ForeignKey?

```
class User(models.Model):
    id = models.IntegerField(
        primary_key=True,
    )

class Post(models.Model):
    id = models.IntegerField(
        primary_key=True,
    )
    user = models.ForeignKey(
        User,
        on_delete=models.CASCADE,
    )
```

```
CREATE TABLE user (
    id int(11) NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE post (
    id int(11) NOT NULL,
    user_id int(11) NOT NULL,
    PRIMARY KEY (id),
    KEY c564eba6 (user_id),
    CONSTRAINT c564eba6 FOREIGN KEY (user_id)
        REFERENCES user (id)
);
```

### 3. What really is a ForeignKey?

```
class User(models.Model):
    id = models.IntegerField(
        primary_key=True,
    )

class Post(models.Model):
    id = models.IntegerField(
        primary_key=True,
    )
    user_id = models.IntegerField(
        db_index=True,
    )
    user = models.ForeignKey(
        User,
        on_delete=models.CASCADE,
        from_fields=['user_id'],
        to_fields=['id'],
    )
```

```
CREATE TABLE user (
    id int(11) NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE post (
    id int(11) NOT NULL,
    user_id int(11) NOT NULL,
    PRIMARY KEY (id),
    KEY c564eba6 (user_id),
    CONSTRAINT c564eba6 FOREIGN KEY (user_id)
        REFERENCES user (id)
);
```

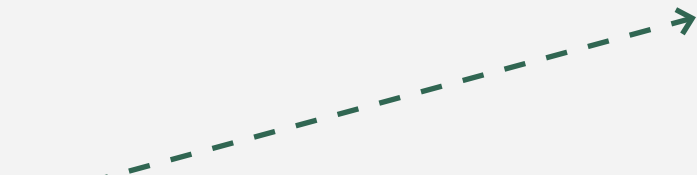
### 3. What really is a ForeignKey?

```
class User(models.Model):  
    id = models.IntegerField(  
        primary_key=True,  
    )
```

```
class Post(models.Model):  
    id = models.IntegerField(  
        primary_key=True,  
    )  
    user_id = models.IntegerField(  
        db_index=True,  
    )  
    user = models.ForeignKey(  
        User,  
        on_delete=models.CASCADE,  
        from_fields=['user_id'],  
        to_fields=['id'],  
    )
```

```
CREATE TABLE user (  
    id int(11) NOT NULL,  
    PRIMARY KEY (id)  
);
```

```
CREATE TABLE post (  
    id int(11) NOT NULL,  
    user_id int(11) NOT NULL,  
    PRIMARY KEY (id),  
    KEY c564eba6 (user_id),  
    CONSTRAINT c564eba6 FOREIGN KEY (user_id)  
        REFERENCES user (id)  
);
```



### 3. What really is a ForeignKey?


```
class User(models.Model):
    id = models.IntegerField(
        primary_key=True,
    )

class Post(models.Model):
    id = models.IntegerField(
        primary_key=True,
    )
    user_id = models.IntegerField(
        db_index=True,
    )
    user = models.ForeignKey(
        User,
        on_delete=models.CASCADE,
        from_fields=['user_id'],
        to_fields=['id'],
    )
```

### 3. What really is a ForeignKey?

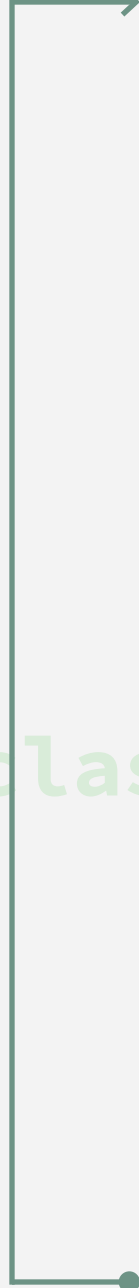
```
class User(models.Model):
    id = models.IntegerField(
        primary_key=True,
    )

class Post(models.Model):
    id = models.IntegerField(
        primary_key=True,
    )
    user_id = models.IntegerField(
        db_index=True,
    )
    user = models.ForeignKey(
        User,
        on_delete=models.CASCADE,
        from_fields=['user_id'],
        to_fields=['id'],
    )
```



