Data Bases 2

Telcom Service Application
2021-2022
Matteo Savino
Giacomo Vinati

Index

Specifications (Consumer Application)

- ***Telco service Applications***

A telco company offers prepaid online services to web users. Two client applications using the same database need to be developed.

- ***Consumer Application***

The consumer application has a public Landing page with a form for login and a form for registration. Registration requires a username (which can be assumed as the unique identification parameter), a password and an email. Login leads to the Home page of the consumer application. Registration leads back to the landing page where the user can login.The user can log in before browsing the application or browse it without logging in. If the user has logged in, his/her username appears in the top right corner of all the application pages.The Home page of the consumer application displays the service packages offered by the telco company.

Specifications (Consumer Application 2)

- ***Consumer Application 2***

A service package has an ID and a name (e.g., “Basic”, “Family”, “Business”, “All Inclusive”, etc). It comprises one or more services. Services are of four types: fixed phone, mobile phone, fixed internet, and mobile internet. The mobile phone service specifies the number of minutes and SMSs included in the package plus the fee for extra minutes and the fee for extra SMSs. The fixed phone service has no specific configuration parameters. The mobile and fixed internet services specify the number of Gigabytes included in the package and the fee for extra Gigabytes. A service package must be associated with one validity period. A validity period specifies the number of months (12, 24, or 36). Each validity period has a different monthly fee (e.g., 20€/month for 12 months, 18€/month for 24 months, and 15€ /month for 36 months). A package may be associated with one or more optional products (e.g., an SMS news feed, an internet TV channel, etc.). The validity period of an optional product is the same as the validity period that the user has chosen for the service package. An optional product has a name and a monthly fee independent of the validity period duration. The same optional product can be offered in different service packages.

From the Home page, the user can access a Buy Service page for purchasing a service package and thus creating a service subscription. The Buy Service page contains a form for purchasing a service package. The form allows the user to select one package from the list of available ones and choose the validity period duration and the optional products to buy together with the chosen service. The form also allows the user to select the start date of his/her subscription. After choosing the service packages, the validity period and (0 or more) optional products, the user can press a CONFIRM button. The application displays a CONFIRMATION page that summarizes the details of the chosen service package, the validity period, the optional products and the total price to be pre-paid: (monthly fee of service package * number of months) + (sum of monthly fees of options * number of months).

Specifications (Consumer Application 3)

- ***Consumer Application 3***

If the user has already logged in, the CONFIRMATION page displays a BUY button. If the user has not logged in, the CONFIRMATION page displays a link to the login page and a link to the REGISTRATION page. After either logging in or registering and immediately logging in, the CONFIRMATION page is redisplayed with all the confirmed details and the BUY button. When the user presses the BUY button, an order is created. The order has an ID and a date and hour of creation. It is associated with the user and with the service package, its validity period and the chosen optional products. It also contains the total value (as in the CONFIRMATION page) and the start date of the subscription. After creating the order, the application bills the customer by calling an external service. If the external service accepts the billing, the order is marked as valid and a service activation schedule is created for the user. A service activation schedule is a record of the services and optional products to activate for the user with their date of activation and date of deactivation. If the external service rejects the billing, the order is put in the rejected status and the user is flagged as insolvent. When an insolvent user logs in, the home page also contains the list of rejected orders. The user can select one of such orders, access the CONFIRMATION page, press the BUY button and attempt the payment again. When the same user causes three failed payments, an alert is created in a dedicated auditing table, with the user Id, username, email, and the amount, date and time of the last rejection.

Specifications (Employee Application)

- ***Employee Application***

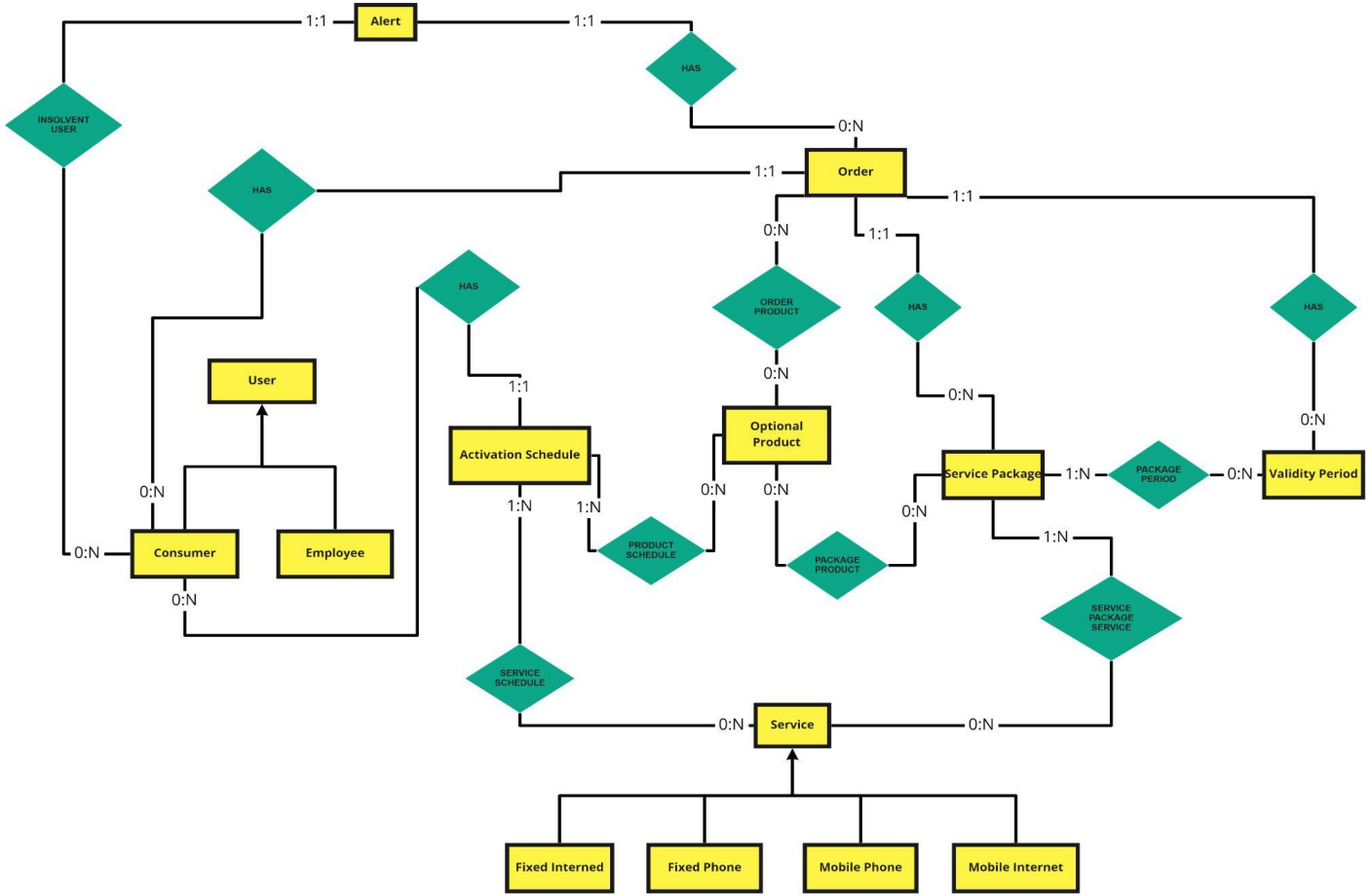
The employee application allows the authorized employees of the telco company to log in. In the Home page, a form allows the creation of service packages, with all the needed data and the possible optional products associated with them. The same page lets the employee create optional products as well. A Sales Report page allows the employee to inspect the essential data about the sales and about the users over the entire lifespan of the application:

  - Number of total purchases per package.Number of total purchases per package and validity period.
  - Total value of sales per package with and without the optional products.
  - Average number of optional products sold together with each service package.
  - List of insolvent users, suspended orders and alerts.
  - Best seller optional product, i.e. the optional product with the greatest value of sales across all the sold service packages.

