**Number of total purchases per package**

create table `total\_purchases\_per\_package`(

`packageId` int NOT NULL AUTO\_INCREMENT,

`total\_purchases` int NOT NULL DEFAULT 0,

PRIMARY KEY (`packageId`),

CONSTRAINT `total\_purchases\_per\_package\_servicepackage` FOREIGN KEY (`packageId`) REFERENCES `servicepackage` (`packageId`)

)

DELIMITER $$

create trigger insert\_new\_total\_purchases\_per\_package after insert ON servicepackage

for each row

begin

insert into total\_purchases\_per\_package(packageId, total\_purchases) values (new.packageId, 0);

end$$

DELIMITER ;

DELIMITER $$

create trigger update\_total\_purchases\_per\_package after update ON orders

for each row

begin

if not(old.orderState <=> new.orderstate) and new.orderstate <=> "Paid" then

update total\_purchases\_per\_package

set total\_purchases = total\_purchases + 1

where packageId in (select packageId from period where new.periodId = ID);

end if;

end$$

DELIMITER ;

DELIMITER $$

create trigger insert\_paid\_total\_purchases\_per\_package after insert ON orders

for each row

begin

if new.orderstate <=> "Paid" then

update total\_purchases\_per\_package

set total\_purchases = total\_purchases + 1

where packageId in (select packageId from period where new.periodId = ID);

end if;

end$$

DELIMITER ;

**Number of total purchases per package and validity period**

create table `total\_purchases\_per\_package\_validityperiod`(

`periodID` int NOT NULL AUTO\_INCREMENT,

`total\_purchases` int NOT NULL DEFAULT 0,

PRIMARY KEY (`periodID`),

CONSTRAINT `total\_purchases\_per\_package\_validityperiod\_period` FOREIGN KEY (`periodID`) REFERENCES `period` (`ID`)

)

DELIMITER $$

create trigger insert\_new\_total\_purchases\_per\_package\_validityperiod after insert ON period

for each row

begin

insert into total\_purchases\_per\_package\_validityperiod(periodID, total\_purchases) values (new.ID,0);

end$$

DELIMITER ;

DELIMITER $$

create trigger update\_total\_purchases\_per\_package\_validityperiod after update ON orders

for each row

begin

if not(old.orderState <=> new.orderstate) and new.orderstate <=> "Paid" then

update total\_purchases\_per\_package\_validityperiod

set total\_purchases = total\_purchases + 1

where periodID = new.periodId;

end if;

end$$

DELIMITER ;;

DELIMITER $$

create trigger insert\_paid\_total\_purchases\_per\_package\_validityperiod after insert ON orders

for each row

begin

if new.orderstate <=> "Paid" then

update total\_purchases\_per\_package\_validityperiod

set total\_purchases = total\_purchases + 1

where periodID = new.periodId;

end if;

end$$

DELIMITER ;

**Total sales per package with and without optional product**

create table `total\_sales\_per\_package`(

`packageId` int NOT NULL AUTO\_INCREMENT,

`totalSales` int NOT NULL DEFAULT 0,

`totalSalesWithOptionalProduct` int NOT NULL DEFAULT 0,

PRIMARY KEY (`packageId`),

CONSTRAINT `total\_sales\_per\_package\_servicepackage` FOREIGN KEY (`packageId`) REFERENCES `servicepackage` (`packageId`)

)

DELIMITER $$

create trigger insert\_new\_total\_sales\_per\_package after insert ON servicepackage

for each row

begin

insert into total\_sales\_per\_package(packageId, totalSales,totalSalesWithOptionalProduct) values (new.packageId, 0,0);

end$$

DELIMITER ;

DELIMITER $$

create trigger update\_sales\_purchases\_per\_package after update ON orders

for each row

begin

declare packageId2 int;

declare monthlyFee2 float;

declare validityPeriod2 int;

select

packageId,

monthlyFee,

validityPeriod

into

packageId2, monthlyFee2, validityPeriod2

from period where new.periodId = ID;

if not(old.orderState <=> new.orderstate) and new.orderstate <=> "Paid" then

update total\_sales\_per\_package

set totalSales = totalSales + monthlyFee2\*validityPeriod2,

totalSalesWithOptionalProduct = totalSalesWithOptionalProduct + new.totalFee

where packageId = packageId2;

end if;

end$$

DELIMITER ;

