

JPA / Hibernate Advanced Mappings

Nuttachai Kulthammanit

JPA / Hibernate Advanced Mappings Overview

Chapter 1

Advanced Mappings

- ใน Database ของเราส่วนใหญ่จะมีหลายตาราง
 - ตารางแต่ละอันก็จะมีความสัมพันธ์กัน
 - One-to-one
 - One-to-many
 - Many-many
- เราต้องใส่ข้อมูลอันนี้ให้กับ Hibernate ด้วย

One-to-one Mappings

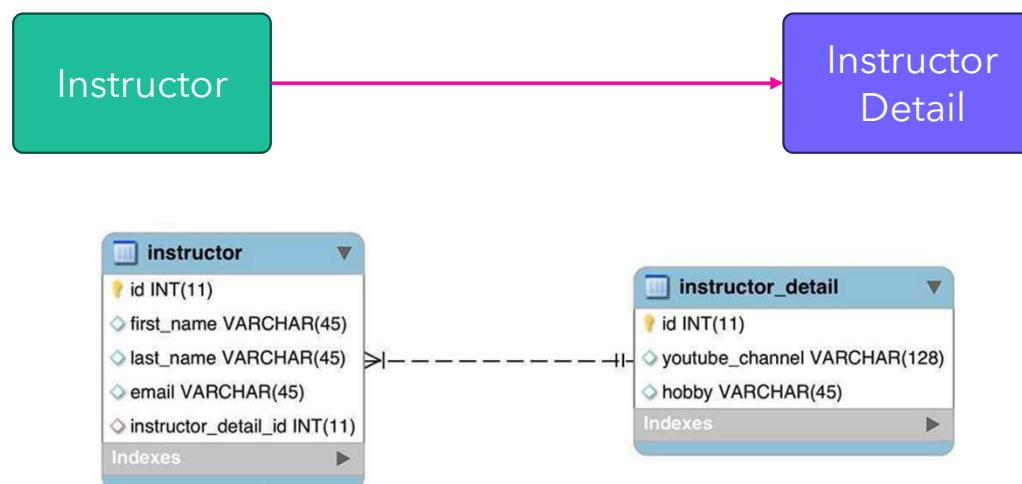
- แบบหนึ่งต่อหนึ่ง (One-to-one)
 - ความสัมพันธ์แบบหนึ่งต่อหนึ่ง เป็นความสัมพันธ์ระหว่างสิ่งหนึ่งกับสิ่งหนึ่งที่มีเพียงหนึ่งเดียวเท่านั้น เช่น อธิการบดีมีหน้าที่บริหารมหาวิทยาลัยเพียงมหาวิทยาลัยเดียวและในมหาวิทยาลัยนั้น ๆ จะมีอธิการบดีบริหารงานในขณะนั้น ๆ เพียงคนเดียวเช่นกัน สามารถเขียนเป็นไกด์อะแกรมได้ดังนี้



Ref: [หนึ่งของความสัมพันธ์ - ระบบฐานข้อมูล \(Database System\) \(google.com\)](#)

One-to-one Mappings

- Instructor สามารถมี Instructor Detail ได้ 1 อัน



One-to-many Mappings

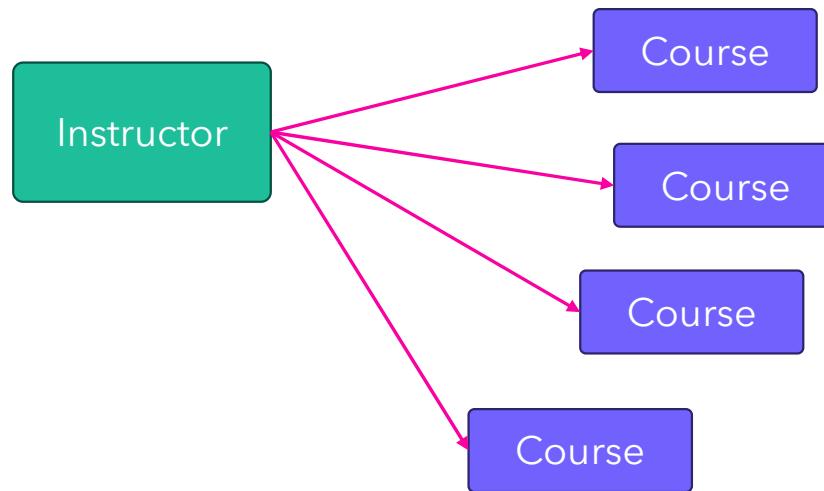
- แบบหนึ่งต่อหลาย (One-to-many)
- ความสัมพันธ์แบบหนึ่งต่อหลาย เป็นความสัมพันธ์ระหว่างสิ่งหนึ่งกับสิ่งหนึ่งที่มีเพียงหนึ่งกับอีกด้านหนึ่งเป็นกลุ่ม เช่น สมาชิกผู้บริจาก โลหิตสามารถบริจาคโลหิตได้หลาย ๆ ครั้งและ การบริจาคนั้นบริจากโดยสมาชิกคนเดียว สามารถเขียนเป็นໄ叨อะแกรมได้ดังนี้



Ref: [บทดูของความสัมพันธ์ - ระบบฐานข้อมูล \(Database System\) \(google.com\)](#)

One-to-many Mappings

- Instructor สามารถมี Courses ได้หลายอัน



Many-to-many Mappings

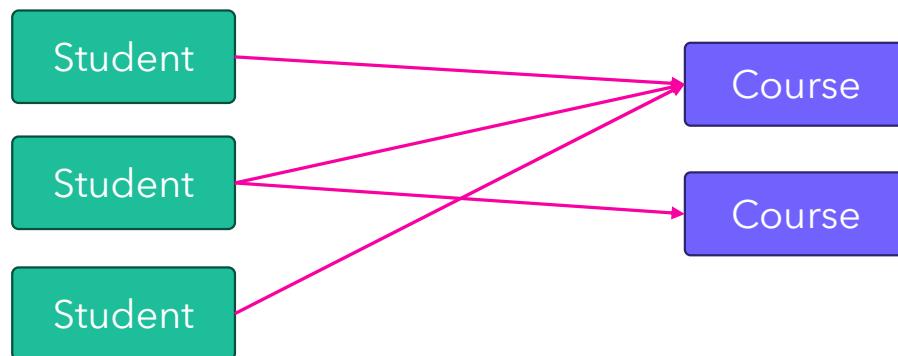
- แบบกลุ่มต่อกลุ่ม (Many-to-many)
 - ความสัมพันธ์แบบกลุ่มต่อกลุ่ม เป็นความสัมพันธ์ระหว่างสิ่งหนึ่งกับสิ่งหนึ่งที่มีได้หลาย ๆ อย่าง เช่น นักศึกษาสามารถลงทะเบียนเรียนได้หลาย ๆ รายวิชาและในแต่ละรายวิชานี้ นักศึกษาลงทะเบียนเรียนได้หลาย ๆ คน สามารถเขียนเป็นโครงแกรมได้ดังนี้



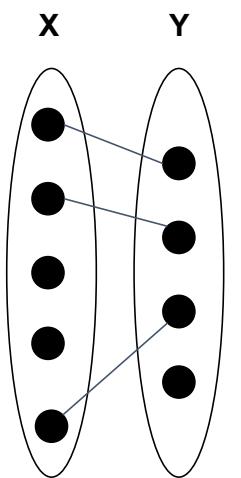
Ref: [บทดูของความสัมพันธ์ - ระบบฐานข้อมูล \(Database System\) \(google.com\)](#)

Many-to-many Mappings

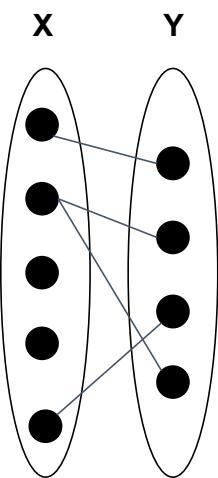
- Students เรียนได้หลาย Courses
- Courses หนึ่งมี Student เรียนได้หลายคน



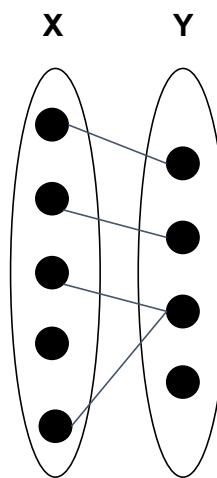
ความสัมพันธ์ระหว่างตาราง



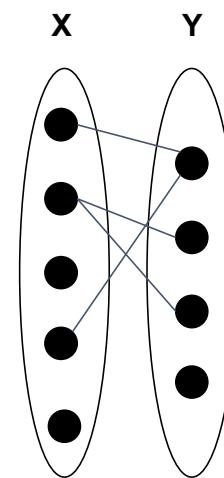
1-to-1



1-to-Many

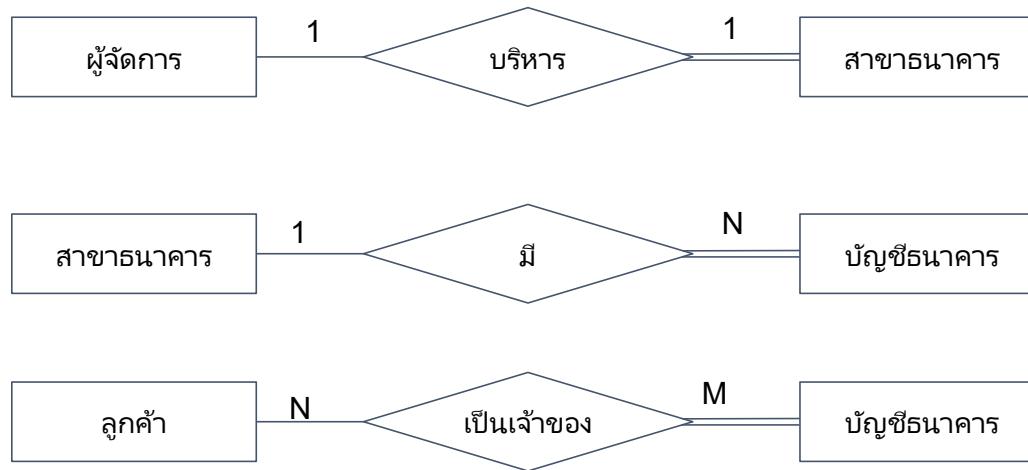


Many-to-1



Many-to-Many

ความสัมพันธ์ระหว่างตาราง



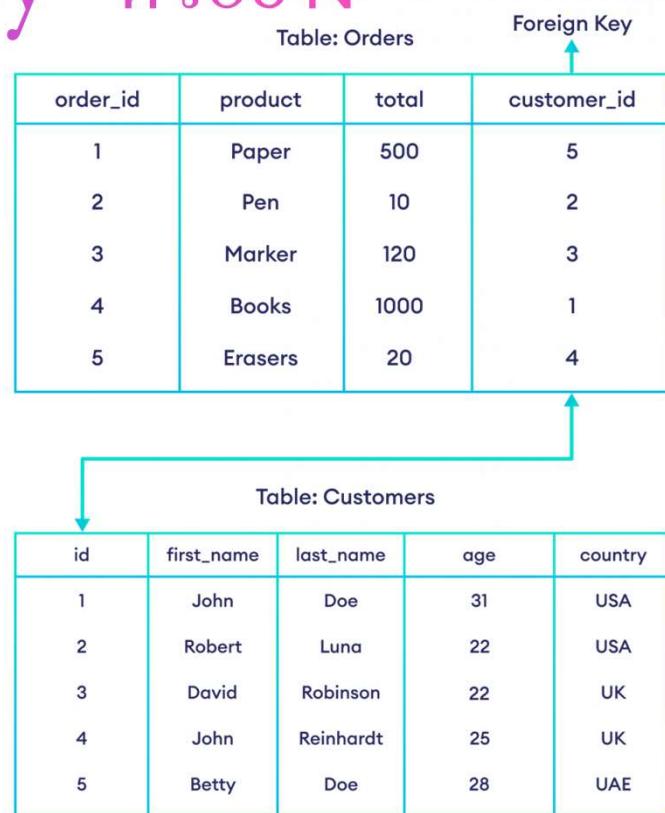
หลักการของฐานข้อมูล

- Primary Key
- Foreign Key
- Cascade

Primary Key และ Foreign Key

- Primary key: ใช้สำหรับ identify ตัว Rows ในตาราง
- Foreign key:
 - ใช้สำหรับการเชื่อมตารางเข้าด้วยกัน
 - เป็นคอลัมน์ใน Database ที่ refers ถึง Primary key ในตารางอื่น

