Hibernate Inheritance

Session 5

Version 1.2

September 2018

Agenda

- **Introduction**
- Table per Class Hierarchy
- Table per Subclass
- Mixing Table per Class Hierarchy with Table per Subclass
- Table per Concrete Class

Representation in Schema

- The inheritance relation supports three strategies for representation in schema:
  - Table per class hierarchy
  - Table per subclass
  - Table per concrete class

- Each of these techniques has different costs and benefits

Agenda

- Introduction
- **Table per Class Hierarchy**
- Table per Subclass
- Mixing Table per Class Hierarchy with Table per Subclass
- Table per Concrete Class

Table Per Class (1/4)

- A single table for the class hierarchy
- Discriminator column contains key to identify the base type
  - Pros
    - Offers best performance even for in the deep hierarchy since single select may suffice
    - polymorphic and non-polymorphic queries perform well
    - Ad-hoc reporting is possible without complex joins or unions
    - Schema evolution is straightforward
  - Cons
    - Changes to members of the hierarchy require column to be altered, added or removed from the table
    - Database is not normalized
    - Not null constraint can’t be applied to the columns for properties declared by subclasses

Table Per Class (2/4)

- Suppose we have an interface `Payment` with the implementors namely, `CreditCardPayment`, `CashPayment`, and `ChequePayment`
- Following class diagram shows the table per class hierarchy mapping:

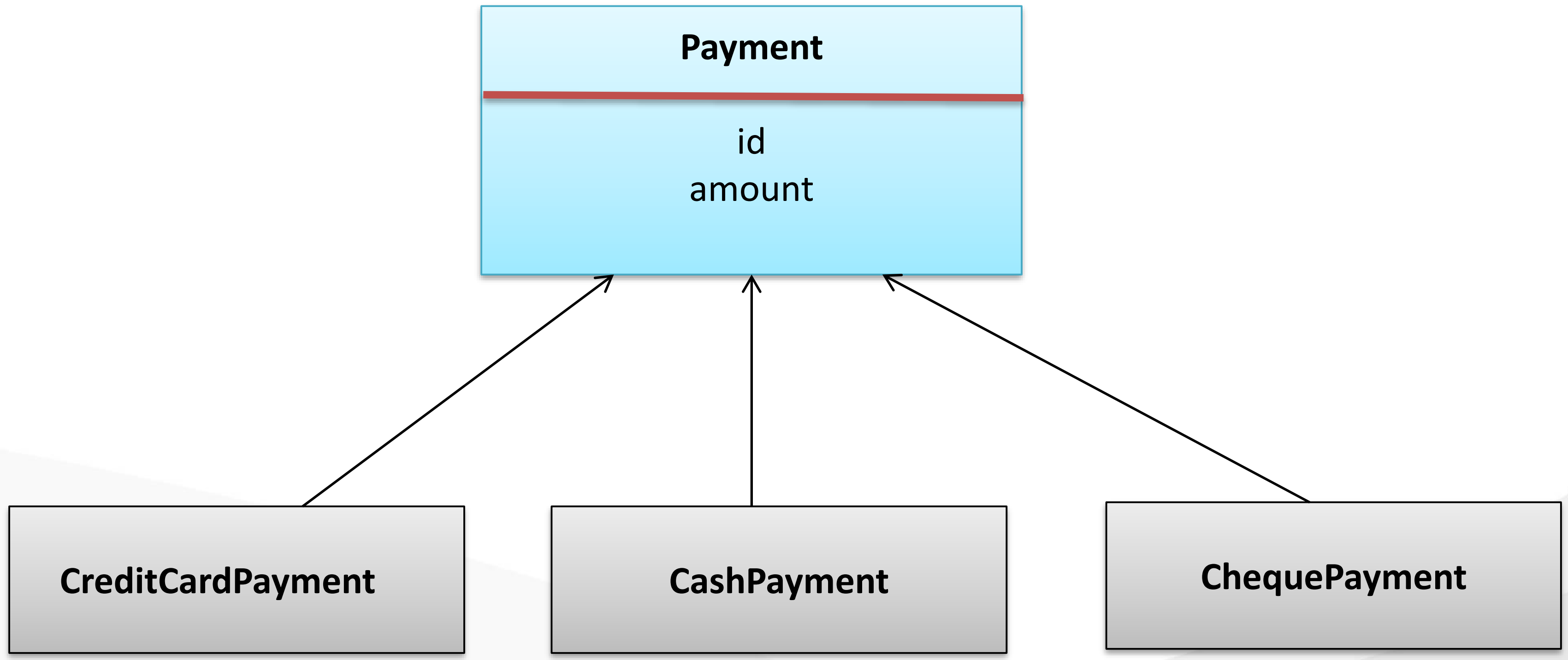


Table Per Class (3/4)