DELIMITER $$

create trigger insert\_sales\_purchases\_per\_package after insert ON orders

for each row

begin

declare packageId2 int;

declare monthlyFee2 float;

declare validityPeriod2 int;

select

packageId,

monthlyFee,

validityPeriod

into

packageId2, monthlyFee2, validityPeriod2

from period where new.periodId = ID;

if new.orderstate <=> "Paid" then

update total\_sales\_per\_package

set totalSales = totalSales + monthlyFee2\*validityPeriod2,

totalSalesWithOptionalProduct = totalSalesWithOptionalProduct + new.totalFee

where packageId = packageId2;

end if;

end$$

DELIMITER ;

**Average number of sales per optional product with each service package**

create table `average\_sales\_optionalproduct\_per\_servicepackage`(

`packageId` int NOT NULL AUTO\_INCREMENT,

`averageOptionalProducts` float NOT NULL DEFAULT 0,

PRIMARY KEY (`packageId`),

CONSTRAINT `average\_sales\_optionalproduct\_per\_servicepackage\_servicepackage` FOREIGN KEY (`packageId`) REFERENCES `servicepackage` (`packageId`)

)

DELIMITER $$

create trigger insert\_new\_average\_sales\_optionalproduct\_per\_servicepackage after insert ON servicepackage

for each row

begin

insert into average\_sales\_optionalproduct\_per\_servicepackage(packageId, averageOptionalProducts) values (new.packageId, 0);

end$$

DELIMITER ;

DELIMITER $$

create trigger update\_average\_sales\_optionalproduct\_per\_servicepackage after update ON orders

for each row

begin

declare packageId2 int;

declare totalPackage int;

declare newProducts int;

select packageId into packageId2 from period where new.periodId = ID;

select count(\*) into totalPackage from orders where periodId = new.periodId;

select count(\*) into newProducts from order\_optionalproduct where order\_Id = new.orderId;

if not(old.orderState <=> new.orderstate) and new.orderstate <=> "Paid" then

update average\_sales\_optionalproduct\_per\_servicepackage

set averageOptionalProducts = (averageOptionalProducts \* (totalPackage -1) + newProducts)/totalpackage

where packageId = packageId2;

end if;

end$$

DELIMITER ;

DELIMITER $$

create trigger insert\_average\_sales\_optionalproduct\_per\_servicepackage after insert ON order\_optionalproduct

for each row

begin

declare packageId2 int;

declare totalPackage int;

declare newProducts int;

declare newPeriodID int;

declare newOrderState char(20);

select periodId,orderState into newPeriodID, newOrderState from orders where new.order\_id = orderId;

select packageId into packageId2 from period where newPeriodID = ID;

select count(\*) into totalPackage from orders where periodId = newPeriodID;

select count(\*) into newProducts from order\_optionalproduct where order\_id = new.order\_id;

if newOrderState <=> "Paid" then

update average\_sales\_optionalproduct\_per\_servicepackage

set averageOptionalProducts = (averageOptionalProducts \* (totalPackage -1) + newProducts)/totalpackage

where packageId = packageId2;

end if;

end$$

DELIMITER ;

**List of insolvent users**

create table `insolvent\_users`(

`username` varchar(64) NOT NULL,

PRIMARY KEY (`username`),

CONSTRAINT `insolvent\_users\_users` FOREIGN KEY (`username`) REFERENCES `users` (`username`)

)

DELIMITER $$

create trigger insert\_insolvent\_users after update ON users

for each row

begin

if not(new.isInsolvent <=> old.isInsolvent) and new.isInsolvent <=> 1 then

insert into insolvent\_users(username) values (new.username);

end if;

end$$

DELIMITER ;