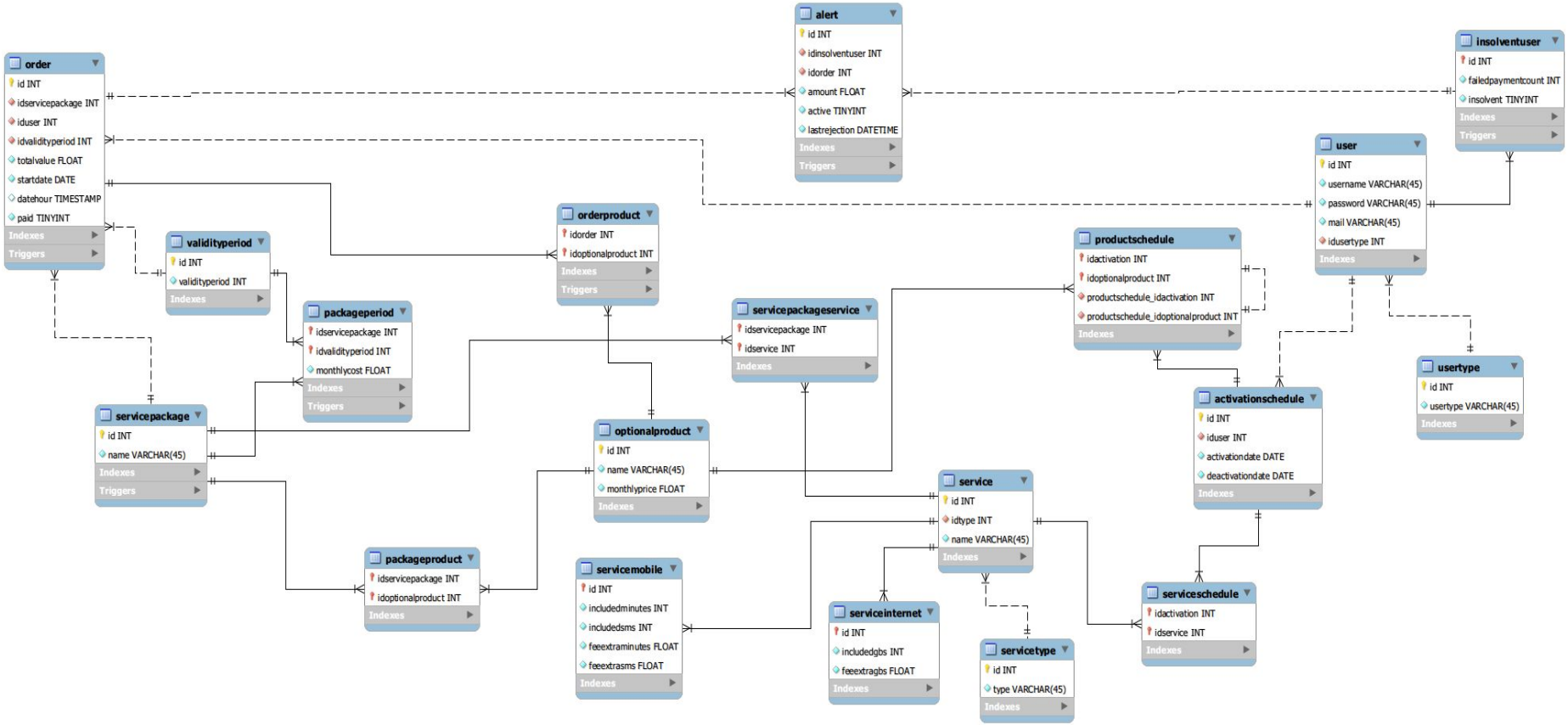
Revision of the specifications

- The aggregate data of the sales report is computed by triggers that populate materialized view tables.
- The payment will be simulated, for demonstration purposes, by allowing the user to choose if the payment will be successful or rejected.
- Triggers regarding service packages, for the sales report page, consider paid orders.
- Once a user has failed two payments in a row, every following failed payment will generate an alert, conceivably more alerts for the same order will be created.
- Any alert will be deactivated as soon as the user pays the related order.
- The system is optimistic so whenever an insolvent user complete a payment he is no longer considered insolvent even if he still has some suspended orders. The system expects the user to pay them as soon as possible.
- Activation schedules and alerts are not shown in the home pages of either the consumer or the employee.
- Telco service mobile and internet additional data are not shown because in the application they would be used just for displaying purposes.

Entity Relationship model 1



Entity Relationship model 2 (from MySqlWorkbench)



Explanation of ER diagram

- The materialized view tables for the sales report page are not in the ER diagram. They would have decreased readability without enhancing the completeness.

Relational model (Logical Data Model) 1

- ActivationSchedule(id, *iduser*, activationdate, deactivationdate)
- Alert(id, *idinsolventuser*, *idorder*, amount, active, lastrejection)
- InsolventUser(*id*, failedpaymentcount, insolvent)
- OptionalProduct(id, name, monthlyprice)
- Order(id, *idservicepackage*, *iduser*, *idvalidityperiod*, totalvalue, startdate, datehour, paid)
- OrderProduct(*idorder*, *idoptionalproduct*)
- PackagePeriod(*idservicepackage*, *idvalidityperiod*, monthlycost)
- PackageProduct(*idservicepackage*, *idoptionalproduct*)
- ProductSchedule(*idactivationschedule*, *idoptionalproduct*)
- Service(id, *idtype*, name)

Relational model 2

- ServiceInternet(*id*, includedgbs, feeextragbs)
- ServiceMobile(*id*, includedminutes, includedsms, feeextraminutes, feeextrasms)
- ServicePackage(id, name)
- ServicePackageService(*idservicepackage*, *idservice*)
- ServiceSchedule(*idactivationschedule*, *idservice*)
- ServiceType(id, type)
- User(id, username, password, mail, *idusertype)*
- Usertype(id, usertype)
- Validityperiod(id, validityperiod)

Relational model 3 (materialized view)

- mv_alerts(id, *idalert*)
- mv_bestproduct(id, *idoptionalproduct*, value, sales)
- mv_insolventuser(id, *idinsolventuser*)
- mv_package(id, *idservicepackage*, sales, value, valuewithproducts, avgoptionalproducts)
- mv_packageperiod(id, *idservicepackage*, *idperiod*, sales)
- mv_suspendedorders(id, *idorder*)

Explanation of relational model

- In order to increase readability we did not include edges (arrows) in the model. We just underlined the columns that are foreign keys. The table they reference to is easily inferable by column names.
- We decided to aggregate the User table and to create additional tables for the columns that are related to the possible specializations.
- We decided to aggregate the Service table and to create additional tables for the columns that are related to the possible specializations.

SQL DDL 1

```
CREATE TABLE `activationschedule` (

  `id` int NOT NULL AUTO_INCREMENT,

  `iduser` int NOT NULL,

  `activationdate` date NOT NULL,

  `deactivationdate` date NOT NULL,

  PRIMARY KEY (`id`),

  UNIQUE KEY `id_UNIQUE` (`id`),

  KEY `FK_UserActivationSchedule_idx`
(`iduser`),

  CONSTRAINT `FK_UserActivationSchedule` FOREIGN
KEY (`iduser`) REFERENCES `user` (`id`) ON
UPDATE CASCADE

)
```

```
CREATE TABLE `alert` (

  `id` int NOT NULL AUTO_INCREMENT,

  `idinsolventuser` int NOT NULL,

  `idorder` int NOT NULL,

  `amount` float NOT NULL,

  `active` tinyint NOT NULL DEFAULT '1',

  `lastrejection` datetime NOT NULL DEFAULT
CURRENT_TIMESTAMP,

  PRIMARY KEY (`id`),

  UNIQUE KEY `id_UNIQUE` (`id`),

  KEY `FK_AlertUser_idx` (`idinsolventuser`),

  KEY `FK_AlertOrder_idx` (`idorder`),

  CONSTRAINT `FK_AlertOrder` FOREIGN KEY (`idorder`)
REFERENCES `order` (`id`) ON DELETE CASCADE ON UPDATE
CASCADE,

  CONSTRAINT `FK_AlertUser` FOREIGN KEY
(`idinsolventuser`) REFERENCES `insolventuser` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE

)
```

SQL DDL 2

```
CREATE TABLE `insolventuser` (

  `id` int NOT NULL,

  `failedpaymentcount` int NOT NULL DEFAULT '0',

  `insolvent` tinyint NOT NULL DEFAULT '0',

  PRIMARY KEY (`id`),

  UNIQUE KEY `id_UNIQUE` (`id`),

  CONSTRAINT `FK_InsolventUser` FOREIGN KEY
(`id`) REFERENCES `user` (`id`) ON DELETE
CASCADE ON UPDATE CASCADE

)
```

```
CREATE TABLE `mv_alerts` (

  `id` int NOT NULL AUTO_INCREMENT,

  `idalert` int NOT NULL,

  PRIMARY KEY (`id`),

  UNIQUE KEY `id_UNIQUE` (`id`),

  UNIQUE KEY `idalert_UNIQUE` (`idalert`),

  CONSTRAINT `FK_idalert` FOREIGN KEY
(`idalert`) REFERENCES `alert` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE

)
```

SQL DDL 3

```
CREATE TABLE `mv_bestproduct` (
  `id` int NOT NULL AUTO_INCREMENT,
  `idoptionalproduct` int NOT NULL,
  `value` float NOT NULL,
  `sales` int NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`),
  KEY `FK_MvBestProduct_idx` (`idoptionalproduct`),
  CONSTRAINT `FK_MvBestProduct` FOREIGN KEY
(`idoptionalproduct`) REFERENCES `optionalproduct`
(`id`) ON DELETE CASCADE ON UPDATE CASCADE
)
```

```
CREATE TABLE `mv_insolventuser` (
  `id` int NOT NULL AUTO_INCREMENT,
  `idinsolventuser` int NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`),
  UNIQUE KEY `idinsolventuser_UNIQUE`
(`idinsolventuser`),
  CONSTRAINT `FK_MVInsolventUser` FOREIGN KEY
(`idinsolventuser`) REFERENCES `user` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE
)
```

SQL DDL 4

```
CREATE TABLE `mv_package` (
  `id` int NOT NULL AUTO_INCREMENT,
  `idpackage` int NOT NULL,
  `sales` int DEFAULT NULL,
  `value` float DEFAULT NULL,
  `valuewithproducts` float DEFAULT NULL,
  `avgoptionalproducts` float DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`),
  UNIQUE KEY `idpackage_UNIQUE` (`idpackage`),
  KEY `FL_MVPackage_idx` (`idpackage`),
  CONSTRAINT `FL_MVPackage` FOREIGN KEY
(`idpackage`) REFERENCES `servicepackage` (`id`)
ON DELETE CASCADE ON UPDATE CASCADE
)
```

```
CREATE TABLE `mv_packageperiod` (
  `id` int NOT NULL AUTO_INCREMENT,
  `idpackage` int NOT NULL,
  `idperiod` int NOT NULL,
  `sales` int NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`),
  KEY `FK_MVPackagePeriod_idx` (`idpackage`),
  KEY `FK_MVPackagePeriodPeriod_idx` (`idperiod`),
  KEY `FK_MVPackagePeriodVP_idx` (`idperiod`),
  CONSTRAINT `FK_MVPackagePeriod` FOREIGN KEY
(`idpackage`) REFERENCES `servicepackage` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `FK_MVPackagePeriodVP` FOREIGN KEY
(`idperiod`) REFERENCES `validityperiod` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE
)
```

SQL DDL 5

```
CREATE TABLE `mv_suspendedorders` (
  `id` int NOT NULL AUTO_INCREMENT,
  `idorder` int NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`),
  UNIQUE KEY `idorder_UNIQUE` (`idorder`),
  CONSTRAINT `FK_MVSuspendedOrders` FOREIGN KEY (`idorder`) REFERENCES `order` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
)
```

```
CREATE TABLE `optionalproduct` (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(45) NOT NULL,
  `monthlyprice` float NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`)
)
```

SQL DDL 6

```
CREATE TABLE  `order` (

  `id` int NOT NULL AUTO_INCREMENT,

  `idservicepackage` int NOT NULL,

  `iduser` int NOT NULL,

  `idvalidityperiod` int NOT NULL,

  `totalvalue` float NOT NULL,

  `startdate` date NOT NULL,

  `datehour` timestamp NULL DEFAULT CURRENT_TIMESTAMP,

  `paid` tinyint NOT NULL DEFAULT '0',

  PRIMARY KEY (`id`),

  UNIQUE KEY `id_UNIQUE` (`id`),

  KEY `OrderUser_idx` (`iduser`),

  KEY `OrderPackage_idx` (`idservicepackage`),

  KEY `FK_OrderValidityPeriodOfPackage_idx` (`idvalidityperiod`),

  CONSTRAINT `FK_OrderPackage` FOREIGN KEY (`idservicepackage`) REFERENCES `servicepackage` (`id`) ON UPDATE
CASCADE,

  CONSTRAINT `FK_OrderUser` FOREIGN KEY (`iduser`) REFERENCES `user` (`id`) ON UPDATE CASCADE,

  CONSTRAINT `FK_OrderValidityPeriodOfPackage` FOREIGN KEY (`idvalidityperiod`) REFERENCES `validityperiod` (`id`)
ON UPDATE CASCADE

)
```