### 3. What really is a ForeignKey?

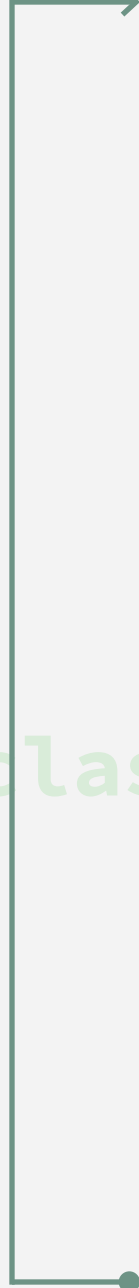
```
class User(models.Model):  
    → id = models.IntegerField(  
        primary_key=True,  
    )  
    post_set =  
        (ReverseManyToOneDescriptor instance)  
  
class Post(models.Model):  
    id = models.IntegerField(  
        primary_key=True,  
    )  
    • user_id = models.IntegerField(  
        db_index=True,  
    )  
    user =  
        (ForwardManyToOneDescriptor instance)
```



### 3. What really is a ForeignKey?

```
class User(models.Model):
    id = models.IntegerField(
        primary_key=True,
    )
    post_set =
        (ReverseManyToManyDescriptor instance)

class Post(models.Model):
    id = models.IntegerField(
        primary_key=True,
    )
    user_id = models.IntegerField(
        db_index=True,
    )
    user =
        (ForwardManyToOneDescriptor instance)
```

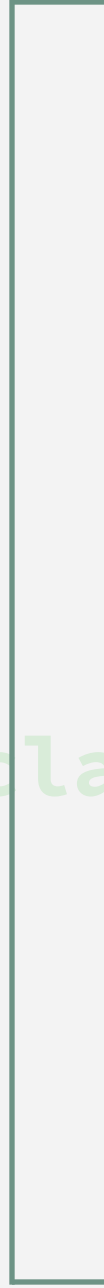


```
>>> post.user
<User: User object (1)>
```

### 3. What really is a ForeignKey?

```
class User(models.Model):
    id = models.IntegerField(
        primary_key=True,
    )
    post_set =
        (ReverseManyToOneDescriptor instance)

class Post(models.Model):
    id = models.IntegerField(
        primary_key=True,
    )
    user_id = models.IntegerField(
        db_index=True,
    )
    user =
        (ForwardManyToOneDescriptor instance)
```



```
>>> post.user
```

```
<User: User object (1)>
```

```
>>> user.post_set
```

```
<django.db.models.fields.related_descriptors.
create_reverse_many_to_one_manager.<locals>.
RelatedManager object at 0x10e082a90>
```



### 3. What really is a ForeignKey?

## Descriptors

<https://docs.python.org/3/howto/descriptor.html>

<https://dev.to/dawranliou/writing-descriptors-in-python-36>

1  
2  
3  
4  
5  
6  
7

# Referential integrity: how should we enforce it?

#참조무결성 #어떻게할까

4. Referential integrity: how should we enforce it?

**django**



**REFERENTIAL INTEGRITY**

#### 4. Referential integrity: how should we enforce it?

[...] This is because of referential integrity. In order to maintain a relationship between two objects, Django needs to know that the primary key of the related object is valid.

[...] Controls whether or not a constraint should be created in the database for this foreign key. The default is True, and that's almost certainly what you want; setting this to False can be very bad for data integrity.

#### 4. Referential integrity: how should we enforce it?

[...] This is because of **referential integrity**. In order to maintain a relationship between two objects, Django needs to know that the primary key of the related object is valid.

[...] Controls whether or not a constraint should be created in the database for this foreign key. The default is True, and that's almost certainly what you want; setting this to False **can be very bad for data integrity**.

#### 4. Referential integrity: how should we enforce it?

# Referential integrity

From Wikipedia, the free encyclopedia

Referential integrity is a property of data stating references within it are valid. In the context of relational databases, it requires every value of one column of a table to exist as a value of another column in a different (or the same) table.

#### 4. Referential integrity: how should we enforce it?

# Referential integrity

From Wikipedia, the free encyclopedia



Referential integrity is a property of data stating references within it are valid. In the context of relational databases, it requires every value of one column of a table to exist as a value of another column in a different (or the same) table.

artist_id	artist_name
1	Bono
2	Cher
3	Nuno Bettencourt

Link Broken

artist_id	album_id	album_name
3	1	Schizophonic
4	2	Eat the rich
3	3	Crave (single)