Foreign Key - ຕົວຢ່າງ



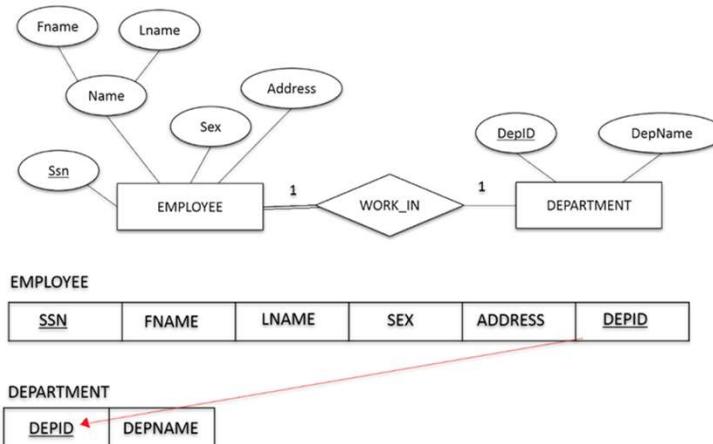
Ref: <https://www.programiz.com/sites/tutorial2program/files/foreign-key.png>

การแปลงจาก ER-Diagram เป็น Schema

ทบทวน

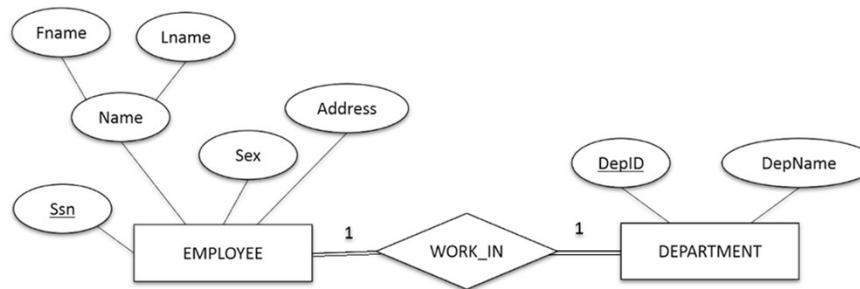
One-to-one

- วิธีที่ 1 เพิ่ม Foreign key ไปที่ตารางลูก (ตารางที่อยู่เดียว ๆ ไม่ได้ – ตารางที่เป็น Total Participation)
 - เช่น คน(ตารางแม่) กับ ใบขับขี่(ตารางลูก)
 - เอา id คนไปใส่ใน ใบขับขี่
 - เพราะ ใบขับขี่ จะอยู่แบบไม่มีคนไม่ได้
 - แต่คนอยู่แบบไม่มีใบขับขี่ได้
 - เช่น แผนก(ตารางแม่) กับ พนักงาน(ตารางลูก)
 - เอา id แผนกไปใส่ใน พนักงาน
 - เพราะ พนักงานอยู่เดียว ๆ แบบไม่มีแผนกไม่ได้
 - แต่แผนกอยู่เดียว ๆ แบบไม่มีพนักงานได้
 - ถ้าตารางนั้นอยู่เดียว ๆ ได้ทั้งคู่อาจจะพิจารณาวิธีที่ 2



One-to-one

- วิธีที่ 2 รวมตารางเข้าด้วยกัน – ถ้าทั้งสองตารางอยู่เดี่ยว ๆ ได้ด้วยกันทั้งคู่ (เป็น Total Participation ทั้งสองข้าง)

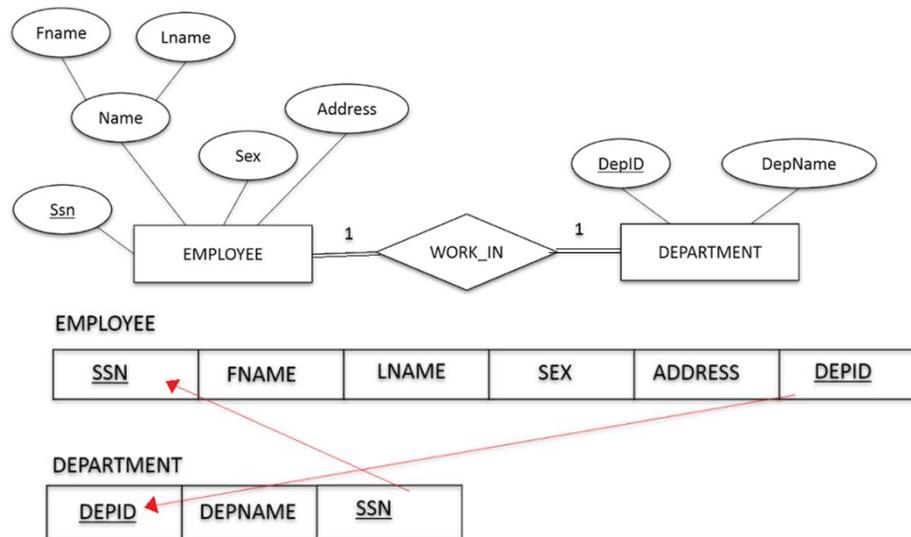


EMPLOYEE_DEPARTMENT

SSN	FNAME	LNAME	SEX	ADDRESS	DEPID	DEPNAME
-----	-------	-------	-----	---------	-------	---------

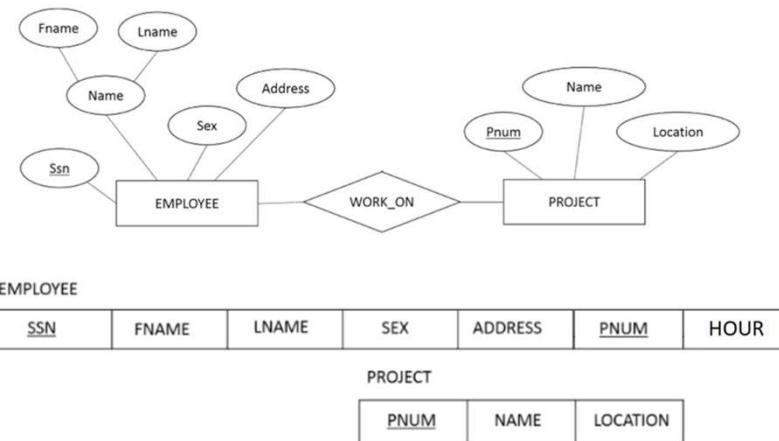
One-to-one

- วิธีที่ 3 Cross reference - เอา Primary key ของแต่ละตารางไปใส่เป็น Foreign key ของอีกตาราง



One-to-many

- ให้นำ primary key ของฝั่ง one ไปใส่เป็น foreign key ของฝั่ง many



One-to-many

- ให้นำ primary key ของฝั่ง one ไปใส่เป็น foreign key ของฝั่ง many

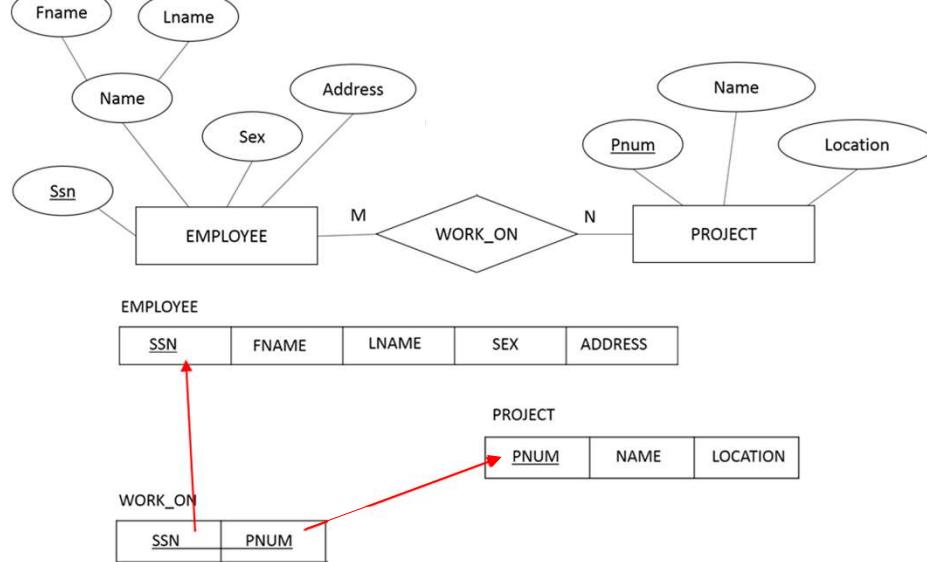
The diagram illustrates a One-to-Many relationship between two tables: Orders and Items. An arrow labeled "One-to-Many" points from the Orders table to the Items table. The Orders table has columns ID, Total, and Date. The Items table has columns ID, OrderID, Name, Qty, and Price.

Orders		
ID	Total	Date
1	20.00	1/2/2015
2	10.00	2/3/2015
3	15.00	5/10/2015

Items				
ID	OrderID	Name	Qty	Price
1	1	Shirt	1	10.00
2	1	Socks	2	5.00
3	2	Belt	1	10.00

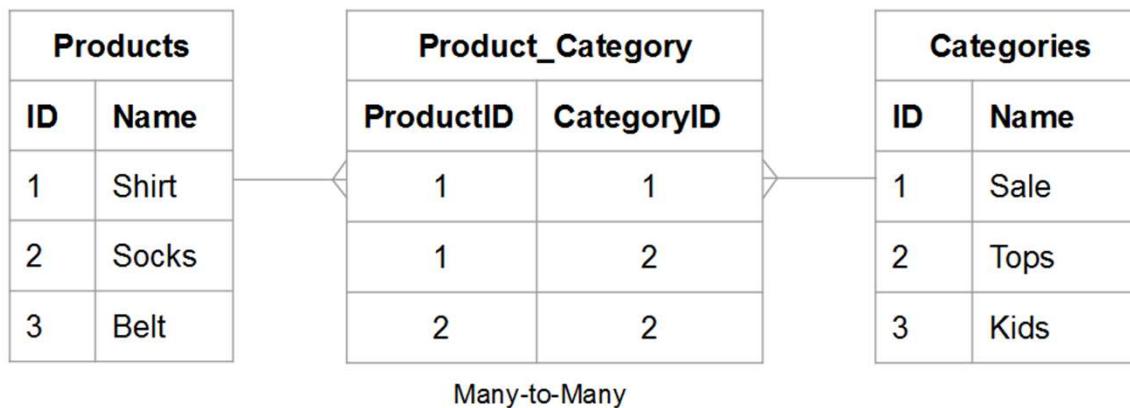
Many-to-many

- เพิ่มตารางกลางขึ้นมาเพื่อเป็นตัวบอกความสัมพันธ์ระหว่าง 2 ตารางนี้ โดยตารางกลางจะเก็บ Foreign key ของทั้งสองตารางนั้นไว้



Many-to-many

- เพิ่มตารางกลางขึ้นมาเพื่อเป็นตัวบอกความสัมพันธ์ระหว่าง 2 ตารางนี้ โดยตารางกลางจะเก็บ Foreign key ของทั้งสองตารางนั้นไว้