SQL DDL 7

```
CREATE TABLE `orderproduct` (

  `idorder` int NOT NULL,

  `idoptionalproduct` int NOT NULL,

  PRIMARY KEY (`idorder`,`idoptionalproduct`),

  KEY `FK_OptionalProductOrder_idx`
(`idoptionalproduct`),

  CONSTRAINT `FK_OptionalProductOrder` FOREIGN
KEY (`idoptionalproduct`) REFERENCES
`optionalproduct` (`id`) ON DELETE CASCADE ON
UPDATE CASCADE,

  CONSTRAINT `FK_OrderOptionalProduct` FOREIGN
KEY (`idorder`) REFERENCES `order` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE

)
```

```
CREATE TABLE `packageperiod` (

  `idservicepackage` int NOT NULL,

  `idvalidityperiod` int NOT NULL,

  `monthlycost` float NOT NULL,

  PRIMARY KEY
(`idservicepackage`,`idvalidityperiod`),

  KEY `ValidityPeriod_idx` (`idvalidityperiod`),

  KEY `ServicePackage_idx` (`idservicepackage`),

  CONSTRAINT `FK_ServicePackageValidityPeriod`
FOREIGN KEY (`idservicepackage`) REFERENCES
`servicepackage` (`id`) ON DELETE CASCADE ON
UPDATE CASCADE,

  CONSTRAINT `FK_ValidityPeriodServicePackage`
FOREIGN KEY (`idvalidityperiod`) REFERENCES
`validityperiod` (`id`) ON DELETE CASCADE ON
UPDATE CASCADE

)
```

SQL DDL 8

```
CREATE TABLE `packageproduct` (

  `idservicepackage` int NOT NULL,

  `idoptionalproduct` int NOT NULL,

  PRIMARY KEY
(`idoptionalproduct`,`idservicepackage`),

  KEY `ServicePackage_idx` (`idservicepackage`),

  CONSTRAINT `FK_OptionalProductServicePackage`
FOREIGN KEY (`idoptionalproduct`) REFERENCES
`optionalproduct` (`id`) ON DELETE CASCADE ON
UPDATE CASCADE,

  CONSTRAINT `FK_ServicePackageOptionalProduct`
FOREIGN KEY (`idservicepackage`) REFERENCES
`servicepackage` (`id`) ON DELETE CASCADE ON
UPDATE CASCADE

)
```

```
CREATE TABLE `productschedule` (

  `idactivation` int NOT NULL,

  `idoptionalproduct` int NOT NULL,

  PRIMARY KEY
(`idactivation`,`idoptionalproduct`),

  KEY `FK_OptionalProductActivation_idx`
(`idoptionalproduct`),

  KEY `FK_ProductActivation_idx`
(`idactivation`),

  CONSTRAINT `FK_ActivationOP` FOREIGN KEY
(`idactivation`) REFERENCES
`activationschedule` (`id`) ON DELETE CASCADE
ON UPDATE CASCADE,

  CONSTRAINT `FK_OptionalProductActivation`
FOREIGN KEY (`idoptionalproduct`) REFERENCES
`optionalproduct` (`id`) ON UPDATE CASCADE

)
```

SQL DDL 9

```
CREATE TABLE `service` (
  `id` int NOT NULL AUTO_INCREMENT,
  `idtype` int NOT NULL,
  `name` varchar(45) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`),
  UNIQUE KEY `name_UNIQUE` (`name`),
  KEY `idtype_idx` (`idtype`),
  CONSTRAINT `FK_Type` FOREIGN KEY (`idtype`) REFERENCES `servicetype` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
)
```

```
CREATE TABLE `serviceinternet` (
  `id` int NOT NULL,
  `includedgbs` int NOT NULL,
  `feeextragbs` float NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`),
  CONSTRAINT `FK_ServiceInternet` FOREIGN KEY (`id`) REFERENCES `service` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
)
```

SQL DDL 10

```
CREATE TABLE `servicemobile` (
  `id` int NOT NULL,
  `includedminutes` int NOT NULL,
  `includedsms` int NOT NULL,
  `feeextraminutes` float NOT NULL,
  `feeextrasms` float NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`),
  CONSTRAINT `FK_ServiceMobile` FOREIGN KEY (`id`) REFERENCES `service` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
)
```

```
CREATE TABLE `servicepackage` (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(45) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`),
  UNIQUE KEY `name_UNIQUE` (`name`)
)
```

SQL DDL 11

```
CREATE TABLE `servicepackageservice` (

  `idservicepackage` int NOT NULL,

  `idservice` int NOT NULL,

  PRIMARY KEY (`idservicepackage`,`idservice`),

  KEY `FK_ServicePackageService2_idx`
(`idservice`),

  CONSTRAINT `FK_ServicePackageService` FOREIGN KEY
(`idservicepackage`) REFERENCES `servicepackage`
(`id`) ON DELETE CASCADE ON UPDATE CASCADE,

  CONSTRAINT `FK_ServicePackageService2` FOREIGN
KEY (`idservice`) REFERENCES `service` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE

)
```

```
CREATE TABLE `serviceschedule` (

  `idactivation` int NOT NULL,

  `idservice` int NOT NULL,

  PRIMARY KEY (`idactivation`,`idservice`),

  KEY `FK_ServiceActivation_idx`
(`idservice`),

  KEY `FK_ActivationS_idx` (`idactivation`),

  CONSTRAINT `FK_ActivationS` FOREIGN KEY
(`idactivation`) REFERENCES
`activationschedule` (`id`) ON DELETE
CASCADE ON UPDATE CASCADE,

  CONSTRAINT `FK_ServiceActivation` FOREIGN
KEY (`idservice`) REFERENCES `service`
(`id`) ON UPDATE CASCADE

)
```

SQL DDL 12

```
CREATE TABLE `servicetype` (
  `id` int NOT NULL AUTO_INCREMENT,
  `type` varchar(45) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`)
)
```

```
CREATE TABLE `usertype` (
  `id` int NOT NULL AUTO_INCREMENT,
  `usertype` varchar(45) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`),
  UNIQUE KEY `usertype_UNIQUE` (`usertype`)
)
```

```
CREATE TABLE `user` (
  `id` int NOT NULL AUTO_INCREMENT,
  `username` varchar(45) NOT NULL,
  `password` varchar(45) NOT NULL,
  `mail` varchar(45) NOT NULL,
  `idusertype` int NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `username_UNIQUE` (`username`),
  UNIQUE KEY `id_UNIQUE` (`id`),
  UNIQUE KEY `mail_UNIQUE` (`mail`),
  KEY `FK_UserType_idx` (`idusertype`),
  CONSTRAINT `FK_UserType` FOREIGN KEY
(`idusertype`) REFERENCES `usertype` (`id`)
)
```

SQL DDL 13

```
CREATE TABLE `validityperiod` (
  `id` int NOT NULL AUTO_INCREMENT,
  `validityperiod` int NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`)
)
```

Triggers design and code

The triggers are used to fill materialized view tables to allow the employee to get this data:

- Insolvent Users, Suspended orders, Alerts
- Best seller optional product
- Number of sales per package and validity period (Package period)
- Number of sales, value of sales, value of sales with optional products, average number of optional product sold per package (Package)

Triggers 1.1 (Insolvent Users)

```
CREATE TRIGGER db2_savino_vinati.MV_UserBecomesInsolventUpdate
      AFTER UPDATE
      ON db2_savino_vinati.insolventuser
      FOR EACH ROW
BEGIN
    IF (OLD.insolvent = 0 AND NEW.insolvent= 1) THEN
        INSERT INTO mv_insolventuser(idinsolventuser)
        VALUES(new.id);
      ELSEIF(OLD.insolvent = 1 AND NEW.insolvent = 0) THEN
            DELETE FROM mv_insolventuser WHERE idinsolventuser = new.id;
    END IF;
END
```

Triggers 1.2 (Suspended orders)

```
CREATE
TRIGGER db2_savino_vinati.MV_SuspendedOrderInsert
      AFTER INSERT
      ON db2_savino_vinati.`order`
      FOR EACH ROW
BEGIN
    IF (NEW.paid = 0) THEN
        INSERT INTO mv_suspendedorders(idorder)
        VALUES(new.id);
    END IF;
END
```

```
CREATE
TRIGGER db2_savino_vinati.MV_SuspendedOrderUpdate
      AFTER UPDATE
      ON db2_savino_vinati.`order`
      FOR EACH ROW
BEGIN
    IF (OLD.paid = 0 AND NEW.paid = 1) THEN
        DELETE FROM mv_suspendedorders
        WHERE idorder = old.id;
    END IF;
END
```

Triggers 1.3 (Alerts)

```
CREATE
TRIGGER db2_savino_vinati.MV_AlertInsert
      AFTER INSERT
      ON db2_savino_vinati.alert
      FOR EACH ROW
BEGIN

      IF (NEW.active = 1) THEN

      INSERT INTO mv_alerts (idalert)
      VALUES (new.id);
    END IF;
END
```

```
CREATE
TRIGGER db2_savino_vinati.MV_AlertUpdate
      AFTER UPDATE
      ON db2_savino_vinati.alert
      FOR EACH ROW
BEGIN
      IF (OLD.active = 1 AND NEW.active = 0)
THEN
      DELETE FROM mv_alerts WHERE idalert =
new.id;
    END IF;
END
```

Triggers 2.1 (BestProduct)

```
CREATE TRIGGER db2_savino_vinati. MV_BestProduct
AFTER INSERT ON db2_savino_vinati.orderproduct
FOR EACH ROW
BEGIN
DECLARE countBP INTEGER;
DECLARE valueBP float;
DECLARE salesBP integer;
DECLARE idBP integer;
SELECT COUNT(*) into countBP
FROM mv_bestproduct;
IF countBP < 1 THEN
INSERT INTO mv_bestproduct (idoptionalproduct, value, sales) VALUES(new.idoptionalproduct, 0, 0);
END IF;
SELECT op.idoptionalproduct as idoptionalproduct, SUM(opp.monthlyprice * vp.validityperiod) as `value`, COUNT(op.idorder) as sales
INTO idBP, valueBP, salesBP FROM `order` o
LEFT JOIN validityperiod vp ON o.idvalidityperiod = vp.id
LEFT JOIN orderproduct op ON o.id = op.idorder
LEFT JOIN optionalproduct opp ON op.idoptionalproduct = opp.id
WHERE o.paid = 1
GROUP BY opp.id ORDER BY `value` DESC LIMIT 1;
UPDATE mv_bestproduct SET idoptionalproduct = idBP, `value` = valueBP, sales = salesBP;
END
```