#### 4. Referential integrity: how should we enforce it?

```
class User(models.Model):  
    id = models.IntegerField(  
        primary_key=True,  
    )  
  
class Post(models.Model):  
    id = models.IntegerField(  
        primary_key=True,  
    )  
    user = models.ForeignKey(  
        User,  
        on_delete=models.CASCADE,  
    )
```

```
CREATE TABLE user (  
    id int(11) NOT NULL,  
    PRIMARY KEY (id)  
);  
  
CREATE TABLE post (  
    id int(11) NOT NULL,  
    user_id int(11) NOT NULL,  
    PRIMARY KEY (id),  
    KEY c564eba6 (user_id),  
    CONSTRAINT c564eba6 FOREIGN KEY (user_id)  
        REFERENCES user (id)  
);
```

#### 4. Referential integrity: how should we enforce it?

```
class User(models.Model):
    id = models.IntegerField(
        primary_key=True,
    )

class Post(models.Model):
    id = models.IntegerField(
        primary_key=True,
    )
    user = models.ForeignKey(
        User,
        on_delete=models.CASCADE,
    )
```

```
CREATE TABLE user (
    id int(11) NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE post (
    id int(11) NOT NULL,
    user_id int(11) NOT NULL,
    PRIMARY KEY (id),
    KEY c564eba6 (user_id),
    CONSTRAINT c564eba6 FOREIGN KEY (user_id)
        REFERENCES user (id)
);
```

#### 4. Referential integrity: how should we enforce it?

```
class User(models.Model):
    id = models.IntegerField(
        primary_key=True,
    )

class Post(models.Model):
    id = models.IntegerField(
        primary_key=True,
    )
    user = models.ForeignKey(
        User,
        on_delete=models.CASCADE,
    )
```

```
CREATE TABLE user (
    id int(11) NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE post (
    id int(11) NOT NULL,
    user_id int(11) NOT NULL,
    PRIMARY KEY (id),
    KEY c564eba6 (user_id),
    CONSTRAINT c564eba6 FOREIGN KEY (user_id)
        REFERENCES user (id)
);
```



4. Referential integrity: how should we enforce it?

참조 무결성이 지켜지지 않을 이유

**1. Hard deletions.**

#### 4. Referential integrity: how should we enforce it?

참조 무결성이 지켜지지 않을 이유

### 1. Hard deletions.

- Cascade operations

참조하는 행이 100만 개라면?

#### 4. Referential integrity: how should we enforce it?

참조 무결성이 지켜지지 않을 이유

### 1. Hard deletions.

- Cascade operations

참조하는 행이 100만 개라면?

- Null vs. invalid value

로그, 장부... 존재하지 않는 pk 값도 의미 있다

4. Referential integrity: how should we enforce it?

참조 무결성이 지켜지지 않을 이유

**2. Cross-database relations. Partitioning.**

4. Referential integrity: how should we enforce it?

참조 무결성이 지켜지지 않을 이유

## **2. Cross-database relations. Partitioning.**

- Cross-database relations

DB가 물리적으로 분리되면 정책 일관성이 깨진다



4. Referential integrity: how should we enforce it?

참조 무결성이 지켜지지 않을 이유

## 2. Cross-database relations. Partitioning.

- Cross-database relations

DB가 물리적으로 분리되면 정책 일관성이 깨진다

- Partitioning

테이블이 샤딩 내지 파티셔닝되는 경우의 어려움

1  
2  
3  
4  
5  
6  
7

# How are join types determined?

#JOIN #INNER #OUTER

5. How are join types determined?

`select_related()`을 하니 인스턴스가 사라진다?

## 5. How are join types determined?

`select_related()`을 하니 인스턴스가 사라진다?

```
>>> Post.objects.count()
```

```
100
```

## 5. How are join types determined?

`select_related()`을 하니 인스턴스가 사라진다?

```
>>> Post.objects.count()
```

```
100
```

```
>>> Post.objects.select_related('user').count()
```