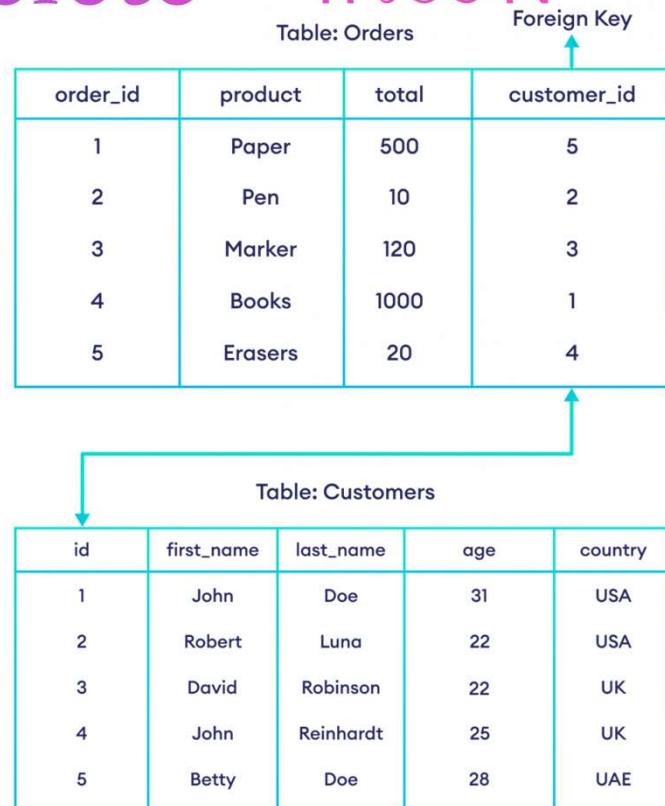


Cascade

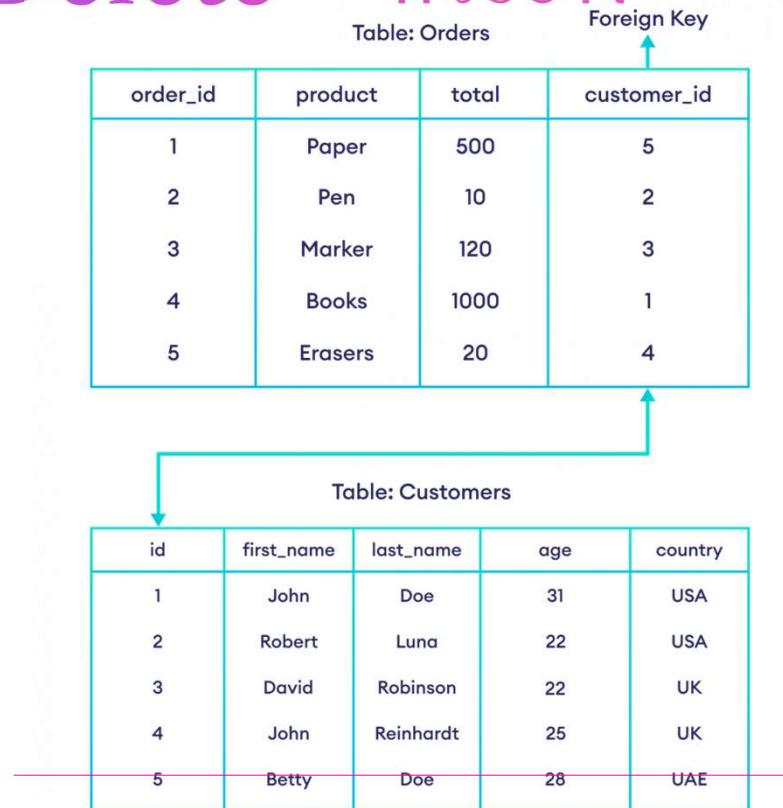
- เราสามารถทำ cascade operation ได้
- เป็นการทำ operation ที่จะกระทบไปถึงตัว entities ที่มันมีความสัมพันธ์ด้วย
 - เช่น ถ้าเรา save ตัว Instructor ตัว InstructorDetail ก็จะถูก Save ไปด้วย
 - ถ้าเราลบ Instructor ตัว InstructorDetail ก็จะถูกลบไปด้วย
 - เรียกว่า “CASCADE DELETE”



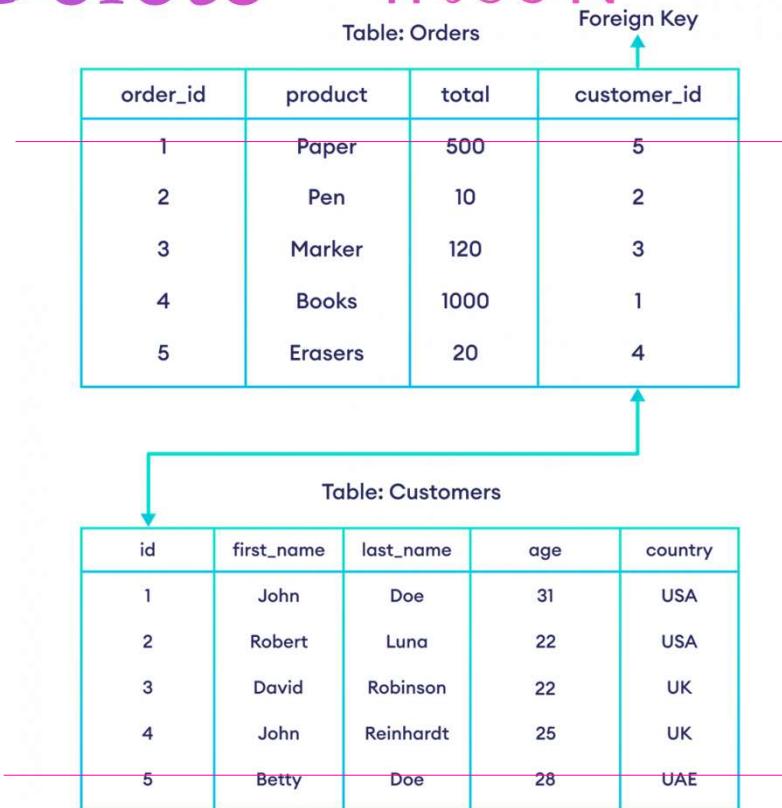
Cascade Delete – ตัวอย่าง



Cascade Delete – ตัวอย่าง

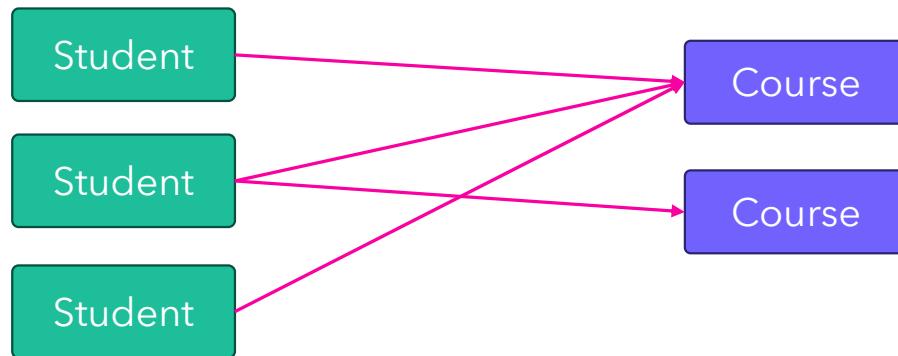


Cascade Delete – ตัวอย่าง



Cascade Delete

- Cascade delete จะขึ้นอยู่กับสถานการณ์ด้วย
- เช่น ตัวอย่างนี้เราก็ไม่ควรทำ Cascade delete



Fetch Types: Eager vs Lazy Loading

- เวลาเราดึง (Fetch) ข้อมูล เราควรดึงมาทั้งหมดเลยไหม?
 - เวลาเราดึง Student มาเราควรดึง Entity ที่เกี่ยวข้องมาด้วยเลยไหม
 - เช่น Course ที่นักเรียนลงทะเบียนมาด้วยในตอนนั้นที่ยังไม่แน่ใจว่าจะใช้ด้วยไหม
 - Eager จะดึงข้อมูลที่เกี่ยวข้องมาทั้งหมด
 - Lazy ยังไม่ดึงจนกว่าเราจะใช้งาน

Uni-Directional

- ถ้าเราดึง Instructor มา เราสามารถเข้าถึง InstructorDetail จาก Instructor ได้เลย
- แต่เราดึง Instructor Detail มาเราไม่สามารถเข้าถึง Instructor ได้
- เพราะเป็น Uni-Directional



Bi-Directional

- ถ้าเราดึง Instructor มา เราสามารถเข้าถึง InstructorDetail จาก Instructor ได้เลย
- และถ้าเราดึง Instructor Detail มาเราก็สามารถเข้าถึง Instructor ได้เช่นกัน
- เพราะเป็น Bi-Directional

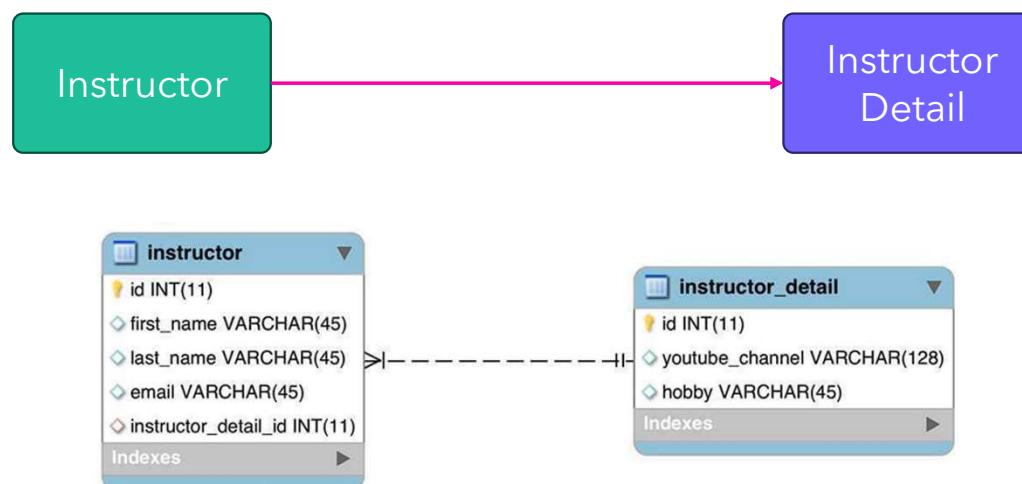


One-to-one Mapping

Chapter 2

One-to-one Mappings

- Instructor สามารถมี Instructor Detail ได้ 1 อัน



Uni-Directional

- ถ้าเราดึง Instructor มา เราสามารถเข้าถึง InstructorDetail จาก Instructor ได้เลย
- แต่เราดึง Instructor Detail มาเราไม่สามารถเข้าถึง Instructor ได้
- เพราะเป็น Uni-Directional



One-to-one Mapping และ Uni-Directional

- วิธีทำ

1. สร้างตารางใน Database
2. สร้างคลาส InstructorDetail
3. สร้างคลาส Instructor
4. สร้าง Main App

One-to-one Mapping และ Uni-Directional

- 1. สร้างตารางใน Database
 - ตาราง: instructor_detail

```
CREATE TABLE `instructor_detail` (
  `id` int NOT NULL AUTO_INCREMENT,
  `youtube_channel` varchar(128) DEFAULT NULL,
  `hobby` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`id`),
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;
```



One-to-one Mapping และ Uni-Directional

- 1. สร้างตารางใน Database
 - ตาราง: instructor

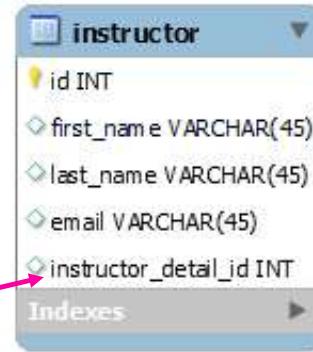
```
CREATE TABLE `instructor` (
  `id` int NOT NULL AUTO_INCREMENT,
  `first_name` varchar(45) DEFAULT NULL,
  `last_name` varchar(45) DEFAULT NULL,
  `email` varchar(45) DEFAULT NULL,
  `instructor_detail_id` int DEFAULT NULL,
  PRIMARY KEY (`id`),
  ...
)
```