Triggers 2.2 (BestProduct)

```
CREATE TRIGGER db2_savino_vinati. MV_BestProductUpdate

AFTER UPDATE ON db2_savino_vinati.`order`

FOR EACH ROW

BEGIN

DECLARE valueBP float;

DECLARE salesBP integer;

DECLARE idBP integer;

IF old.paid = 0 AND new.paid = 1 THEN

SELECT op.idoptionalproduct as idoptionalproduct, SUM(opp.monthlyprice * vp.validityperiod) as `value`, COUNT(op.idorder) as sales

INTO idBP, valueBP, salesBP FROM `order` o

LEFT JOIN validityperiod vp ON o.idvalidityperiod = vp.id

LEFT JOIN orderproduct op ON o.id = op.idorder

LEFT JOIN optionalproduct opp ON op.idoptionalproduct = opp.id

WHERE o.paid = 1 GROUP BY opp.id ORDER BY `value` DESC

LIMIT 1;

UPDATE mv_bestproduct SET idoptionalproduct = idBP, `value` = valueBP, sales = salesBP;

END IF;

END
```

Triggers 3.1 (Package Period)

```
CREATE TRIGGER db2_savino_vinati.MV_PackagePeriodOrderInsert

AFTER INSERT ON db2_savino_vinati.`order`

FOR EACH ROW

BEGIN

DECLARE countPOI integer;

DECLARE periodPP integer;

DECLARE idservicepackagePP integer;

DECLARE salesPP integer;

SET idservicepackagePP = new.idservicepackage;

SET periodPP = new.idvalidityperiod;

SELECT COUNT(*) into countPOI FROM mv_packageperiod mvp

WHERE mvp.idpackage = idservicepackagePP AND mvp.idperiod = periodPP;

IF countPOI < 1 THEN

INSERT INTO mv_packageperiod ( idpackage, idperiod, sales) VALUES ( idservicepackagePP, periodPP, 0 );

END IF;

SELECT COUNT(o.id) INTO salesPP  FROM `order` o

WHERE o.paid = 1 AND o.idservicepackage = idservicepackagePP AND o.idvalidityperiod = periodPP;

IF new.paid= 1 THEN

UPDATE mv_packageperiod mvp SET mvp.sales = salesPP WHERE  mvp.idpackage = idservicepackagePP AND mvp.idperiod = periodPP;

END IF;

END
```

Triggers 3.2 (Package Period)

```
CREATE TRIGGER db2_savino_vinati.MV_PackagePeriodOrderUpdate
AFTER UPDATE ON db2_savino_vinati.`order`
FOR EACH ROW
FOLLOWS MV_PackageUpdateOrder
BEGIN
DECLARE periodPP integer;
DECLARE idservicepackagePP integer;
DECLARE salesPP integer;
IF old.paid = 0 AND new.paid = 1 THEN
SET idservicepackagePP = new.idservicepackage;
SET periodPP = new.idvalidityperiod;
SELECT COUNT(o.id) INTO salesPP FROM `order` o
WHERE o.paid = 1 AND o.idservicepackage = idservicepackagePP AND o.idvalidityperiod = periodPP;
UPDATE mv_packageperiod mvp SET mvp.sales = salesPP WHERE  mvp.idpackage = idservicepackagePP AND mvp.idperiod= periodPP;
END IF;
END
```

Triggers 3.3 (Package Period)

```
CREATE TRIGGER db2_savino_vinati.MV_PackagePeriodPackageInsert
AFTER INSERT
ON db2_savino_vinati.packageperiod
FOR EACH ROW
BEGIN
DECLARE periodPP integer;
DECLARE idservicepackagePP integer;
SET idservicepackagePP = new.idservicepackage;
SET periodPP = new.idvalidityperiod;
INSERT INTO mv_packageperiod (idpackage, idperiod, sales) VALUES (idservicepackagePP,periodPP,0);
END
```

Triggers 3.4 (Package Period)

```
CREATE TRIGGER db2_savino_vinati.MV_PackageOrderProductInsert
AFTER INSERT ON db2_savino_vinati.orderproduct
FOR EACH ROW
BEGIN
DECLARE avgoptionalproductsPI float;
DECLARE paidPI tinyint;
DECLARE idpackagePI int;
SELECT o.idservicepackage, o.paid INTO idpackagePI, paidPI FROM `order` o WHERE o.id = new.idorder;
IF paidPI = 1  THEN
SELECT CAST(AVG(a.count)as FLOAT) as avgoptionalproducts INTO avgoptionalproductsPI
FROM servicepackage sp
LEFT JOIN `order` o ON sp.id = o.idservicepackage
LEFT JOIN ( SELECT op.idorder as idorder, COUNT(op.idorder) as count FROM orderproduct op
            GROUP BY idorder ) as a
ON a.idorder = o.id WHERE idpackagePI = o.idservicepackage AND o.paid = 1  GROUP BY sp.id;
UPDATE mv_package SET avgoptionalproducts = avgoptionalproductsPI
WHERE mv_package.idpackage = idpackagePI;
END IF;
END
```

Triggers 4.1 (Package)

```
CREATE
TRIGGER db2_savino_vinati.MV_Package
AFTER INSERT
ON db2_savino_vinati.servicepackage
FOR EACH ROW
BEGIN
INSERT INTO mv_package (idpackage, sales, `value`, valuewithproducts, avgoptionalproducts)
VALUES (new.id,null,null,null,null);
END
```

Triggers 4.2 (Package)

```
CREATE TRIGGER db2_savino_vinati.MV_PackageInsertOrder

AFTER INSERT ON db2_savino_vinati.`order`

FOR EACH ROW

BEGIN

DECLARE countPI INTEGER; DECLARE idpackagePI INTEGER; DECLARE salesPI integer; DECLARE valuePI float; DECLARE valuewithproductsPI float; DECLARE avgoptionalproductsPI float;

SELECT COUNT(*) into countPI FROM mv_package WHERE new.idservicepackage = mv_package.idpackage;

IF countPI < 1 THEN SET idpackagePI = new.idservicepackage;

INSERT INTO mv_package (idpackage, sales, `value`, valuewithproducts, avgoptionalproducts) VALUES(idpackagePI,0, 0,0,0);

END IF;

SELECT  sp.id as idpackage, COUNT(o.id) as sales, SUM(vp.validityperiod * pp.monthlycost) as `value`, SUM(o.totalvalue) as

valuewithproducts, CAST(AVG(a.count)as FLOAT) as avgoptionalproducts INTO idpackagePI,salesPI,valuePI,valuewithproductsPI, avgoptionalproductsPI

FROM servicepackage sp

LEFT JOIN `order` o ON sp.id = o.idservicepackage

LEFT JOIN validityperiod vp ON o.idvalidityperiod = vp.id

LEFT JOIN packageperiod pp ON vp.id = pp.idvalidityperiod AND o.idservicepackage = pp.idservicepackage

LEFT JOIN ( SELECT op.idorder as idorder, COUNT(op.idorder) as

count FROM orderproduct op GROUP BY idorder ) as a

ON a.idorder = o.id WHERE o.idservicepackage = new.idservicepackage AND o.paid

= 1 GROUP BY sp.id;

IF new.paid=1 THEN  UPDATE mv_package SET idpackage = idpackagePI, sales = salesPI, `value` = valuePI, valuewithproducts =

valuewithproductsPI, avgoptionalproducts = avgoptionalproductsPI WHERE mv_package.idpackage = new.idservicepackage;

END IF;

END
```

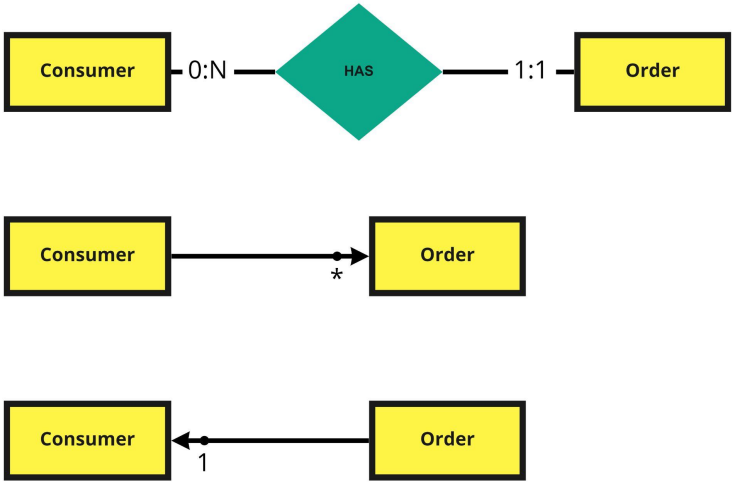
Triggers 4.3 (Package)

```
CREATE TRIGGER db2_savino_vinati.MV_PackageNoNull
BEFORE INSERT
ON db2_savino_vinati.mv_package
FOR EACH ROW
BEGIN
IF(new.sales is null) THEN
SET new.sales = 0;
 END IF;
IF(new.`value` is null) THEN
SET new.`value` = 0;
END IF;
IF (new.valuewithproducts is null) THEN
SET new.valuewithproducts = 0;
END IF;
IF(new.avgoptionalproducts is null) THEN
SET new.avgoptionalproducts = 0;
END IF;
END
```

Triggers 4.4(Package)