DELIMITER $$

create trigger delete\_insolvent\_users after update ON users

for each row

begin

if not(new.isInsolvent <=> old.isInsolvent) and new.isInsolvent <=> 0 then

delete from insolvent\_users where username = new.username;

end if;

end$$

DELIMITER ;

**List of suspended orders**

create table `suspended\_orders`(

`orderId` int NOT NULL AUTO\_INCREMENT,

PRIMARY KEY (`orderId`),

CONSTRAINT `suspended\_orders\_orders` FOREIGN KEY (`orderId`) REFERENCES `orders` (`orderId`)

)

DELIMITER $$

create trigger insert\_suspended\_orders after insert ON orders

for each row

begin

if new.orderState <=> "Rejected" then

insert into suspended\_orders(orderId) values (new.orderId);

end if;

end$$

DELIMITER ;

DELIMITER $$

create trigger delete\_suspended\_orders after update ON orders

for each row

begin

if new.orderState <=> "Paid" and old.orderState <=>"Rejected" then

delete from suspended\_orders where orderId = new.orderId;

end if;

end$$

DELIMITER ;

**List of Alerts**

CREATE TABLE `alert` (

`alertId` int NOT NULL AUTO\_INCREMENT,

`username` varchar(64) NOT NULL,

`amount` float NOT NULL,

`lastRejectionDateTime` timestamp NOT NULL,

PRIMARY KEY (`alertId`),

CONSTRAINT `username` FOREIGN KEY (`username`) REFERENCES `users` (`username`)

)

DELIMITER $$

create trigger update\_failed\_attempt after update ON orders

for each row

begin

declare attempts int;

declare email2 varchar(64);

select FailedAttempts,email into attempts,email2 from users where username;

if new.orderState <=> "Rejected" and attempts <=> 2 then

insert into alert(username,amount,lastRejectionDateTime,email) values (new.username,new.totalFee,current\_timestamp(),email2);

end if;

if new.orderState <=>"Rejected" then

update users

set FailedAttempts = attempts + 1, isInsolvent = 1 where username = new.username;

end if;

end$$

DELIMITER;

DELIMITER $$

create trigger delete\_insolvence after delete ON suspended\_orders

for each row

begin

declare username2 varchar(64);

declare unpaid int;

select username into username2 from orders where orderId = old.orderId limit 1;

select count(\*) into unpaid from orders where username = username2 and orderState = "Rejected";

if unpaid <=> 0 then

update users

set isInsolvent = 0 , FailedAttempts = 0 where username = username2;

end if;

end$$

DELIMITER ;

DELIMITER $$

create trigger insert\_failed\_attempt after insert ON orders

for each row

begin

declare attempts int;

declare email2 varchar(64);

select FailedAttempts,email into attempts,email2 from users where username = new.username limit 1;

if new.orderState <=> "Rejected" and attempts <=> 2 then

insert into alert(username,amount,lastRejectionDateTime,email) values (new.username,new.totalFee,current\_timestamp(),email2);

end if;

if new.orderState <=>"Rejected" then

update users

set FailedAttempts = attempts + 1, isInsolvent = 1 where username = new.username;

end if;

end$$

DELIMITER ;

**Best seller optional product**

Al momento considero tra i best seller anche gli optional product non pagati

create table `bestseller\_optionalproduct`(

`ID` int NOT NULL AUTO\_INCREMENT,

`productID` int,

`sales` int,

PRIMARY KEY (`ID`),

CONSTRAINT `bestseller\_optionalproduct\_optionalproduct` FOREIGN KEY (`productID`) REFERENCES `optionalproduct` (`productID`)

)

DELIMITER $$

create trigger update\_bestseller\_optionalproduct after update ON order\_optionalproduct

for each row

begin

declare optionalproduct\_Id2 int;

declare total int;

SELECT optionalproduct\_Id, count(\*)

into optionalproduct\_Id2, total

FROM order\_optionalproduct

GROUP BY optionalproduct\_Id

ORDER BY count(\*) DESC

LIMIT 1;

update bestseller\_optionalproduct

set productID = optionalproduct\_Id2, sales = total where ID = 1;

end$$

DELIMITER ;