- Following code snippet shows the mapping for table per class hierarchy:

```
<class name ="Payment" table=PAYMENT",
<id name="id" type="long" column="PAYMENT_ID">
<generator class="native" />
</id>  I
<discriminator column="PAYMENT_TYPE" type="string"/>
</<property name="amount" column="AMOUNT"/>
...
<subclass name="CreditCardPayment" discriminator-value="CREDIT">
<property name="creditCardtype" column="CCTYPE" />
...
</subclass>
<subclass name="CashPayment" discriminator-value="CASH">
...
</subclass>
<subclass name="ChequePayment" discriminator-value="CHEQUE">
...
</subclass>
</class>
```

- The **PAYMENT_TYPE** represents the discriminator column
- Columns declared by the subclasses, such as CCTYPE, cannot have NOT NULL constraints

CitiusTech

Table Per Class (4/4)

- Following figure shows the table representation of the table per class hierarchy in the database:

PAYMENT	
PK	**PAYMENT_ID**
	PAYMENT_TYPE AMOUNT CCTYPE DENOM

CitiusTech

Agenda

- Introduction
- Table per Class Hierarchy
- **Table per Subclass**
- Mixing Table per Class Hierarchy with Table per Subclass
- Table per Concrete Class

Table Per Subclass (1/3)

- One table for each class in the hierarchy

  Foreign key relationship exists between common table and subclass table
  - Pros
    - Does not require complex changes to the schema when a single parent class is modified
    - Works well with shallow hierarchy
  - Cons
    - Can result in poor performance – as hierarchy grows, the number of joins required to construct a leaf class also grows
- How to define the mapping ?
  - Use `<joined-subclass>` element with extends attribute

Table Per Subclass (2/3)

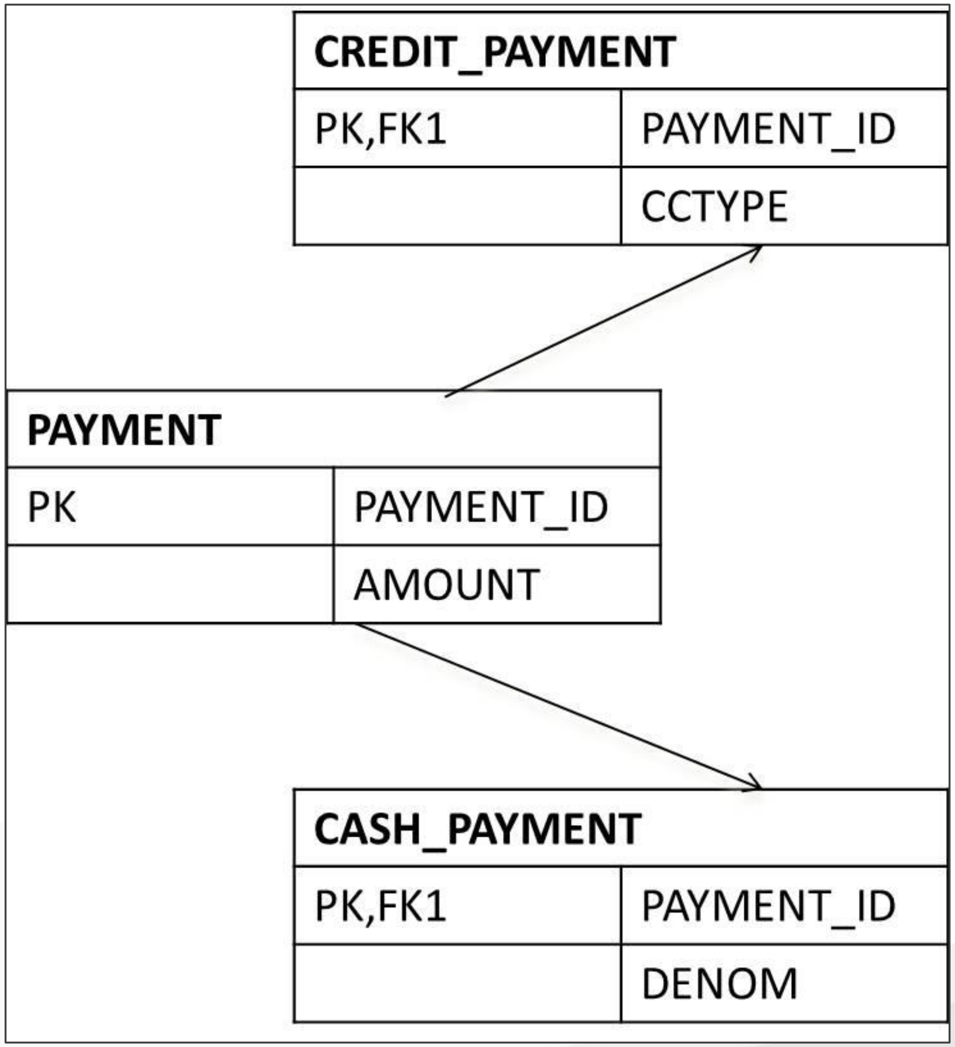
- Following code snippet shows the table per subclass hierarchy:

```
<class name ="Payment" table=PAYMENT",
<id name="id" type="long" column="PAYMENT_ID">
<generator class="native" />
</id>
</<property name="amount" column="AMOUNT"/>
...
<joined-subclass name="CreditCardPayment" table="CREDIT_PAYMENT">
    <key column="PAYMENT_ID"/>
<property name="creditCardtype" column="CCTYPE" />
...
</joined-subclass>
<joined-subclass name="CashPayment" table="CASH_PAYMENT">
    <key column="PAYMENT_ID"/>
...
</joined-subclass>
<joined-subclass name="ChequePayment" table="CHEQUE_PAYMENT">
    <key column="PAYMENT_ID"/>
...
</joined-subclass>
</class>
```