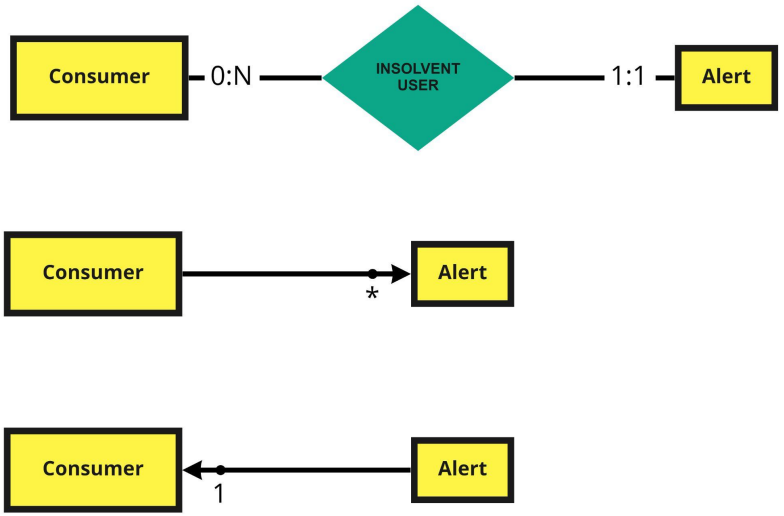
```
CREATE TRIGGER db2_savino_vinati.MV_PackageUpdateOrder

AFTER UPDATE ON db2_savino_vinati.`order`

FOR EACH ROW

BEGIN

DECLARE idpackagePI integer; DECLARE salesPI integer; DECLARE valuePI float; DECLARE valuewithproductsPI float; DECLARE avgoptionalproductsPI float;

IF old.paid = 0 AND new.paid = 1 THEN

SELECT  sp.id as idpackage, COUNT(o.id) as sales,SUM(vp.validityperiod * pp.monthlycost) as `value`,

SUM(o.totalvalue) as valuewithproducts,CAST(AVG(a.count)as FLOAT) as avgoptionalproducts

INTO idpackagePI,salesPI,valuePI,valuewithproductsPI,avgoptionalproductsPI

FROM servicepackage sp

LEFT JOIN `order` o ON sp.id = o.idservicepackage

LEFT JOIN validityperiod vp ON o.idvalidityperiod = vp.id

LEFT JOIN packageperiod pp ON vp.id = pp.idvalidityperiod AND o.idservicepackage = pp.idservicepackage

LEFT JOIN (SELECT op.idorder as idorder, COUNT(op.idorder) as count  FROM orderproduct op GROUP BY idorder ) as a

ON a.idorder = o.id WHERE o.idservicepackage = new.idservicepackage AND o.paid = 1

GROUP BY sp.id;

UPDATE mv_packageSET idpackage = idpackagePI, sales = salesPI, `value` = valuePI, valuewithproducts =

valuewithproductsPI, avgoptionalproducts = avgoptionalproductsPI

WHERE mv_package.idpackage = new.idservicepackage;

END IF;

END
```

ORM relationship design



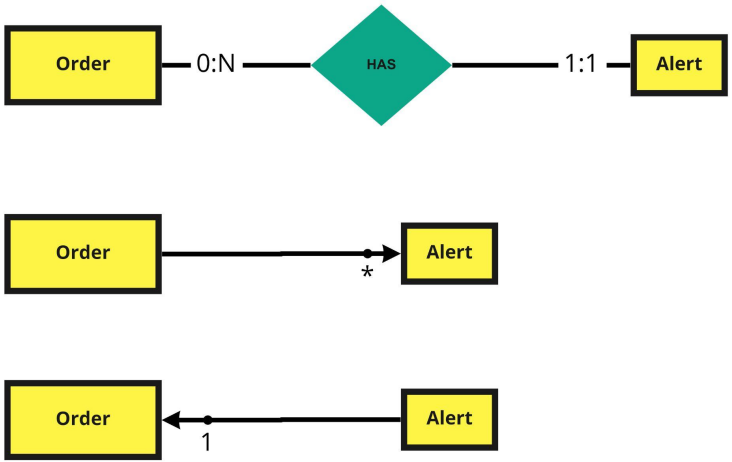
- Consumer → Order @OneToMany is necessary to list the order associated to the consumer.
- Order → Consumer @ManyToOne is necessary and it is used for persisting the entity.

ORM relationship design



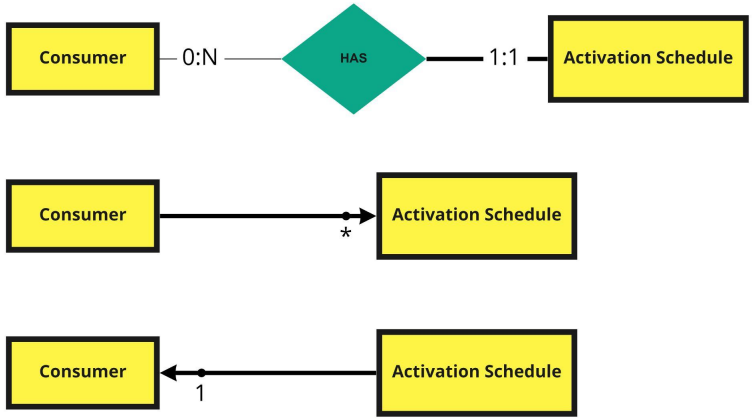
- Consumer → Alert @OneToMany is necessary to list the alerts generated in case of rejected transaction (#OfRejectedTransaction >= 3).
- Alert→ Consumer @ManyToOne is necessary to persist the alert including the details of the consumer (username and email) which is associated to the entity.

ORM relationship design



- Order → Alert @OneToMany is not necessary.
- Alert → Order @ManyToOne is necessary to persist the alert including the order associated to the entity.

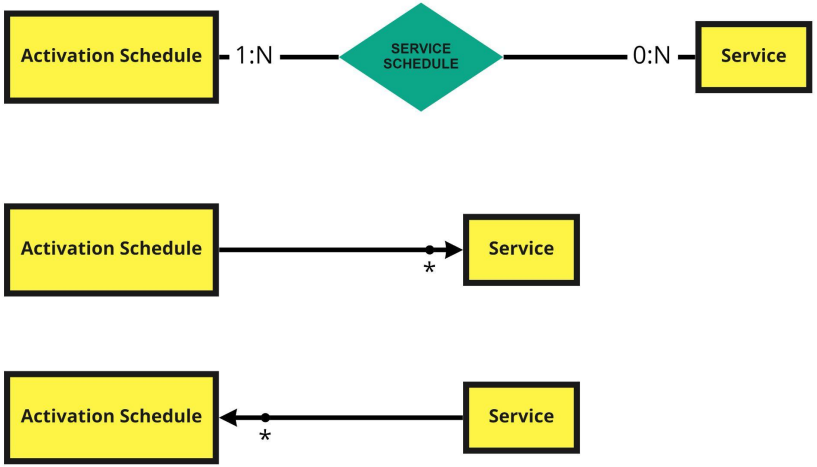
ORM relationship design



•Consumer → Activation Schedule
@OneToMany is necessary to persist the activation schedules of the consumer.

•Activation Schedule → Consumer
@ManyToOne is necessary to update the inverse relations for persisting the entity.

ORM relationship design

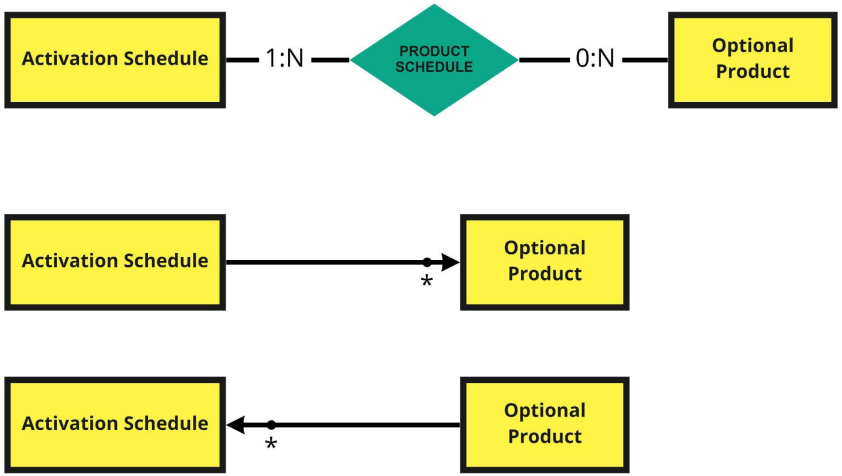


•Activation Schedule → Service
@ManyToMany is necessary for persisting the services associated to the activation schedule.

- FetchType = EAGER

•Service → Activation Schedule
@ManyToMany is not necessary.

ORM relationship design

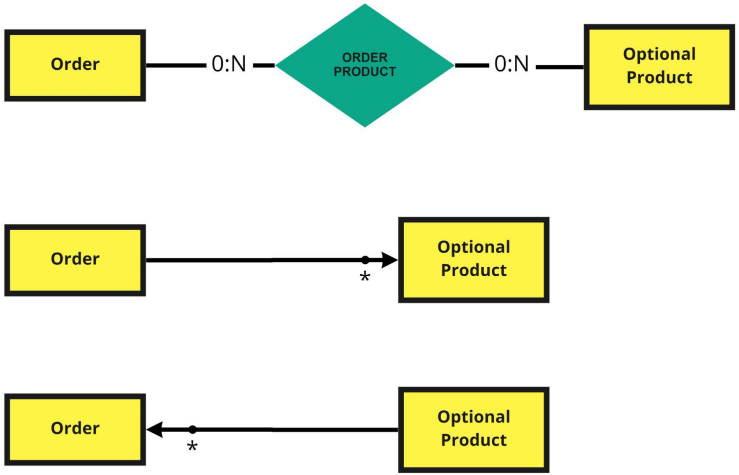


•Activation Schedule → Optional Product @ManyToMany is necessary for persisting the optional products associated to the activation schedule.

- FetchType = EAGER

•Optional Product → Activation Schedule @ManyToOne is not necessary.

ORM relationship design

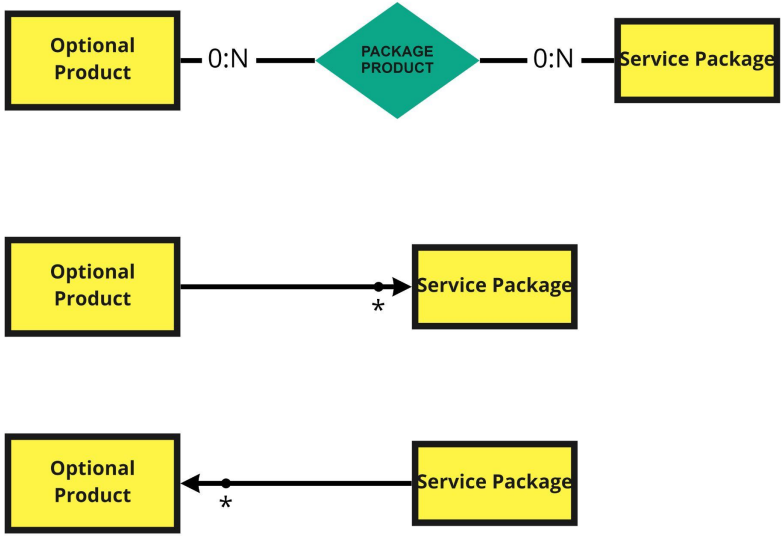


•Order → Optional Product @ManyToMany is necessary to list the optional product/s associated to the order (and for persisting them).