DELIMITER $$

create trigger insert\_bestseller\_optionalproduct after insert ON order\_optionalproduct

for each row

begin

declare optionalproduct\_Id2 int;

declare total int;

SELECT optionalproduct\_Id, count(\*)

into optionalproduct\_Id2, total

FROM order\_optionalproduct

GROUP BY optionalproduct\_Id

ORDER BY count(\*) DESC

LIMIT 1;

update bestseller\_optionalproduct

set productID = optionalproduct\_Id2, sales = total where ID = 1;

end$$

DELIMITER ;

**Service activation scheduler service**

CREATE TABLE `activation\_scheduler\_service` (

`activationId` int NOT NULL AUTO\_INCREMENT,

`serviceId` int NOT NULL,

`startTime` timestamp NOT NULL,

`endTime` timestamp NOT NULL,

`username` varchar(64) NOT NULL,

PRIMARY KEY (`activationId`),

KEY `activation\_scheduler\_service\_serviceId` (`serviceId`),

KEY `activation\_scheduler\_service\_username` (`username`),

CONSTRAINT `activation\_scheduler\_service\_serviceId` FOREIGN KEY (`serviceId`) REFERENCES `service` (`serviceId`),

CONSTRAINT `activation\_scheduler\_service\_username` FOREIGN KEY (`username`) REFERENCES `users` (`username`)

)

DELIMITER $$

create trigger insert\_scheduler\_service after insert ON orders

for each row

begin

insert into activation\_scheduler\_service(serviceId,startTime,endTime,username) select

servicepackage\_service.service\_id, orders.startTime, orders.endTime, orders.username

from servicepackage\_service join period on servicepackage\_service.servicepackage\_id = period.packageId

join orders on orders.periodId = period.ID where orders.orderState = "Paid" and orders.orderId = new.orderId;

end$$

DELIMITER ;

DELIMITER $$

create trigger update\_scheduler\_service after update ON orders

for each row

begin

insert into activation\_scheduler\_service(serviceId,startTime,endTime,username) select

servicepackage\_service.service\_id, orders.startTime, orders.endTime, orders.username

from servicepackage\_service join period on servicepackage\_service.servicepackage\_id = period.packageId

join orders on orders.periodId = period.ID where orders.orderState = "Paid" and orders.orderId = new.orderId

and old.orderState = "Rejected";

end$$

DELIMITER ;

**Service activation scheduler optional product**

CREATE TABLE `activation\_scheduler\_optionalproduct` (

`activationId` int NOT NULL,

`productId` int NOT NULL,

`startTime` timestamp NOT NULL,

`endTime` timestamp NOT NULL,

`username` varchar(64) NOT NULL,

PRIMARY KEY (`activationId`),

CONSTRAINT `activation\_scheduler\_optionalproduct\_productId` FOREIGN KEY (`productId`) REFERENCES `optionalproduct` (`productID`),

CONSTRAINT `activation\_scheduler\_optionalproduct\_username` FOREIGN KEY (`username`) REFERENCES `users` (`username`)

)

DELIMITER $$

create trigger insert\_scheduler\_optionalproduct after insert ON orders

for each row

begin

insert into activation\_scheduler\_optionalproduct(productId,startTime,endTime,username) select

servicepackage\_optionalproduct.optionalproduct\_productID, orders.startTime, orders.endTime, orders.username

from servicepackage\_optionalproduct join period on servicepackage\_id = period.packageId

join orders on orders.periodId = period.ID where orders.orderState = "Paid" and orders.orderId = new.orderId;

end$$

DELIMITER ;

DELIMITER $$

create trigger update\_scheduler\_optionalproduct after update ON orders

for each row

begin

insert into activation\_scheduler\_optionalproduct(productId,startTime,endTime,username) select

servicepackage\_optionalproduct.optionalproduct\_productID, orders.startTime, orders.endTime, orders.username

from servicepackage\_optionalproduct join period on servicepackage\_id = period.packageId

join orders on orders.periodId = period.ID where orders.orderState = "Paid" and orders.orderId = new.orderId

and old.orderState = "Rejected";

end$$

DELIMITER ;