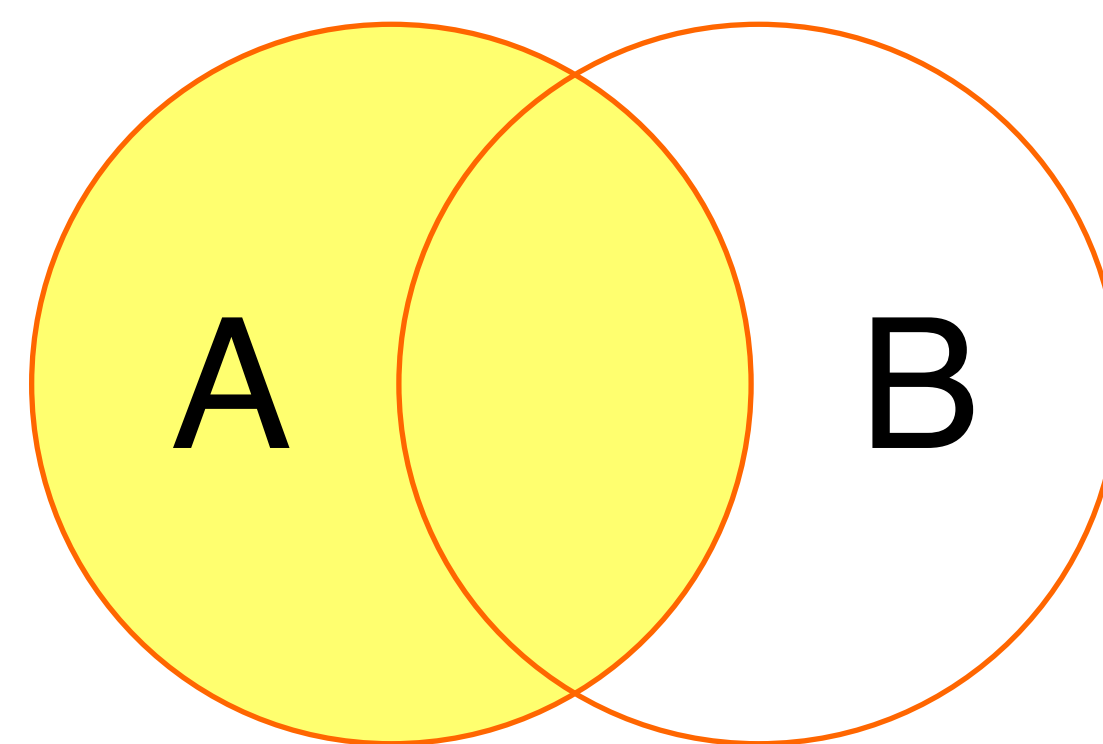
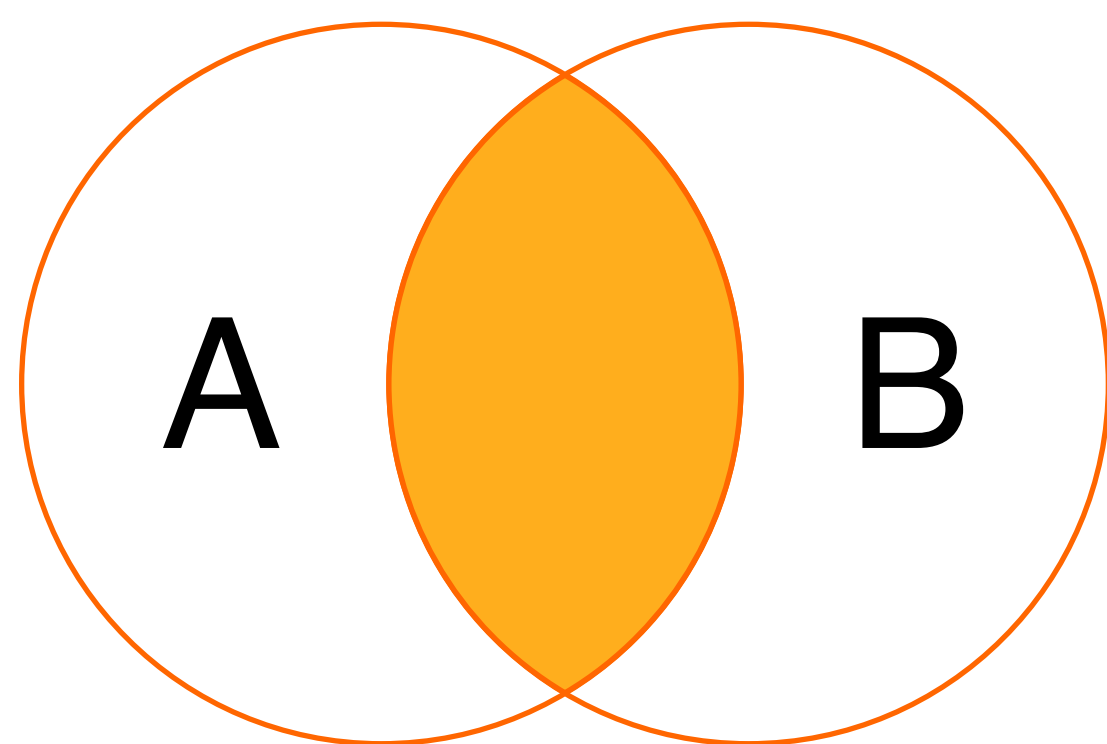
```
92
```