- FetchType = EAGER

•Optional Product → Order @ManyToMany is not necessary.

ORM relationship design

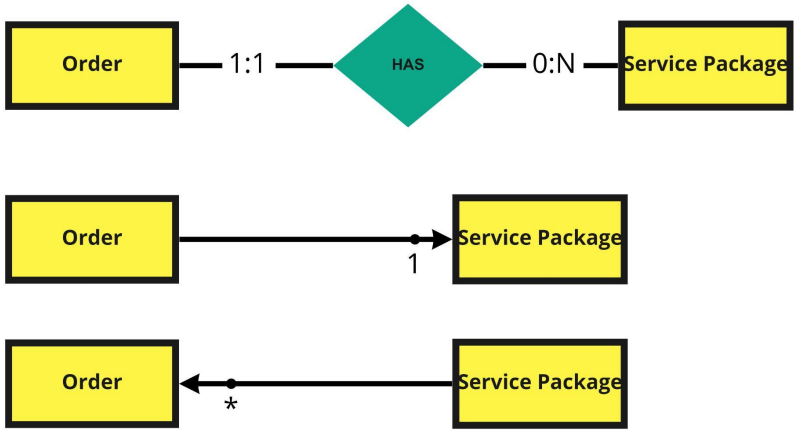


•Optional Product → Service Package
@ManyToMany is not necessary.

•Service Package → Optional Product
@ManyToMany is necessary to list the optional products associated to the service package (and for persisting them).

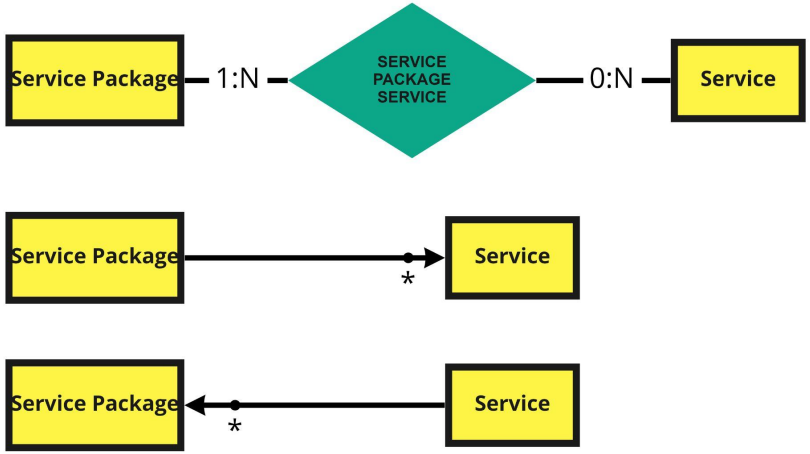
- FetchType = EAGER

ORM relationship design



- Order → Service Package @ManyToOne is necessary to list the service package associated to the order (and for persisting it).
- Service Package → Order @OneToMany is not necessary.

ORM relationship design

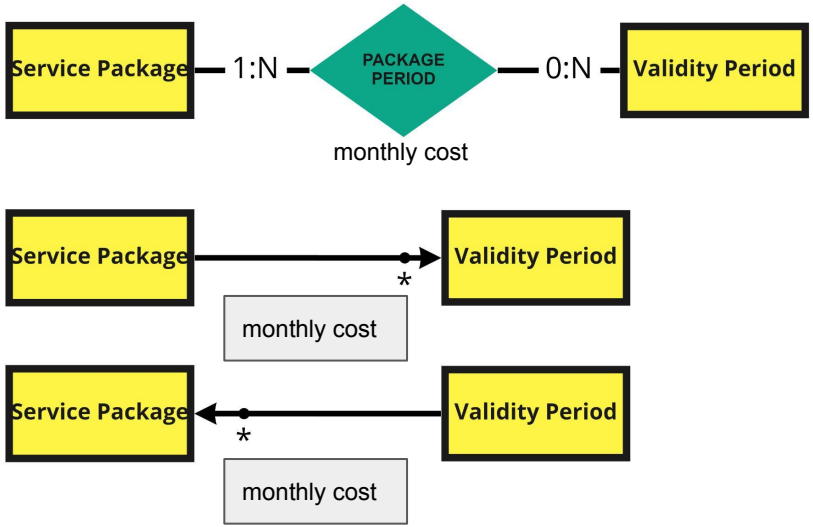


•Service Package → Service @ManyToMany is necessary to list the service associated to the service package (and for persisting them).

- FetchType = EAGER

•Service → Service Package @ManyToMany is not necessary.

ORM relationship design

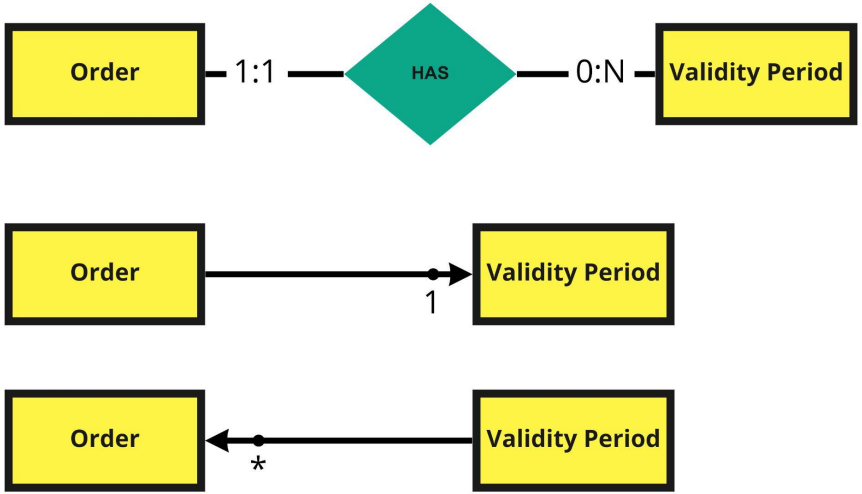


•Service Package → Validity Period needed to show all the validity periods of a service package:

- Owner = validity period;
- FetchType = EAGER;
- It must support the retrieval of the monthly cost of each validity period.

•Validity Period → Service Package
@ManyToMany is not necessary.

ORM relationship design



- Order → Validity Period @ManyToOne is necessary for showing the validity period associated to the order (and for persisting it).
- Validity Period → Order @OneToMany is not necessary.

Entities 1 (Activation Schedule)

```
@Entity
@Table(name = "activationschedule", schema="db2_savino_vinati")
@NamedQuery(name="Activationschedule.findAll", query="SELECT acsc FROM Activationschedule acsc")
public class Activationschedule implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Temporal(TemporalType.DATE)
    private Date activationdate;

    @Temporal(TemporalType.DATE)
    private Date deactivationdate;

    @ManyToOne
    @JoinColumn(name="iduser")
    private User user;


    @ManyToMany(fetch=FetchType.EAGER)
    @JoinTable(
            name="serviceschedule",
            joinColumns= { @JoinColumn(name="idactivation")},
            inverseJoinColumns= { @JoinColumn(name="idservice")}
    )
    private List<Service> services;

    @ManyToMany(fetch=FetchType.EAGER)
    @JoinTable(
            name="productschedule",
            joinColumns= { @JoinColumn(name="idactivation")},
            inverseJoinColumns= { @JoinColumn(name="idoptionalproduct")}
    )
    private List<Optionalproduct> optionalproducts;
```

Entities 2 (Alert)

```
@Entity
@Table(name = "alert", schema="db2_savino_vinati")
@NamedQuery(name="Alert.findAll", query="SELECT a FROM Alert a")
public class Alert implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @ManyToOne
    @JoinColumn(name="idinsolventuser")
    private Insolventuser insolventUser;

    @OneToOne
    @JoinColumn(name="idorder")
    private Order order;

    private float amount;

    @Column(name="active", insertable = false)
    private Boolean active;

    @Column(name="lastrejection", insertable = false, updatable = false)
    @Temporal(TemporalType.TIMESTAMP)
    private Date lastrejection;
```

Entities 3 (InsolventUser)

```
@Entity
@Table(name = "insolventuser", schema="db2_savino_vinati")
@NamedQuery(name="Insolventuser.findAll", query="SELECT iu FROM Insolventuser iu")
public class Insolventuser implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    private int id;

    private int failedpaymentcount;

    private boolean insolvent;

    @OneToOne
    @JoinColumn(name = "id")
    @MapsId
    private User user;

    @OneToMany(mappedBy="insolventUser", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<Alert> alerts;
```

Entities 4 (MvAlert)

```
@Entity
@Table(name="mv_alerts")
@NamedQuery(name="MvAlert.findAll", query="SELECT m FROM MvAlert m")
public class MvAlert implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @OneToOne
    @JoinColumn(name = "idalert")
    private Alert alert;
```

Entities 5 (MvBestProduct)

```
@Entity
@Table(name="mv_bestproduct")
@NamedQuery(name="MvBestproduct.findAll", query="SELECT m FROM MvBestproduct m")
public class MvBestproduct implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @OneToOne
    @JoinColumn(name = "idoptionalproduct")
    private Optionalproduct optionalproduct;

    private float value;

    private int sales;
```

Entities 6 (MvInsolventUser)

```
@Entity
@Table(name="mv_insolventuser")
@NamedQuery(name="MvInsolventUser.findAll", query="SELECT miu FROM MvInsolventUser miu")
public class MvInsolventUser implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @OneToOne
    @JoinColumn(name = "idinsolventuser")
    private Insolventuser insolventuser;
```

Entities 7 (MvPackage)

```
@Entity
@Table(name="mv_package")
@NamedQuery(name="MvPackage.findAll", query="SELECT m FROM MvPackage m")
public class MvPackage implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private float avgoptionalproducts;

    @OneToOne
    @JoinColumn(name = "idpackage")
    private Servicepackage servicepackage;

    private int sales;

    private float value;

    private float valuewithproducts;
```