One-to-one Mapping และ Uni-Directional

- 1. สร้างตารางใน Database
 - ตาราง: instructor

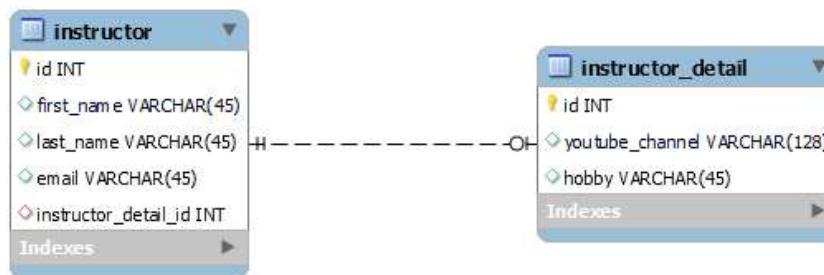
```
CREATE TABLE `instructor` (
  `id` int NOT NULL AUTO_INCREMENT,
  `first_name` varchar(45) DEFAULT NULL,
  `last_name` varchar(45) DEFAULT NULL,
  `email` varchar(45) DEFAULT NULL,
  `instructor_detail_id` int DEFAULT NULL,
  PRIMARY KEY (`id`),
  ...
)
```



columน์สำหรับการทำ
 Foreign key

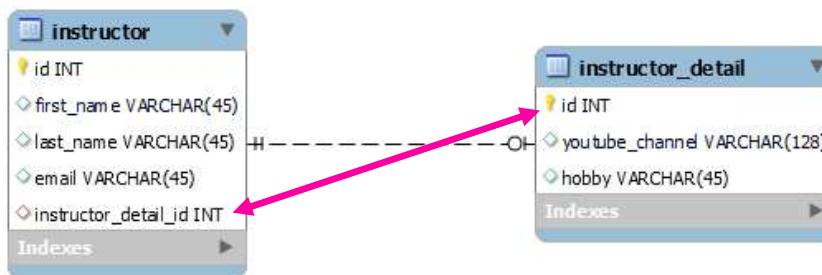
Foreign key

- ใช้ในการเชื่อมตารางเข้าด้วยกัน
- เป็น Field ในตารางที่อ้างอิงถึง Primary key ของอีกตาราง



Foreign key

- ใช้ในการเชื่อมตารางเข้าด้วยกัน
- เป็น Field ในตารางที่อ้างอิงถึง Primary key ของอีกตาราง



Foreign key – ตัวอย่าง

ตาราง: instructor

id	first_name	last_name	instructor_detail_id
1	Somchai	Rukdee	3
2	Anek	Oreo	1
3	Somboon	Pochana	2

ตาราง: instructor_detail

id	youtube_channel	hobby
1	www.youtube.com/oreo_tv	Piano
2	www.youtube.com/somboon_food	Football
3	www.youtube.com/somchai_good_learn	Running

Foreign key – ตัวอย่าง

คอลัมน์
Foreign key

ตาราง: instructor

id	first_name	last_name	instructor_detail_id
1	Somchai	Rukdee	3
2	Anek	Oreo	1
3	Somboon	Pochana	2

ตาราง: instructor_detail

id	youtube_channel	hobby
1	www.youtube.com/oreo_tv	Piano
2	www.youtube.com/somboon_food	Football
3	www.youtube.com/somchai_good_learn	Running

Foreign key – ตัวอย่าง

ตาราง: instructor

id	first_name	last_name	instructor_detail_id
1	Somchai	Rukdee	3
2	Anek	Oreo	1
3	Somboon	Pochana	2

คอลัมน์
Foreign key

ตาราง: instructor_detail

id	youtube_channel	hobby
1	www.youtube.com/oreo_tv	Piano
2	www.youtube.com/somboon_food	Football
3	www.youtube.com/somchai_good_learn	Running

Foreign key – ตัวอย่าง

ตาราง: instructor

id	first_name	last_name	instructor_detail_id
1	Somchai	Rukdee	3
2	Anek	Oreo	1
3	Somboon	Pochana	2

คอลัมน์
Foreign key

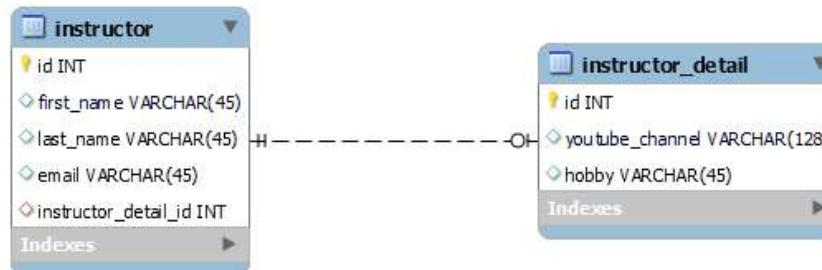
ตาราง: instructor_detail

id	youtube_channel	hobby
1	www.youtube.com/oreo_tv	Piano
2	www.youtube.com/somboon_food	Football
3	www.youtube.com/somchai_good_learn	Running

วิธีการกำหนด Foreign key

```
CREATE TABLE `instructor` (
  `id` int NOT NULL AUTO_INCREMENT,
  `first_name` varchar(45) DEFAULT NULL,
  `last_name` varchar(45) DEFAULT NULL,
  `email` varchar(45) DEFAULT NULL,
  `instructor_detail_id` int DEFAULT NULL,
  PRIMARY KEY (`id`),
  CONSTRAINT `FK_DETAIL` FOREIGN KEY (`instructor_detail_id`) REFERENCES `instructor_detail` (`id`)
);

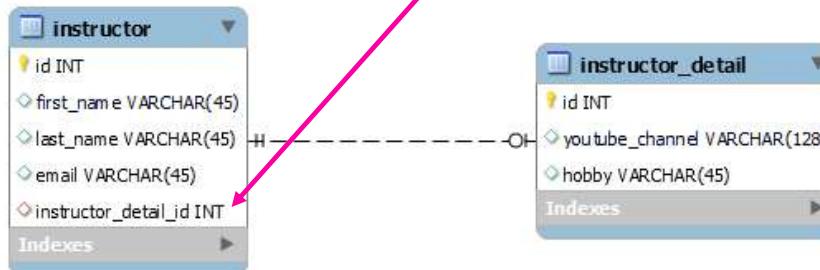
```



วิธีการกำหนด Foreign key

```
CREATE TABLE `instructor` (
  `id` int NOT NULL AUTO_INCREMENT,
  `first_name` varchar(45) DEFAULT NULL,
  `last_name` varchar(45) DEFAULT NULL,
  `email` varchar(45) DEFAULT NULL,
  `instructor_detail_id` int DEFAULT NULL,
  PRIMARY KEY (`id`),
  CONSTRAINT `FK_DETAIL` FOREIGN KEY (`instructor_detail_id`) REFERENCES `instructor_detail` (`id`)
);
```

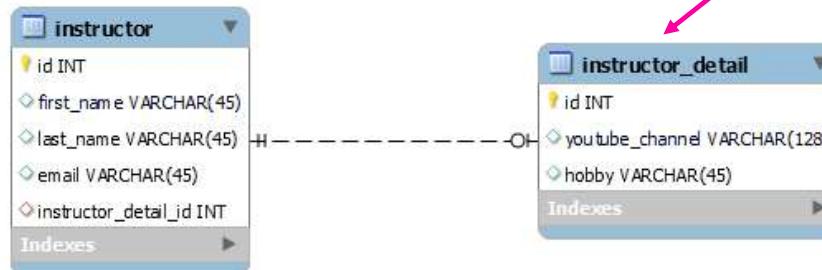
ชื่อคอลัมน์สำหรับใส่
Foreign key



วิธีการกำหนด Foreign key

```
CREATE TABLE `instructor` (
  `id` int NOT NULL AUTO_INCREMENT,
  `first_name` varchar(45) DEFAULT NULL,
  `last_name` varchar(45) DEFAULT NULL,
  `email` varchar(45) DEFAULT NULL,
  `instructor_detail_id` int DEFAULT NULL,
  PRIMARY KEY (`id`),
  CONSTRAINT `FK_DETAIL` FOREIGN KEY (`instructor_detail_id`) REFERENCES `instructor_detail` (`id`)
);
```

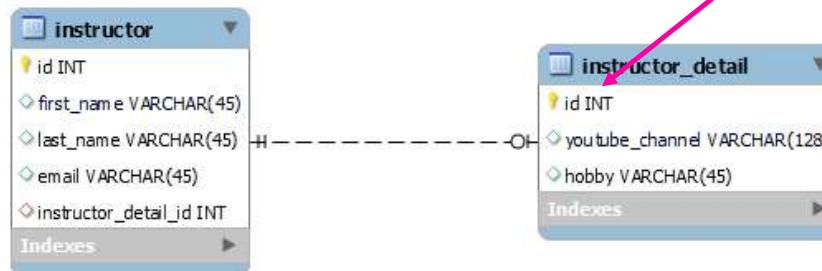
ตารางที่มี Primary key
ที่เราต้องการอ้างอิง



วิธีการกำหนด Foreign key

```
CREATE TABLE `instructor` (
  `id` int NOT NULL AUTO_INCREMENT,
  `first_name` varchar(45) DEFAULT NULL,
  `last_name` varchar(45) DEFAULT NULL,
  `email` varchar(45) DEFAULT NULL,
  `instructor_detail_id` int DEFAULT NULL,
  PRIMARY KEY (`id`),
  CONSTRAINT `FK_DETAIL` FOREIGN KEY (`instructor_detail_id`) REFERENCES `instructor_detail` (`id`)
);
```

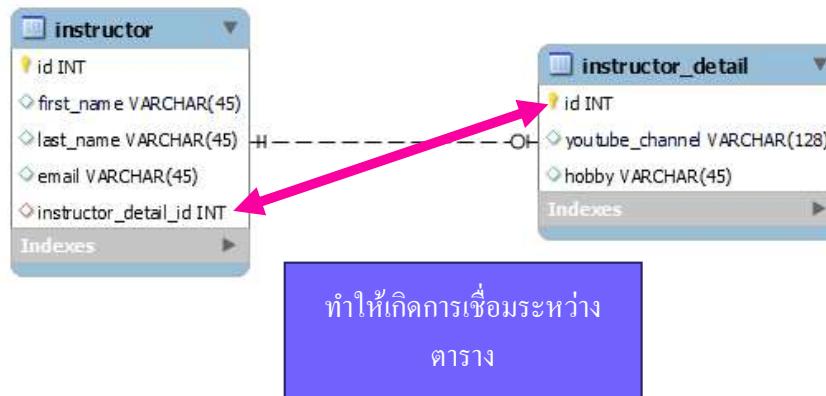
คอลัมน์ของ Primary



วิธีการกำหนด Foreign key

```
CREATE TABLE `instructor` (
  `id` int NOT NULL AUTO_INCREMENT,
  `first_name` varchar(45) DEFAULT NULL,
  `last_name` varchar(45) DEFAULT NULL,
  `email` varchar(45) DEFAULT NULL,
  `instructor_detail_id` int DEFAULT NULL,
  PRIMARY KEY (`id`),
  CONSTRAINT `FK_DETAIL` FOREIGN KEY (`instructor_detail_id`) REFERENCES `instructor_detail` (`id`)
);

```



วิธีการกำหนด Foreign key

- จุดประสงค์ของ Foreign key คือการดำเนินไว้ซึ่งความสัมพันธ์ระหว่างตาราง
 - Referential Integrity
- มันจะป้องกัน Operational ที่จะทำลายความสัมพันธ์ระหว่างตาราง
- จะมีเฉพาะข้อมูลที่ถูกต้องเท่านั้นที่สามารถเข้ามาใส่ในคอลัมน์ Foreign key ได้
 - จะเป็น Foreign key ที่สามารถอ้างถึง Primary key ได้จริง ๆ

วิธีการกำหนด Foreign key – ตัวอย่าง

คอลัมน์ที่เก็บ Foreign key จะเก็บได้เฉพาะ Primary key ที่มีอยู่จริง

<u>id</u>	<u>first_name</u>	<u>last_name</u>	<u>instructor_detail_id</u>
1	Somchai	Rukdee	3
2	Anek	Oreo	1
3	Somboon	Pochana	2

<u>id</u>	<u>youtube_channel</u>	<u>hobby</u>
1	www.youtube.com/oreo_tv	Piano
2	www.youtube.com/somboon_food	Football
3	www.youtube.com/somchai_good_learn	Running