5. How are join types determined?

```
# django/db/models/sql/constants.py
```

```
INNER = 'INNER JOIN'
```

```
LOUTER = 'LEFT OUTER JOIN'
```



5. How are join types determined?

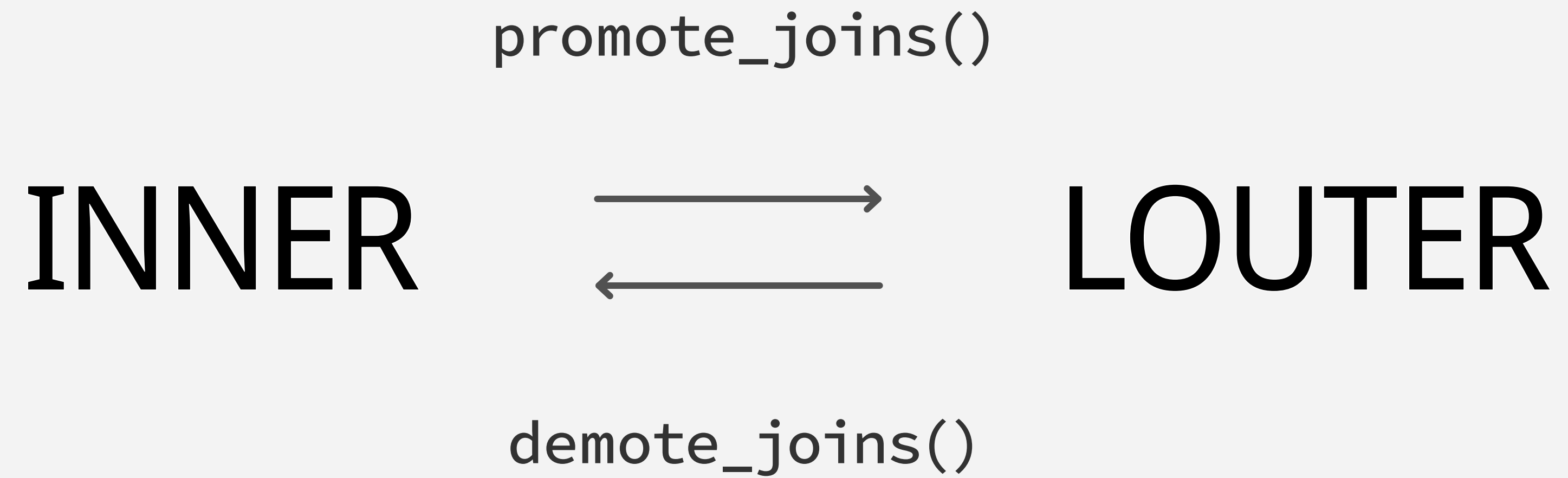
**INNER**



**LOUTER**



5. How are join types determined?



## 5. How are join types determined?

*# django/db/models/sql/query.py*

**class** Query:

**def** promote\_joins(self, aliases):

aliases = list(aliases)

**while** aliases:

alias = aliases.pop(0)

**if** self.alias\_map[alias].join\_type **is** None:

**continue**

parent\_alias = self.alias\_map[alias].parent\_alias

parent\_louter = parent\_alias **and** self.alias\_map[parent\_alias].join\_type == LOUTER

already\_louter = self.alias\_map[alias].join\_type == LOUTER

**if** ((

self.alias\_map[alias].nullable **or**

parent\_louter

) **and not** already\_louter):

self.alias\_map[alias] = self.alias\_map[alias].promote()

aliases.extend(

join **for** join **in** self.alias\_map

**if** self.alias\_map[join].parent\_alias == alias **and** join **not in** aliases

)

## 5. How are join types determined?

*# django/db/models/sql/query.py*

**class** Query:

**def** promote\_joins(self, aliases):

aliases = list(aliases)

**while** aliases:

alias = aliases.pop(0)

**if** self.alias\_map[alias].join\_type **is** None:

**continue**

parent\_alias = self.alias\_map[alias].parent\_alias

parent\_louter = parent\_alias **and** self.alias\_map[parent\_alias].join\_type == LOUTER

already\_louter = self.alias\_map[alias].join\_type == LOUTER

**if** ((

self.alias\_map[alias].nullable **or** 🙌

parent\_louter

) **and not** already\_louter):

self.alias\_map[alias] = self.alias\_map[alias].promote() 🙌

aliases.extend(

join **for** join **in** self.alias\_map

**if** self.alias\_map[join].parent\_alias == alias **and** join **not in** aliases

)