Entities 8 (MvPackageperiod)

```
@Entity
@Table(name="mv_packageperiod")
@NamedQuery(name="MvPackageperiod.findAll", query="SELECT m FROM MvPackageperiod m")
public class MvPackageperiod implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @ManyToOne
    @JoinColumn(name = "idpackage")
    private Servicepackage servicepackage;

    @ManyToOne
    @JoinColumn(name = "idperiod")
    private Validityperiod validityperiod;

    private int sales;
```

Entities 9 (MvSuspendedorder)

```
@Entity
@Table(name="mv_suspendedorders")
@NamedQuery(name="MvSuspendedorder.findAll", query="SELECT m FROM MvSuspendedorder m")
public class MvSuspendedorder implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @OneToOne
    @JoinColumn(name = "idorder")
    private Order suspendedorder;
```

Entities 10 (Optionalproduct)

```
@Entity
@Table(name = "optionalproduct", schema="db2_savino_vinati")
@NamedQuery(name="Optionalproduct.findAll", query="SELECT op FROM Optionalproduct op")
public class Optionalproduct implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private float monthlyprice;

    private String name;
```

Entities 11 (Order)

```
@Entity
@Table(name = "order", schema="db2_savino_vinati")
@NamedQueries({
    @NamedQuery(name="Order.findAll", query="SELECT o FROM Order o"),
    @NamedQuery(name="Order.findFromId", query="SELECT o FROM Order o WHERE o.id = ?1"),
    @NamedQuery(name="Order.findRejectedOrders", query="SELECT o FROM Order o WHERE o.paid = false AND o.user = ?1")
})
public class Order implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private float totalvalue;

    @Column(name="datehour", insertable = false, updatable = false)
    @Temporal(TemporalType.TIMESTAMP)
    private Date datehour;

    @Temporal(TemporalType.DATE)
    private Date startdate;

    private boolean paid;

    @ManyToOne
    @JoinColumn(name="iduser")
    private User user;

    @ManyToOne
    @JoinColumn(name="idservicepackage")
    private Servicepackage servicepackage;

    @ManyToOne
    @JoinColumn(name="idvalidityperiod")
    private Validityperiod validityperiod;

    @ManyToMany(fetch=FetchType.EAGER)
    @JoinTable(
            name="orderproduct",
            joinColumns= { @JoinColumn(name="idorder")},
            inverseJoinColumns= { @JoinColumn(name="idoptionalproduct")}
    )
    private List<Optionalproduct> optionalproducts;
```

Entities 12 (Service)

```
@Entity
@Table(name = "service", schema="db2_savino_vinati")
@NamedQuery(name="Service.findAll", query="SELECT s FROM Service s")
public class Service implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String name;

    @ManyToOne
    @JoinColumn(name="idtype")
    private Servicetype type;

    @OneToOne(mappedBy="service")
    private Servicemobile servicemobile;

    @OneToOne(mappedBy="service")
    private Serviceinternet serviceinternet;
```

Entities 13 (Serviceinternet)

```
@Entity
@Table(name = "serviceinternet", schema="db2_savino_vinati")
@NamedQuery(name="Serviceinternet.findAll", query="SELECT si FROM Serviceinternet si")
public class Serviceinternet implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    private int id;

    private float feeextragbs;

    private int includedgbs;

    @OneToOne
    @JoinColumn(name = "id")
    @MapsId
    private Service service;
```

Entities 14 (Servicemobile)

```
@Entity
@Table(name = "servicemobile", schema="db2_savino_vinati")
@NamedQuery(name="Servicemobile.findAll", query="SELECT sm FROM Servicemobile sm")
public class Servicemobile implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    private int id;

    private float feeextraminutes;

    private float feeextrasms;

    private int includedminutes;

    private int includedsms;

    @OneToOne
    @JoinColumn(name = "id")
    @MapsId
    private Service service;
```

Entities 15 (Servicepackage)

```
@Entity
@Table(name = "servicepackage", schema="db2_savino_vinati")
@NamedQueries({
    @NamedQuery(name="Servicepackage.findAll", query="SELECT sp FROM Servicepackage sp"),
    @NamedQuery(name="Servicepackage.findFromName", query="SELECT sp FROM Servicepackage sp WHERE sp.name = ?1")
})
public class Servicepackage implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String name;

    @ManyToMany(fetch=FetchType.EAGER)
    @JoinTable(
            name="packageproduct",
            joinColumns= { @JoinColumn(name="idservicepackage")},
            inverseJoinColumns= { @JoinColumn(name="idoptionalproduct")}
    )
    private List<Optionalproduct> optionalproducts;


    @ElementCollection(fetch = FetchType.EAGER)
    @CollectionTable(name = "packageperiod", schema = "db2_savino_vinati", joinColumns = @JoinColumn(name = "idservicepackage"))
    @MapKeyJoinColumn(name = "idvalidityperiod")
    @Column(name = "monthlycost")
    private Map<Validityperiod, Float> validityperiods;

    @ManyToMany(fetch=FetchType.EAGER)
    @JoinTable(
            name="servicepackageservice",
            joinColumns= { @JoinColumn(name="idservicepackage")},
            inverseJoinColumns= { @JoinColumn(name="idservice")}
    )
    private List<Service> services;
```

Entities 16 (Servicetype)

```
@Entity
@Table(name = "servicetype", schema="db2_savino_vinati")
@NamedQuery(name="Servicetype.findAll", query="SELECT st FROM Servicetype st")
public class Servicetype implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String type;
```

Entities 17 (User)

```
@Entity
@Table(name = "user", schema="db2_savino_vinati")
@NamedQueries({
    @NamedQuery(name="User.findAll", query="SELECT u FROM User u"),
    @NamedQuery(name = "User.checkCredentials", query = "SELECT r FROM User r  WHERE r.username = ?1 and r.password = ?2"),
    @NamedQuery(name = "User.findFromUsername", query = "SELECT u FROM User u  WHERE u.username = ?1"),
    @NamedQuery(name = "User.findFromMail", query = "SELECT u FROM User u  WHERE u.mail = ?1")
    })
public class User implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String username;
    private String password;
    private String mail;

    @ManyToOne
    @JoinColumn(name="idusertype")
    private Usertype usertype;

    @OneToOne(mappedBy="user", cascade = CascadeType.ALL, orphanRemoval = true)
    private Insolventuser insolventuser;

    @OneToMany(mappedBy="user", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<Activationschedule> activationschedules;

    @OneToMany(fetch = FetchType.EAGER, mappedBy="user", cascade = CascadeType.ALL)
    private List<Order> orders;
```

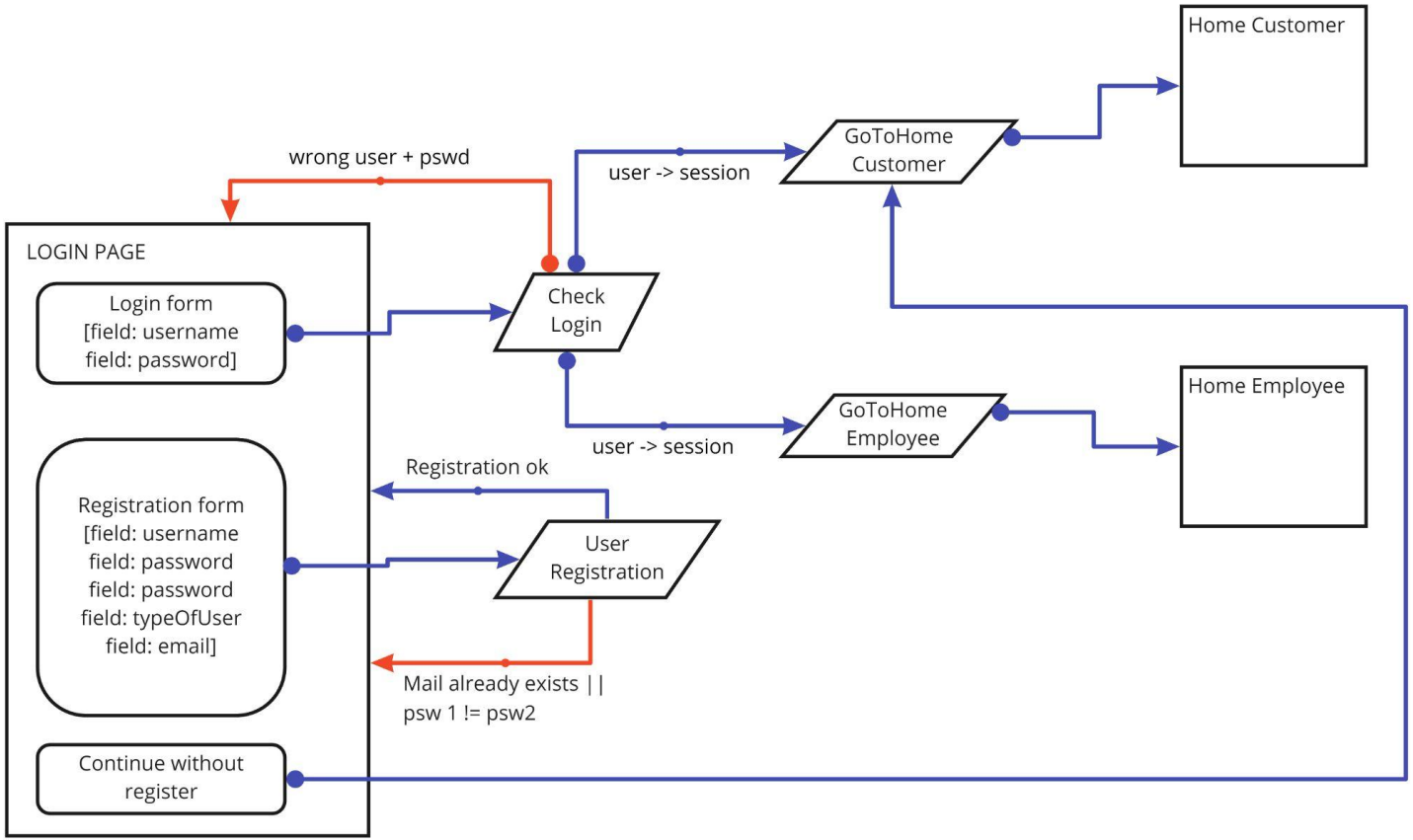
Entities 18 (Usertype)

```
@Entity
@Table(name = "usertype", schema="db2_savino_vinati")
@NamedQueries({
    @NamedQuery(name="Usertype.findAll", query="SELECT ut FROM Usertype ut"),
    @NamedQuery(name="Usertype.findFromType", query="SELECT ut FROM Usertype ut WHERE ut.usertype=?1")
})

public class Usertype implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String usertype;
```