วิธีการกำหนด Foreign key – ตัวอย่าง

คอลัมน์ที่เก็บ Foreign key จะเก็บได้เฉพาะ Primary key ที่มีอยู่จริง

<u>id</u>	<u>first_name</u>	<u>last_name</u>	<u>instructor_detail_id</u>
1	Somchai	Rukdee	3
2	Anek	Oreo	1
3	Somboon	Pochana	4

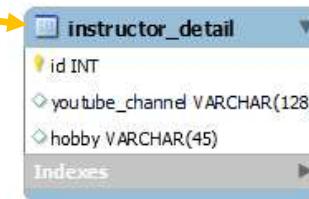
พังทันที่ เพราะ primary key นี้ไม่มีอยู่จริง

<u>id</u>	<u>youtube_channel</u>	<u>hobby</u>
1	www.youtube.com/oreo_tv	Piano
2	www.youtube.com/somboon_food	Football
3	www.youtube.com/somchai_good_learn	Running

One-to-one Mapping

▪ 2. สร้างคลาส InstructorDetail

```
@Entity  
@Table(name = "instructor_detail")  
public class InstructorDetail {  
  
}
```



One-to-one Mapping และ Uni-Directional

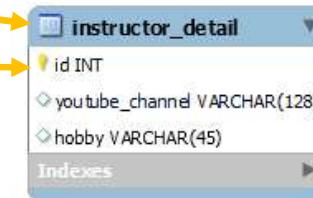
- วิธีทำ

1. สร้างตารางใน Database
2. สร้างคลาส InstructorDetail
3. สร้างคลาส Instructor
4. สร้าง Main App

One-to-one Mapping

▪ 2. สร้างคลาส InstructorDetail

```
@Entity  
@Table(name = "instructor_detail")  
public class InstructorDetail {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column(name = "id")  
    private int id;  
  
}
```



One-to-one Mapping

▪ 2. สร้างคลาส InstructorDetail

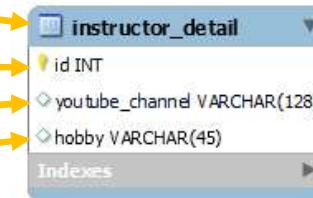
```
@Entity  
@Table(name = "instructor_detail")  
public class InstructorDetail {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column(name = "id")  
    private int id;  
  
    @Column(name = "youtube_channel")  
    private String youtubeChannel;  
  
}
```



One-to-one Mapping

▪ 2. สร้างคลาส InstructorDetail

```
@Entity  
@Table(name = "instructor_detail")  
public class InstructorDetail {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column(name = "id")  
    private int id;  
  
    @Column(name = "youtube_channel")  
    private String youtubeChannel;  
  
    @Column(name = "hobby")  
    private String hobby;  
}
```



One-to-one Mapping

▪ 2. สร้างคลาส InstructorDetail

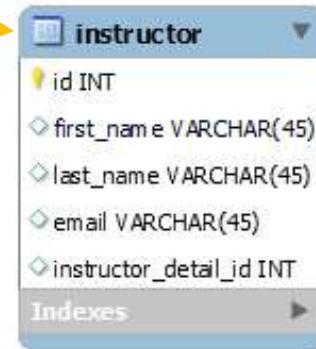
```
@Entity  
@Table(name = "instructor_detail")  
public class InstructorDetail {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column(name = "id")  
    private int id;  
  
    @Column(name = "youtube_channel")  
    private String youtubeChannel;  
  
    @Column(name = "hobby")  
    private String hobby;  
  
    // constructors  
  
    // getters and setters  
}
```



One-to-one Mapping

- 3. สร้างคลาส Instructor

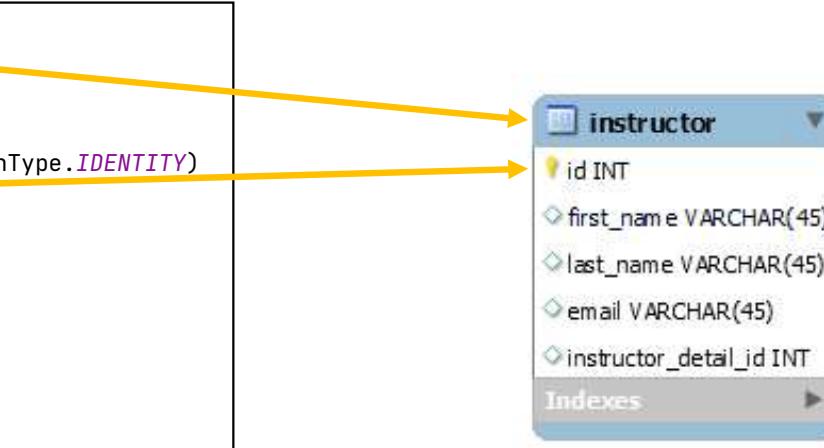
```
@Entity  
@Table(name = "instructor")  
public class Instructor {  
  
}
```



One-to-one Mapping

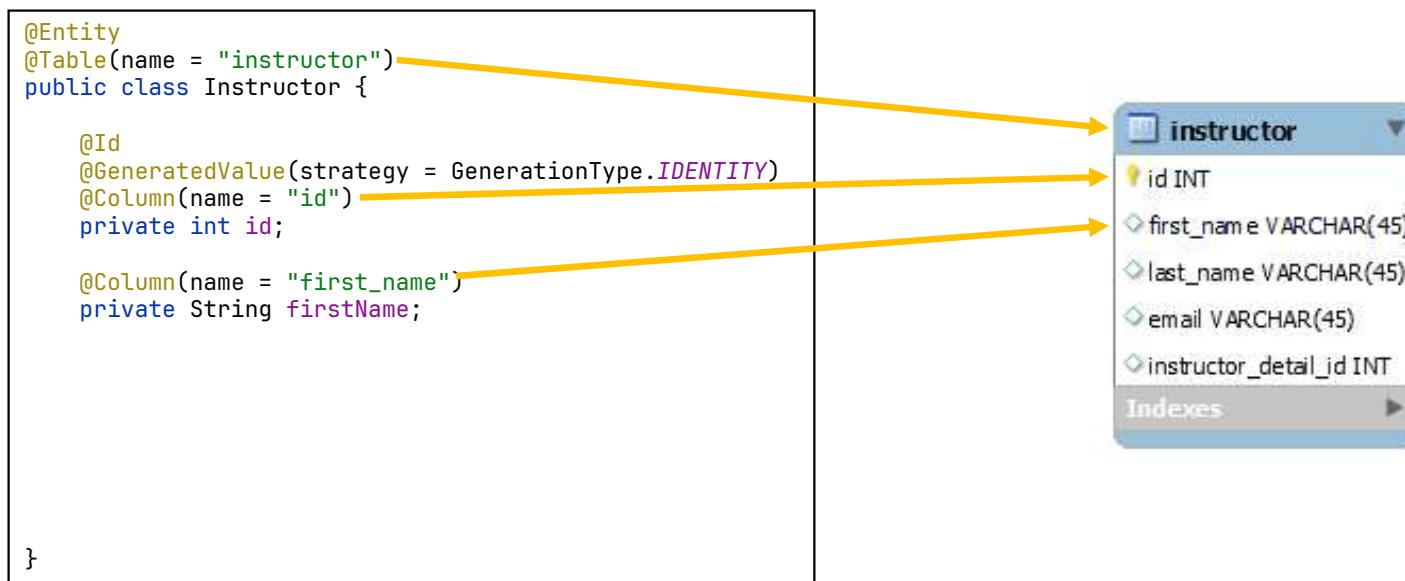
▪ 3. สร้างคลาส Instructor

```
@Entity  
@Table(name = "instructor")  
public class Instructor {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column(name = "id")  
    private int id;  
  
}
```



One-to-one Mapping

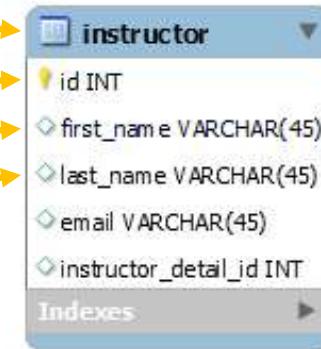
▪ 3. สร้างคลาส Instructor



One-to-one Mapping

▪ 3. สร้างคลาส Instructor

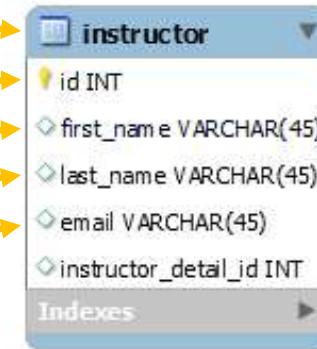
```
@Entity  
@Table(name = "instructor")  
public class Instructor {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column(name = "id")  
    private int id;  
  
    @Column(name = "first_name")  
    private String firstName;  
  
    @Column(name = "last_name")  
    private String lastName;  
  
}
```



One-to-one Mapping

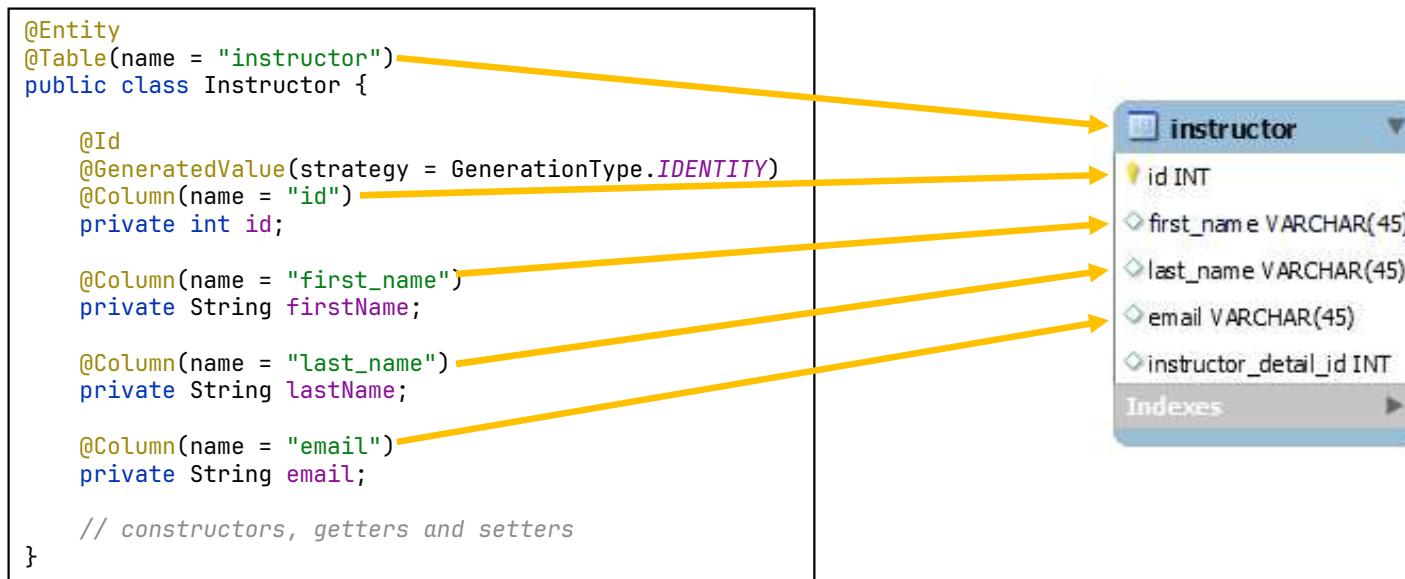
▪ 3. สร้างคลาส Instructor

```
@Entity  
@Table(name = "instructor")  
public class Instructor {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column(name = "id")  
    private int id;  
  
    @Column(name = "first_name")  
    private String firstName;  
  
    @Column(name = "last_name")  
    private String lastName;  
  
    @Column(name = "email")  
    private String email;  
  
}
```