## 5. How are join types determined?

*# django/db/models/sql/query.py*

**class** Query:

**def** promote\_joins(self, aliases):

aliases = list(aliases)

**while** aliases:

alias = aliases.pop(0)

**if** self.alias\_map[alias].join\_type **is** None:

**continue**

parent\_alias = self.alias\_map[alias].parent\_alias

parent\_louter = parent\_alias **and** self.alias\_map[parent\_alias].join\_type == LOUTER

already\_louter = self.alias\_map[alias].join\_type == LOUTER

**if** ((

self.alias\_map[alias].nullable **or**

db\_constraint\_is\_false(self.alias\_map[alias]) **or**

parent\_louter

) **and not** already\_louter):

self.alias\_map[alias] = self.alias\_map[alias].promote()

aliases.extend(

join **for** join **in** self.alias\_map

**if** self.alias\_map[join].parent\_alias == alias **and** join **not in** aliases

)

1  
2  
3  
4  
5  
6  
7

# Can we handle cross-database relations?

#다중DB #참조

## 6. Can we handle cross-database relations?

Django **doesn't currently provide any support** for foreign key or many-to-many relationships spanning multiple databases. If you have used a router to partition models to different databases, any foreign key and many-to-many relationships defined by those models must be internal to a single database.

6. Can we handle cross-database relations?

**1** 데이터베이스 라우팅 문제

**2** 조인 실패 문제

6. Can we handle cross-database relations?

**1 데이터베이스 라우팅 문제**

**2 조인 실패 문제**



## 6. Can we handle cross-database relations?

*# routers.py*

**class** ExampleAuthRouter:

```
def db_for_read(self, model, **hints):  
    if model._meta.app_label == 'auth':  
        return 'default'
```

```
def db_for_write(self, model, **hints):  
    if model._meta.app_label == 'auth':  
        return 'default'
```

*# settings.py*

```
DATABASE_ROUTERS = [  
    'routers.ExampleAuthRouter',  
    ...  
]
```

## 6. Can we handle cross-database relations?

The hints received by the database router can be used to decide which database should receive a given request.

[...] Django tries each router in turn until a database suggestion can be found. If no suggestion can be found, it tries the current **\_state.db** of the hint instance.

## 6. Can we handle cross-database relations?

```
# django/db/utils.py
```

```
class ConnectionRouter:
```

```
    def db_for_read(self, model, **hints):
```

```
        chosen_db = None
```

```
        for router in self.routers:
```

```
            try:
```

```
                method = getattr(router, 'db_for_read')
```

```
            except AttributeError:
```

```
                pass
```

```
            else:
```

```
                chosen_db = method(model, **hints)
```

```
            if chosen_db:
```

```
                return chosen_db
```

```
instance = hints.get('instance')
```

```
if instance is not None and instance._state.db:
```

```
    return instance._state.db
```

```
return 'default'
```

## 6. Can we handle cross-database relations?

```
# django/db/utils.py
```

```
class ConnectionRouter:
```

```
    def db_for_read(self, model, **hints):
```

```
        chosen_db = None
```

```
        for router in self.routers:
```

```
            try:
```

```
                method = getattr(router, 'db_for_read')
```

```
            except AttributeError:
```

```
                pass
```

```
            else:
```

```
                chosen_db = method(model, **hints)
```

```
                if chosen_db:
```

```
                    return chosen_db
```

```
instance = hints.get('instance')
```

```
if instance is not None and instance._state.db: ➡
```

```
    return instance._state.db
```

```
return 'default'
```

## 6. Can we handle cross-database relations?



**ConnectionRouter()**에서 **instance** 힌트를 사용하지 못하도록

- 모든 모델이 커스텀 라우터에서 라우팅되도록 하거나,
- related descriptor에서 라우터에 힌트를 넘겨주는 부분을 모두 제거한다.

6. Can we handle cross-database relations?

**1** 데이터베이스 라우팅 문제

**2** 조인 실패 문제

## 6. Can we handle cross-database relations?



1. Join promotion 로직을 패치한다. (슬라이드 84)
2. **`select_related()`** 대신 **`prefetch_related()`**를 사용한다.

# 감사합니다.

발표 자료 repo

[bit.ly/devdjango-2018-ed](https://bit.ly/devdjango-2018-ed)

 edcwkim

 edcwkim@gmail.com