- Four tables are required
- The three subclass tables have primary key associations to the superclass table so the relational model is actually a one-to-one association

Table Per Subclass (3/3)

- Following figure shows the table per subclass hierarchy schema representation in the database:



Agenda

- Introduction
- Table per Class Hierarchy
- Table per Subclass
- **Mixing Table per Class Hierarchy with Table per Subclass**
- Table per Concrete Class

Mixing Table Per Class and Table Per Subclass

- You can even mix the table per hierarchy and table per subclass strategies using the following approach:

```
<class name="Payment" table="PAYMENT">
   <id name="id" type=long" column="PAYMENT_ID">
    <generator class="native"/>
   </id>
    <discriminator column = "PAYMENT_TYPE" type="string"
<property name="amount" column="AMOUNT" />
...
   <subclass name="CreditCardPayment" discriminator-value="CREDIT">
      <join table="CREDIT_PAYMENT">
          <property name="creditCardType" column="CCTYPE"/>
          ...
      </join>
    </subclass>
   <subclass name="CashPayment" discriminator-value="CASH">
...
   </subclass>
  <subclass name="ChequePayment" discriminator-value="CHEQUE">
...
   </subclass>
</class>
```

- For any of these mapping strategies, a polymorphic association to the root Payment class is mapped using<many-to-one>

```
<many-to-one name=“payment” column=“PAYMENT_ID” class=“Payment”
```

Agenda

- Introduction
- Table per Class Hierarchy
- Table per Subclass
- Mixing Table per Class Hierarchy with Table per Subclass
- **Table per Concrete Class**

Table Per Concrete (1/2)

- Maps each of the concrete classes as normal persistent class
  - Pros
    - Easiest to implement
  - Cons
    - The main problem with this approach is that it doesn't support polymorphic associations very well
    - Data belonging to a parent class is scattered across a number of different tables
    - A query couched in terms of parent class is likely to cause a large number of select operations
    - Changes to a parent class can touch large number of tables
    - This scheme is not recommended for most cases
- How to define the mapping?
  - The mapping of the subclass repeats the properties of the parent class

Table Per Concrete (2/2)

- Maps each of the concrete classes as normal persistent class:

```
<class name="CreditCardPayment" table="CREDIT_PAYMENT">
   <id name="id" type=long" column="CREDIT_PAYMENT_ID">
    <generator class="native"/>
   </id>
<property name="amount" column="CREDIT_AMOUNT" />
...
</class>
<class name="CashPayment" table="CASH_PAYMENT">
   <id name="id" type=long" column="CASH_PAYMENT_ID">
    <generator class="native"/>
   </id>
   <property name="amount" column="CASH_AMOUNT" />
...
</class>
<class name="ChequePayment" table="CHEQUE_PAYMENT">
<id name="id" type=long" column="CREDIT_PAYMENT_ID">
    <generator class="native"/>
   </id>
   <property name="amount" column="CHEQUE_AMOUNT" />
...
</class>
```

Thank You

CitiusTech Markets

CitiusTech Services

CitiusTech Platforms

Accelerating Innovation

CitiusTech Contacts

Email ct-univerct@citiustech.com

www.citiustech.com