One-to-one Mapping

▪ 3. สร้างคลาส Instructor



One-to-one Mapping

- 3. สร้างคลาส Instructor และ เพิ่ม @OneToOne

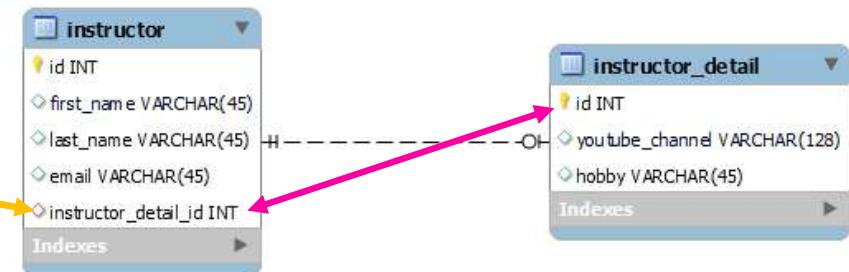
```
@Entity
@Table(name = "instructor")
public class Instructor {

    ...
    @OneToOne
    @JoinColumn(name = "instructor_detail_id")
    private InstructorDetail instructorDetail;
    ...
    // constructors, getters and setters
}
```

One-to-one Mapping

- 3. สร้างคลาส Instructor และ เพิ่ม @OneToOne

```
@Entity  
@Table(name = "instructor")  
public class Instructor {  
  
    ...  
  
    @OneToOne  
    @JoinColumn(name = "instructor_detail_id")  
    private InstructorDetail instructorDetail;  
  
    ...  
  
    // constructors, getters and setters  
}
```

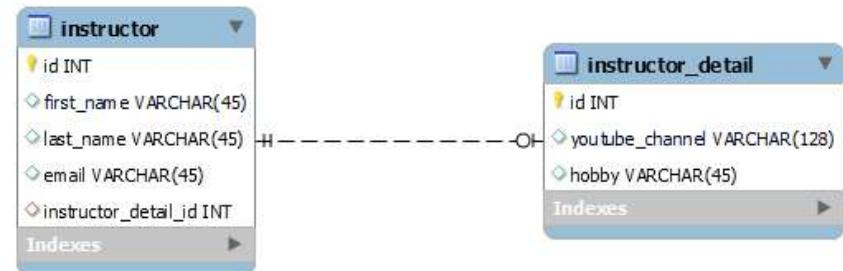


One-to-one Mapping

▪ 3. สร้างคลาส Instructor และเพิ่ม @OneToOne

- มันจะเขียนตารางทั้งสองข้าด้วยกันแบบ @one-to-one
- @JoinColumn คือบอกชื่อคอลัมน์ที่เป็น Foreign Key
- ของตารางที่เป็น Class ในด้านล่าง

```
@Entity  
@Table(name = "instructor")  
public class Instructor {  
  
    ...  
  
    @OneToOne  
    @JoinColumn(name = "instructor_detail_id")  
    private InstructorDetail instructorDetail;  
  
    ...  
  
    // constructors, getters and setters  
}
```

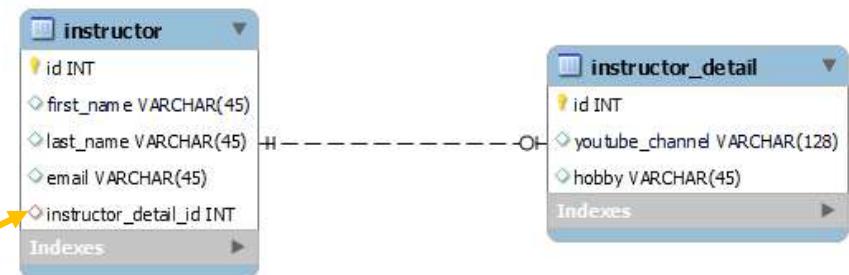


One-to-one Mapping

- 3. สร้างคลาส Instructor และเพิ่ม @OneToOne

- มันจะเชื่อมตารางทั้งสองเข้าด้วยกันแบบ @one-to-one
- **@JoinColumn** คือบอกชื่อคอลัมน์ที่เป็น Foreign Key
- ของตารางที่เป็น Class ในด้านล่าง

```
@Entity  
@Table(name = "instructor")  
public class Instructor {  
  
    ...  
  
    @OneToOne  
    @JoinColumn(name = "instructor_detail_id")  
    private InstructorDetail instructorDetail;  
  
    ...  
  
    // constructors, getters and setters  
}
```

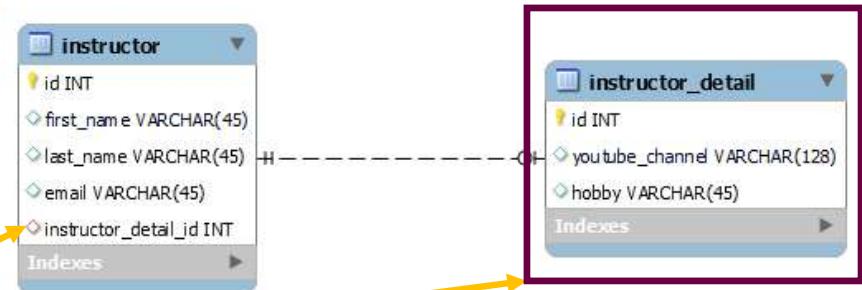


One-to-one Mapping

- 3. สร้างคลาส Instructor และเพิ่ม @OneToOne

- มันจะเขียนตารางทั้งสองเข้าด้วยกันแบบ @one-to-one
- @JoinColumn คือบอกชื่อคอลัมน์ที่เป็น Foreign Key
- ของตารางที่เป็น Class ในด้านล่าง

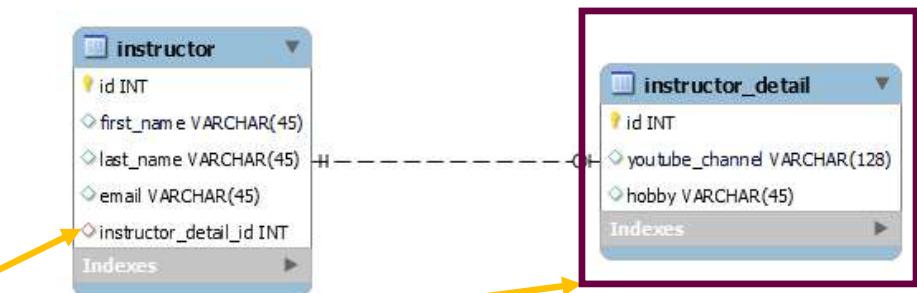
```
@Entity  
@Table(name = "instructor")  
public class Instructor {  
  
    ...  
  
    @OneToOne  
    @JoinColumn(name = "instructor_detail_id")  
    private InstructorDetail instructorDetail;  
  
    ...  
  
    // constructors, getters and setters  
}
```



One-to-one Mapping

- เดี่ยว Hibernate จะเอา `instructor_detail_id` ไปทำในตาราง InstructorDetail ว่า Primary Key ตัวไหน ตรงกับ `instructor_detail_id` นี้ แล้วมันจะเอามาใส่ให้อัตโนมัติ

```
@Entity  
@Table(name = "instructor")  
public class Instructor {  
  
    ...  
  
    @OneToOne  
    @JoinColumn(name = "instructor_detail_id")  
    private InstructorDetail instructorDetail;  
  
    ...  
  
    // constructors, getters and setters  
}
```



การทำงานกับ JPA Entity Object

- Entity Object ก็เป็น instances ที่เก็บอยู่ใน memory ของ Entity classes
- ซึ่ง Entity Object พวจนี้ก็จะเป็นตัวแทนของ Row ใน Database

การทำงานกับ JPA Entity Object

- ชี้ Entity Object พจน์ที่จะเป็นตัวแทนของ Row ใน Database

ตาราง Students ใน Database		
id	first_name	last_name
1	Somchai	Rukdee
2	Anek	Oreo
3	Somboon	Pochana

- Object Entity ที่ชื่อ Student1 จะเปรียบ
เหมือนตัวแทนของ Row ที่ชื่อ Somchai ใน
ตาราง Student
- Object Entity ที่ชื่อ Student2 จะเปรียบ
เหมือนตัวแทนของ Row ที่ชื่อ Anek ในตาราง
Student

```
Student student1 = entityManager.find(Student.class, 1);
Student student2 = entityManager.find(Student.class, 2);

System.out.println(student1.getFirstName()); // Somchai
System.out.println(student2.getFirstName()); // Anek
```

การทำงานกับ JPA Entity Object

- ซึ่งเราสามารถแก้ไขตัวที่เป็น Entity Object ได้

ตาราง Students ใน Database		
id	first_name	last_name
1	Somchai	Rukdee
2	Anek	Oreo
3	Somboon	Pochana

```
Student student1 = entityManager.find(Student.class, 1);
student1.setFirstName("Suchat");
entityManager.persist(student1);
```

การทำงานกับ JPA Entity Object

- ซึ่งเราสามารถแก้ไขตัวที่เป็น Entity Object ได้

ตาราง Students ใน Database		
id	first_name	last_name
1	Somchai	Rukdee
2	Anek	Oreo
3	Somboon	Pochana

- เวลาเราแก้ไข Object Entity ที่ชื่อ student1 ที่เป็นตัวแทน Row ใน Database แล้วเรียก entityManager.persist มันจะยังไม่ Save ทันทีลง Database จนกว่าเราจะมีการ commit/flush

```
Student student1 = entityManager.find(Student.class, 1);
student1.setFirstName("Suchat");
entityManager.persist(student1);
```

การทำงานกับ JPA Entity Object

- ซึ่งเราสามารถแก้ไขตัวที่เป็น Entity Object ได้

ตาราง Students ใน Database		
id	first_name	last_name
1	Somchai	Rukdee
2	Anek	Oreo
3	Somboon	Pochana

- ซึ่งการที่เราใส่ @Transactional มันจะทำเรื่องพากนี้ให้เรา พอหลังจากจบ ฟังก์ชัน มันก็จะมี การ commit/flush ให้

```
@Override  
@Transactional  
public void doSomething(Integer id) {  
    // begin()  
    Student student1 = entityManager.find(Student.class, 1);  
  
    student1.setFirstName("Suchat");  
  
    entityManager.persist(student1);  
    // commit();  
    // flush()  
}
```

การทำงานกับ JPA Entity Object

- ซึ่งเราสามารถแก้ไขตัวที่เป็น Entity Object ได้

ตาราง Students ใน Database		
id	first_name	last_name
1	Suchat	Rukdee
2	Anek	Oreo
3	Somboon	Pochana

- พอเวลาหลังจากจบฟังก์ชันใน Database จึงมีการอัพเดทค่าใน Database

```
@Override  
@Transactional  
public void doSomething(Integer id) {  
    // begin()  
    Student student1 = entityManager.find(Student.class, 1);  
  
    student1.setFirstName("Suchat");  
  
    entityManager.persist(student1);  
    // commit();  
    // flush()  
}
```

Entity Lifecycle

- ชี้ว่า Object Entity มีทั้งหมด 4 สถานะ
 - Managed / Persist
 - Detached
 - Transient / New
 - Removed

Entity Lifecycle

State	Description
Managed / Persist	เป็น State ที่เป็นเหมือนตัวแทนใน Database เวลามีการแก้ไขจะมีการ Track ไว้เสมอว่าเปลี่ยนแปลงตรงไหนบ้าง (แต่ยังไม่เปลี่ยนใน Database จนกว่าจะมีการ flush/commit) เช่น เรา findByID มาจาก Database ตัวนั้นก็จะมีสถานะเป็น Persist
Removed	เป็น State คล้าย ๆ กับ Persist แต่แค่จะถูก Marked ไว้ว่าเป็นสถานะ Deletion และจะยังไม่ถูกลบใน Database จนกว่ามีการ flush/commit เช่นกัน
Transient / New	เป็น State ที่ยังไม่มีความเกี่ยวข้องกับ EntityManager เช่น การสร้าง Entity Object ขึ้นมาใหม่ และยังไม่ได้ทำอะไรกับ Object นี้
Detached	เป็น State สำหรับ entity object ที่ถูกตัดขาดจาก EntityManager เช่น สมมุติเรามีการ close ตัว EntityManager ตัว entity object ทุกตัวก็จะเป็น Detached

Entity Lifecycle – ตัวอย่าง

```
@Override  
@Transactional  
public void createNewStudent() {  
    Student newStudent = new Student();  
  
    newStudent.setFirstName("New Student");  
  
    entityManager.persist(newStudent);  
  
    newStudent.setFirstName("Suchat");  
}
```

ตอนนี้เพิ่งสร้างใหม่ เพราะฉะนั้นมันจะไม่ได้แทน Row ใน Database ตัวไหนเลย และยังไม่มีความสัมพันธ์กับ entityManager มาก่อนด้วย

ตอนนี้ newStudent จึงเป็นสถานะ Transient / New

State	Description
Transient / New	เป็น State ที่ยังไม่มีความเกี่ยวข้องกับ EntityManager เช่น การสร้าง Entity Object ขึ้นมาใหม่ และยังไม่ได้ทำอะไรกับ Object นี้

Entity Lifecycle – ตัวอย่าง

```
@Override  
@Transactional  
public void createNewStudent() {  
    Student newStudent = new Student();  
  
    newStudent.setFirstName("New Student");  
  
    entityManager.persist(newStudent);  
  
    newStudent.setFirstName("Suchat");  
}
```

ตอนแก้ไขค่าตอนนั้นก็ยังคงเป็นสถานะ Transient/New

State	Description
Transient / New	เป็น State ที่ยังไม่มีความเกี่ยวข้องกับ EntityManager เช่น การสร้าง Entity Object ขึ้นมาใหม่ และยังไม่ได้ทำอะไรกับ Object นี้

Entity Lifecycle – ตัวอย่าง

```
@Override  
@Transactional  
public void createNewStudent() {  
    Student newStudent = new Student();  
  
    newStudent.setFirstName("New Student");  
  
    entityManager.persist(newStudent);  
  
    newStudent.setFirstName("Suchat");  
}
```

การเรียก `entityManager.persist` เป็นการเปลี่ยนสถานะของ `newStudent` จาก `transient` เป็น `persist/managed` เพราะฉะนั้น เมื่อมีการเปลี่ยนแปลงอะไรก็ตามมันจะถูก Track ทั้งหมด และเมื่อมีการ `commit/flush` ก็จะมีการบันทึก Database

State	Description
Managed / Persist	เป็น State ที่เป็นเหมือนตัวแทนใน Database เวลามีการแก้ไขอะไรก็ตาม มีการ Track ไว้เสมอว่าเปลี่ยนแปลงตรงไหนบ้าง (แต่ยังไม่เปลี่ยนใน Database จนกว่าจะมีการ flush/commit) เช่น เรา <code>findById</code> มาจาก Database ตัวนั้นก็จะมีสถานะเป็น Persist

Entity Lifecycle – ตัวอย่าง

```
@Override  
@Transactional  
public void createNewStudent() {  
    Student newStudent = new Student();  
  
    newStudent.setFirstName("New Student");  
  
    entityManager.persist(newStudent);  
  
    newStudent.setFirstName("Suchat");  
}
```

เรามีการแก้ไขอีกครั้งนึงโดยเปลี่ยนใจมาใช้ชื่อ “New Student” แล้วแต่เปลี่ยนเป็น “Suchat” แทน และเนื่องจากตอนนี้มันเป็นสถานะ Persist เพราะฉะนั้นการเปลี่ยนแปลงอันนี้ก็จะถูก Track ไว้ด้วย

State	Description
Managed / Persist	เป็น State ที่เป็นเหมือนตัวแทนใน Database เวลามีการแก้ไขอะไรจะมีการ Track ไว้เสมอว่าเปลี่ยนแปลงตรงไหนบ้าง (แต่ยังไม่เปลี่ยนใน Database จนกว่าจะมีการ flush/commit) เช่น เรา findByID จาก Database ตัวนั้นก็จะมีสถานะเป็น Persist

Entity Lifecycle – ตัวอย่าง

```
@Override  
@Transactional  
public void createNewStudent() {  
    Student newStudent = new Student();  
  
    newStudent.setFirstName("New Student");  
  
    entityManager.persist(newStudent);  
  
    newStudent.setFirstName("Suchat");  
}
```

หลังจากจบ Function นี้ ก็จะมีการ commit/flush ให้ (เนื่องจากเราใส่ @Transactional ไว้) ข้อมูลของ entity object ที่มีสถานะเป็น persist ก็จะถูกทำการเปลี่ยนแปลงทั้งหมดบนทีเกลน Database

ตาราง Students ใน Database		
<i>id</i>	<i>first_name</i>	<i>last_name</i>
1	Somchai	Rukdee
2	Anek	Oreo
3	Somboon	Pochana
4	Suchat	-

Entity Lifecycle – ตัวอย่างที่ 2

```
@Override  
@Transactional  
public void queryStudent() {  
    Student student = entityManager.find(Student.class, 1);  
  
    student.setFirstName("Somwang");  
}
```

เรามีการ findByld โดย entity object ที่ได้จะเป็น row ที่มี id = 1 (row ของ somchai) ซึ่งการที่เรา query ออกมาจาก database จะทำให้ entity object ตัวนั้นมีสถานะเป็น persist

ตาราง Students ใน Database

id	first_name	last_name
1	Somchai	Rukdee
2	Anek	Oreo
3	Somboon	Pochana
4	Suchat	-

Entity Lifecycle – ตัวอย่างที่ 2

```
@Override  
@Transactional  
public void queryStudent() {  
    Student student = entityManager.find(Student.class, 1);  
  
    student.setFirstName("Somwang");  
}
```

และเราแก้เปลี่ยนชื่อของ Entity Object ตัวนี้ ซึ่งสถานะเป็น persist และเนื่องจากตอนนี้มันเป็นสถานะ Persist เพราะฉะนั้นการเปลี่ยนแปลงอันนี้ก็จะถูก Track ไว้ด้วย

ตาราง Students ใน Database

id	first_name	last_name
1	Somchai	Rukdee
2	Anek	Oreo
3	Somboon	Pochana
4	Suchat	-

Entity Lifecycle – ตัวอย่างที่ 2

```
@Override  
@Transactional  
public void queryStudent() {  
    Student student = entityManager.find(Student.class, 1);  
  
    student.setFirstName("Somwang");  
}
```

หลังจากจบ Function นี้ ก็จะมีการ commit/flush ให้ (เนื่องจากเราระบุ @Transactional ไว้) ข้อมูลของ entity object ที่มีสถานะเป็น persist ก็จะถูกทำการเปลี่ยนแปลง ทั้งหมดบันทึกลง Database

ตาราง Students ใน Database

id	first_name	last_name
1	Somwang	Rukdee
2	Anek	Oreo
3	Somboon	Pochana
4	Suchat	-

Entity Lifecycle – ตัวอย่างที่ 3

```
@Override  
@Transactional  
public void testDetachStudent() {  
    Student student = entityManager.find(Student.class, 2);  
  
    entityManager.detach(student);  
  
    student.setFirstName("Napon");  
}
```

เรา Query ตัว row ของ Anek ออกมาซึ่งเราตอนนี้สถานะเป็น Persist

ตาราง Students ใน Database		
id	first_name	last_name
1	Somwang	Rukdee
2	Anek	Oreo
3	Somboon	Pochana
4	Suchat	-

Entity Lifecycle – ตัวอย่างที่ 3

```
@Override  
@Transactional  
public void testDetachStudent() {  
    Student student = entityManager.find(Student.class, 2);  
  
    entityManager.detach(student);  
  
    student.setFirstName("Napon");  
}
```

มีการเรียกใช้ `entityManager.detach` ซึ่งเป็นการเปลี่ยนสถานะของ Entity จาก persist ให้เป็น detach เพราะฉะนั้น entity object จะไม่เกี่ยวข้องกับ row ของ Anek อีกต่อไป

ตาราง Students ใน Database		
id	first_name	last_name
1	Somwang	Rukdee
2	Anek	Oreo
3	Somboon	Pochana
4	Suchat	-

Entity Lifecycle – ตัวอย่างที่ 3

```
@Override  
@Transactional  
public void testDetachStudent() {  
    Student student = entityManager.find(Student.class, 2);  
  
    entityManager.detach(student);  
  
    student.setFirstName("Napon");
```

หลังจากนั้นเราจะมีการแก้ไขเป็นชื่อเป็น Napon

ตาราง Students ใน Database		
id	first_name	last_name
1	Somwang	Rukdee
2	Anek	Oreo
3	Somboon	Pochana
4	Suchat	-

Entity Lifecycle – ตัวอย่างที่ 3

```
@Override  
@Transactional  
public void testDetachStudent() {  
    Student student = entityManager.find(Student.class, 2);  
  
    entityManager.detach(student);  
  
    student.setFirstName("Napon");  
}
```

หลังจากจบ Function นี้ ก็จะมีการ commit/flush ให้ ข้อมูลของ entity object ที่มีสถานะเป็น persist ก็จะถูกออกจากเปลี่ยนแปลงทั้งหมดบันทึกลง Database

แต่เนื่องจาก entity object ตัวนี้มีสถานะเป็น detach ไปเรียบร้อยแล้ว เพราะฉะนั้นการเปลี่ยนแปลงใด ๆ ก็ตามของ student จึงไม่มีการบันทึกลง Database

ตาราง Students ใน Database

id	first_name	last_name
1	Somwang	Rukdee
2	Anek	Oreo
3	Somboon	Pochana
4	Suchat	-

Entity Lifecycle – ตัวอย่างที่ 4

```
@Override  
@Transactional  
public void delete() {  
    Student student = entityManager.find(Student.class, 2);  
  
    entityManager.remove(student);  
}
```

มีการ Query ออกมานะเนื่องจากมีสถานะเป็น Persist

ตาราง Students ใน Database		
id	first_name	last_name
1	Somwang	Rukdee
2	Anek	Oreo
3	Somboon	Pochana
4	Suchat	-

Entity Lifecycle – ตัวอย่างที่ 4

```
@Override  
@Transactional  
public void delete() {  
    Student student = entityManager.find(Student.class, 2);  
  
    entityManager.remove(student);  
}
```

มีการเรียกใช้ `entityManager.remove` ซึ่งเป็นการเปลี่ยนสถานะให้เป็น Removed

ตาราง Students ใน Database

<code>id</code>	<code>first_name</code>	<code>last_name</code>
1	Somwang	Rukdee
2	Anek	Oreo
3	Somboon	Pochana
4	Suchat	-

Entity Lifecycle – ตัวอย่างที่ 4

```
@Override  
@Transactional  
public void delete() {  
    Student student = entityManager.find(Student.class, 2);  
  
    entityManager.remove(student);  
}
```

หลังจากจบ Function นี้ ก็จะมีการ commit/flush ให้
แต่ไม่ใช่พำนัช stations persist เท่านั้นที่มีการบันทึกการ
เปลี่ยนแปลงลง Database สถานะ Removed ก็จะบันทึกลง
Database เมื่อcionกันเมื่อมีการ commit/flush

ตาราง Students ใน Database

id	first_name	last_name
1	Somwang	Rukdee
2	Anek	Oreo
3	Somboon	Pochana
4	Suchat	-

Entity Lifecycle – ตัวอย่างที่ 4

```
@Override  
@Transactional  
public void delete() {  
    Student student = entityManager.find(Student.class, 2);  
  
    entityManager.remove(student);  
}
```

หลังจากจบ Function นี้ ก็จะมีการ commit/flush ให้
แต่ไม่ใช่พำนัชสถานะ persist เท่านั้นที่มีการบันทึกการ
เปลี่ยนแปลงลง Database สถานะ Removed ก็จะบันทึกลง
Database เมื่อcionกันเมื่อมีการ commit/flush

ตาราง Students ใน Database

id	first_name	last_name
1	Somwang	Rukdee
3	Somboon	Pochana
4	Suchat	-

Entity Lifecycle – ตัวอย่างที่ 5

```
@Override  
@Transactional  
public void testMerge() {  
    Student student = entityManager.find(Student.class, 1);  
  
    entityManager.detach(student);  
  
    student.setFirstName("Tongchai");  
  
    entityManager.merge(student);  
}
```

มีการ Query ตัว Row ของ Somwang ออกมานะเป็น Persist

ตาราง Students ใน Database		
id	first_name	last_name
1	Somwang	Rukdee
3	Somboon	Pochana
4	Suchat	-

Entity Lifecycle – ตัวอย่างที่ 5

```
@Override  
@Transactional  
public void testMerge() {  
    Student student = entityManager.find(Student.class, 1);  
  
    entityManager.detach(student);  
  
    student.setFirstName("Tongchai");  
  
    entityManager.merge(student);  
}
```

มีการเปลี่ยนให้เป็นสถานะ Detach

ตาราง Students ใน Database

<i>id</i>	<i>first_name</i>	<i>last_name</i>
1	Somwang	Rukdee
3	Somboon	Pochana
4	Suchat	-

Entity Lifecycle – ตัวอย่างที่ 5

```
@Override  
@Transactional  
public void testMerge() {  
    Student student = entityManager.find(Student.class, 1);  
  
    entityManager.detach(student);  
  
    student.setFirstName("Tongchai");  
  
    entityManager.merge(student);  
}
```

เปลี่ยนชื่อให้เป็น Tongchai

ตาราง Students ใน Database		
id	first_name	last_name
1	Somwang	Rukdee
3	Somboon	Pochana
4	Suchat	-

Entity Lifecycle – ตัวอย่างที่ 5

```
@Override  
@Transactional  
public void testMerge() {  
    Student student = entityManager.find(Student.class, 1);  
  
    entityManager.detach(student);  
  
    student.setFirstName("Tongchai");  
  
    entityManager.merge(student);  
}
```

มีการเรียก `entityManager.merge` ซึ่งทำให้ตัว entity object จากสถานะ detach กลับเป็น persist

ตาราง Students ใน Database		
<code>id</code>	<code>first_name</code>	<code>last_name</code>
1	Somwang	Rukdee
3	Somboon	Pochana
4	Suchat	-

Entity Lifecycle – ตัวอย่างที่ 5

```
@Override  
@Transactional  
public void testMerge() {  
    Student student = entityManager.find(Student.class, 1);  
  
    entityManager.detach(student);  
  
    student.setFirstName("Tongchai");  
  
    entityManager.merge(student);  
}
```

หลังจากจบ Function นี้ ก็จะมีการ commit/flush ให้
ข้อมูลของ entity object ที่มีสถานะเป็น persist ก็จะถูกเอา
การเปลี่ยนแปลงทั้งหมดบันทึกลง Database

ตาราง Students ใน Database

id	first_name	last_name
1	Tongchai	Rukdee
3	Somboon	Pochana
4	Suchat	-

Entity Lifecycle – ตัวอย่างที่ 6

มีการ Query ตัว Row ของ Somwang ออกรมา (สถานะเป็น Persist)

```
@Override  
@Transactional  
public void delete(Integer id) {  
    Student student = entityManager.find(Student.class, 1);  
  
    student.setFirstName("Hello");  
  
    System.out.println(student.getFirstName()); // Hello  
  
    entityManager.refresh(student);  
  
    System.out.println(student.getFirstName()); // Tongchai  
}
```

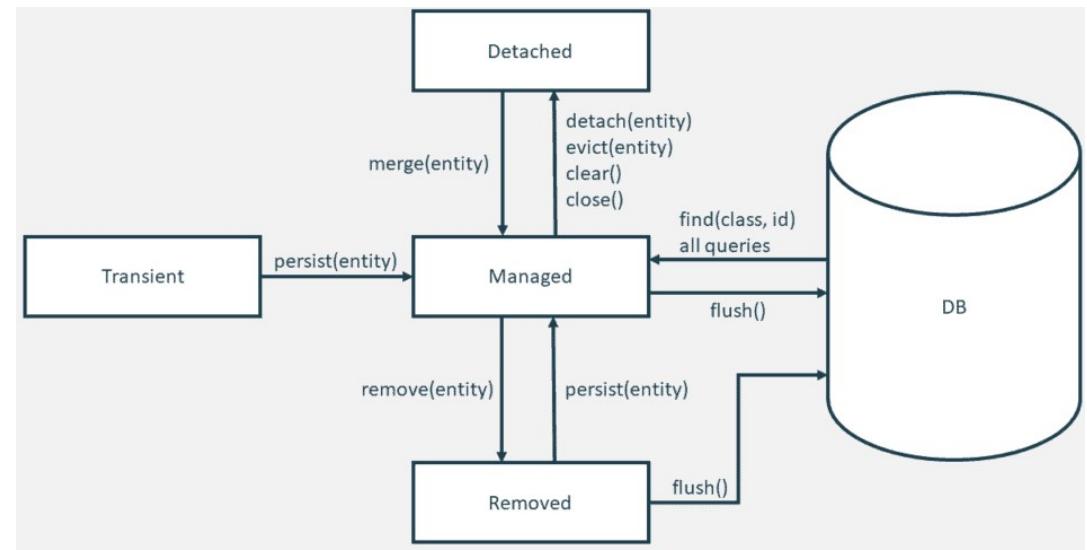
Entity Lifecycle – ตัวอย่างที่ 6

```
@Override  
@Transactional  
public void delete(Integer id) {  
    Student student = entityManager.find(Student.class, 1);  
  
    student.setFirstName("Hello");  
  
    System.out.println(student.getFirstName()); // Hello  
  
    entityManager.refresh(student);  
  
    System.out.println(student.getFirstName()); // Tongchai  
}
```

เป็นการ Reload/synch ตัว Object ด้วยข้อมูลจาก Database

Entity Lifecycle

- สรุป Object Entity มีทั้งหมด 4 สถานะ
 - Detached
 - Transient / New
 - Managed / Persist
 - Removed



แหล่ง: <https://thorben-janssen.com/wp-content/uploads/2020/07/Lifecycle-Model-1024x576.png>

Entity Lifecycle

▪ Operation / Function

Operation	Description
<code>entityManager.detach</code>	แปลงสถานะของ Entity object ให้เป็น detach
<code>entityManager.merge</code>	ถ้าสถานะของ Entity object เป็น detach ฟังก์ชัน merge จะช่วยให้มันกลับเป็น persist
<code>entityManager.persist</code>	แปลงสถานะของ Entity object ให้เป็น persist (แต่จะบันทึกลง database เมื่อมีการ commit/flush)
<code>entityManager.remove</code>	แปลงสถานะของ Entity object ให้เป็น removed (แต่จะมีการลบใน database เมื่อมีการ commit/flush)
<code>entityManager.refresh</code>	reload / synch ตัว entity object ด้วยข้อมูลจาก Database

Cascade

- ทวนอีกครั้ง: เราสามารถใช้ cascade operation ได้
- มันจะทำ Operation ให้กับ entity ที่มีความสัมพันธ์กับ entity นั้น



Cascade

- ทวนอีกครั้ง: เราสามารถใช้ cascade operation ได้
- มันจะทำ Operation ให้กับ entity ที่มีความสัมพันธ์กับ entity นั้น



Cascade Delete

ตาราง: instructor

id	first_name	last_name	instructor_detail_id
1	Somchai	Rukdee	3
2	Anek	Oreo	1
3	Somboon	Pochana	2

คอลัมน์
Foreign key

เรา Delete ตัว instructor Somchai
แต่มันจะลบตัว instructor detail ของ
Somchai ไปด้วย (ในกรณีที่เราใช้ cascade
delete)

ตาราง: instructor_detail

id	youtube_channel	hobby
1	www.youtube.com/oreo_tv	Piano
2	www.youtube.com/somboon_food	Football
3	www.youtube.com/somchai_good_learn	Running

@OneToOne – Cascade Types

- เราสามารถใช้ Cascade Types กับ Entity ที่มีความสัมพันธ์กันได้

Operation	Description
PERSIST	ถ้า Entity ถูก persist ตัว related entity ก็จะถูก persist
REMOVED	ถ้า Entity ถูก removed ตัว related entity ก็จะถูก removed
REFRESH	ถ้า Entity ถูก refresh ตัว related entity ก็จะถูก refresh
DETACH	ถ้า Entity ถูก detach ตัว related entity ก็จะถูก detach
MERGE	ถ้า Entity ถูก merge ตัว related entity ก็จะถูก merge
ALL	เอาทุก Cascade Type ข้างบน

การกำหนด Cascade Types

```
@Entity  
@Table(name = "instructor")  
public class Instructor {  
  
    ...  
  
    @OneToOne(cascade = CascadeType.ALL)  
    @JoinColumn(name = "instructor_detail_id")  
    private InstructorDetail instructorDetail;  
  
    // constructors, getters and setters  
}
```

โดย Default แล้ว มันจะไม่มี cascade operation

การกำหนด Cascade Types หลายอัน

```
@OneToOne(cascade = {  
    CascadeType.DETACH,  
    CascadeType.MERGE,  
    CascadeType.PERSIST,  
    CascadeType.REFRESH,  
    CascadeType.REMOVE  
})
```

One-to-one Mapping และ Uni-Directional

- วิธีทำ

1. สร้างตารางใน Database
2. สร้างคลาส InstructorDetail
3. สร้างคลาส Instructor
4. สร้าง Main App

One-to-one Mapping และ Uni-Directional

▪ 4. สร้าง Main App

- เราจะใช้ Command Line App เมื่อไอน์เดิม
- เพื่อให้เรา Focus เนพะ JPA/Hibernate ได้ง่าย



One-to-one Mapping และ Uni-Directional

- สร้าง DAO interface

```
public interface AppDAO {  
    void save(Instructor instructor);  
}
```



One-to-one Mapping และ Uni-Directional

▪ สร้าง DAO Implementation

```
@Transactional  
public class AppDAOImpl implements AppDAO {  
  
    private EntityManager entityManager;  
  
    @Autowired  
    public AppDAOImpl(EntityManager entityManager){  
        this.entityManager = entityManager;  
    }  
  
    @Override  
    @Transactional  
    public void save(Instructor instructor) {  
        entityManager.persist(instructor);  
    }  
}
```

เมื่อเรา Persist ตัว instructor แล้ว มันจะ persist ตัว instructor detail ด้วย เพราะเราใช้เป็น CascadeType.ALL

Instructor

Instructor
Detail

One-to-one Mapping และ Uni-Directional

- Update ต่อ MainApp

```
@SpringBootApplication
public class FirstSpringBootApplication {

    public static void main(String[] args) {
        SpringApplication.run(FirstSpringBootApplication.class, args);
    }

    @Bean
    public CommandLineRunner commandLineRunner(AppDAO appDAO) {
        return runner -> {
            createInstructor(appDAO);
        };
    }
    ...
}

private void createInstructor(AppDAO appDAO) {
    // create the instructor
    Instructor instructor = new Instructor("Somchai",
        "Jaidee", "Somchai.j@hotmail.com");

    // create the instructor detail
    InstructorDetail instructorDetail = new InstructorDetail(
        "youtube.com/Somchai.j",
        "Learn coding");

    // associate the objects
    tempInstructor.setInstructorDetail(instructorDetail);

    // save the instructor
    System.out.println("Saving instructor: " + tempInstructor);
    appDAO.save(instructor);
    System.out.println("Done!");
}
```

One-to-one Mapping และ Uni-Directional

▪ Update ตัว MainApp

```
@SpringBootApplication
public class FirstSpringBootApplication {

    public static void main(String[] args) {
        SpringApplication.run(FirstSpringBootApplication.class, args);
    }

    @Bean
    public CommandLineRunner commandLineRunner(AppDAO appDAO) {
        return runner -> {
            createInstructor(appDAO);
        };
    }
    ...
}
```

ตัวนี้จะ persist InstructorDetail ด้วย
 เพราะว่าเราใช้ CascadeType.ALL

ใน AppDAO เราเรียกใช้
 entityManager.persist(...)

```
private void createInstructor(AppDAO appDAO) {
    // create the instructor
    Instructor instructor = new Instructor("Somchai",
        "Jaidee", "Somchai.j@hotmail.com");

    // create the instructor detail
    InstructorDetail instructorDetail = new InstructorDetail(
        "youtube.com/Somchai.j",
        "Learn coding");

    // associate the objects
    tempInstructor.setInstructorDetail(instructorDetail);

    // save the instructor
    System.out.println("Saving instructor: " + tempInstructor);
    appDAO.save(instructor);
    System.out.println("Done!");
}
```

One-to-one Mapping

Mini-workshop: one-to-one mappings

Ref: lab-s6-l1