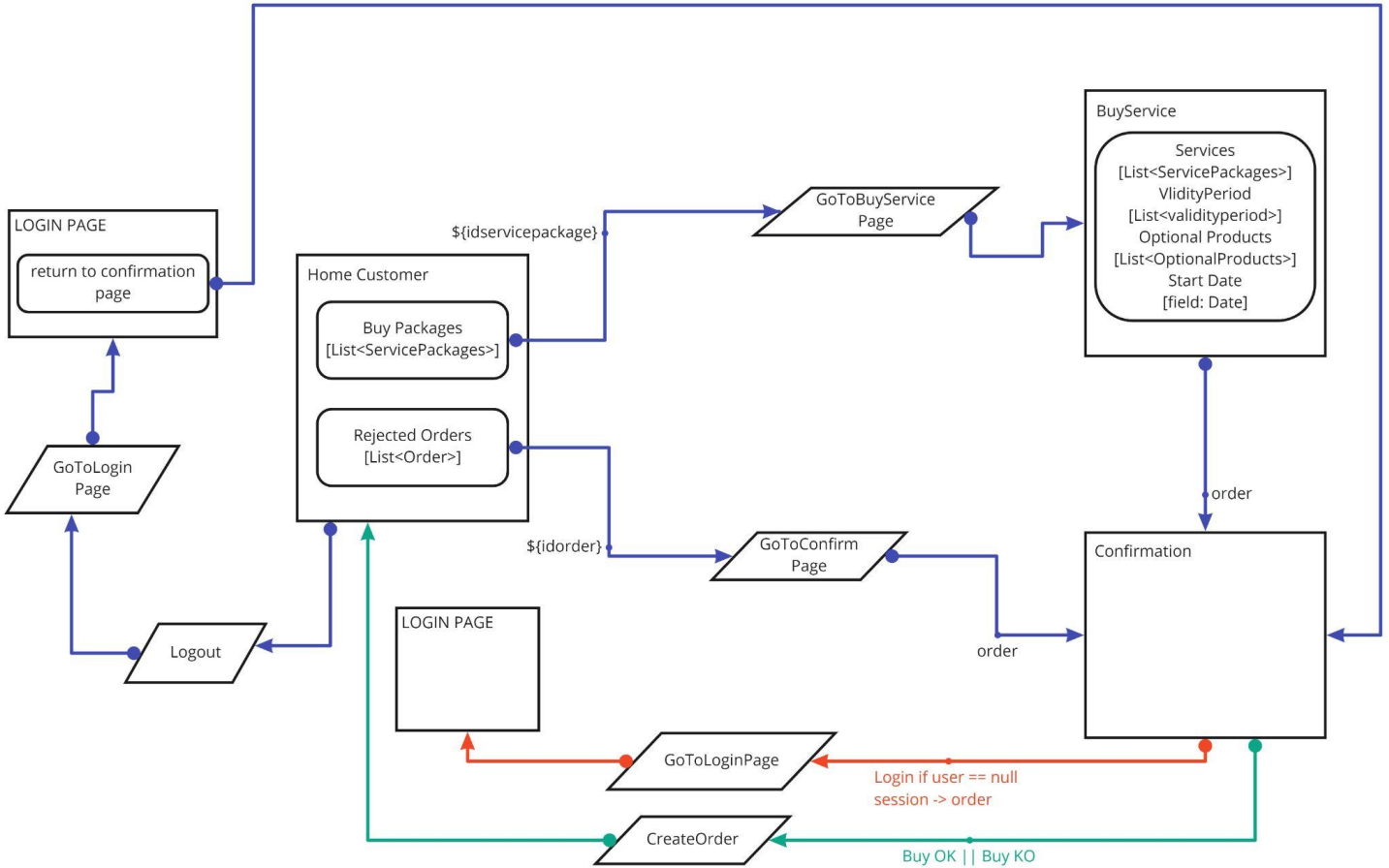
Entities 19 (Validityperiod)

```
@Entity
@Table(name = "validityperiod", schema="db2_savino_vinati")
@NamedQuery(name="Validityperiod.findAll", query="SELECT v FROM Validityperiod v")
public class Validityperiod implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private int validityperiod;
```

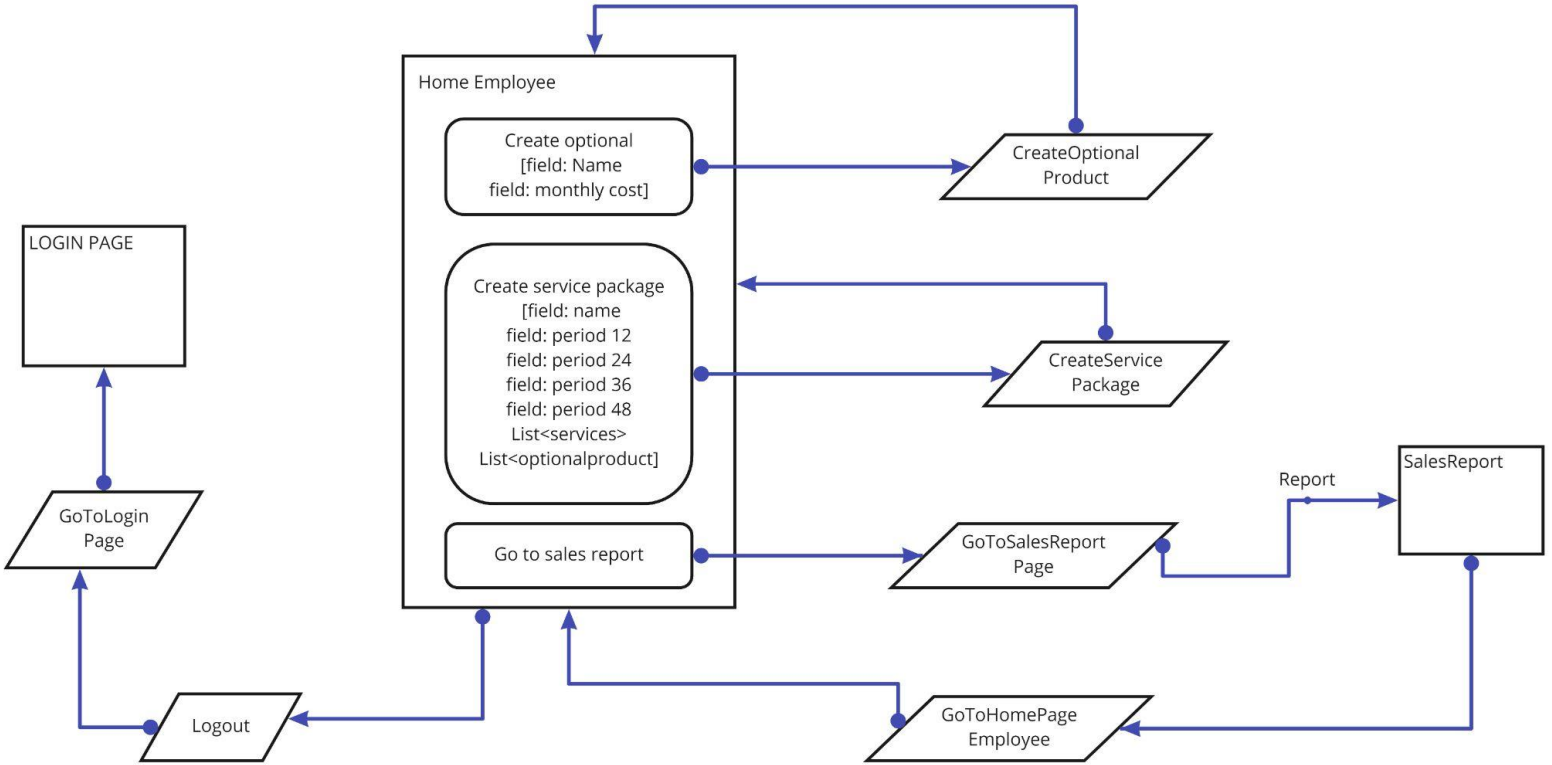
Interaction diagrams 1 (Login and Registration)



Interaction diagrams 2 (Buy service package)



Interaction diagrams 3 (Opt. Product, Service package creation and Sales Report)



Interaction diagrams considerations

- The logout buttons are presented only for homepage even if present also in other pages. They are omitted to enhance readability of graphs.

Components 1

- Client tier
  - CheckLogin: verifies credentials and stores user info the web session
  - CreateOptionalProduct
  - CreateOrder
  - CreateServicePackage
  - GoToBuyServicePage
  - GoToConfirmPage
  - GoToHomePageCustomer
  - GoToHomePageEmployee
  - GoToLoginPage
  - GoToSalesReportPage
  - Logout
  - UserRegistration

Components 2

- Business tier
  - @Stateless OptionalProductService
    - public void createOptionalProduct(String name, float monthlyprice)
  - @Stateless OrderService
    - public Order createOrderNoPersist(Integer idservicepackage, Integer idvalidityperiod, List<Integer> idoptionalproducts, Date date)
    - private Validityperiod checkValidityPeriod(Integer idvalidityperiod)
    - public Servicepackage checkServicePackage(Integer idservicepackage)
    - public float checkOptionalProducts(List<Integer> idoptionalproducts, Order order, float totalprice)
    - public void createOrder(Order order)
    - private void checkInsolventUserAndAlert(Order order)
    - private void createActivationSchedule(Order order, User user)
    - public Order findOrderById(Integer orderid)
    - public List<Order> findAllRejectedOrders(User user)

Components 3

- Business tier (continue)
  - @Stateless SalesReportService
    - public List<MvBestproduct> findAllbestproducts ()
    - public List<MvAlert> findAllmvalerts()
    - public List<MvSuspendedorder> findAllmvsuspendedorders()
    - public List<MvInsolventUser> findAllmvinsolventusers()
    - public List<MvPackageperiod> findAllmvpackageperiods()
    - public List<MvPackage> findAllmvpackages()
  - @Stateless ServicePackageService
    - public List<Servicepackage> findAllServicePackages()
    - public Servicepackage findServicePackageById(int serviceId)
    - public List<Validityperiod> findAllValidityperiods ()
    - public List<Service> findAllServices()
    - public List<Optionalproduct> findAllOptionalproducts()
    - public void checkDuplicatedName(String name)
    - public void createNewServicePackage

Components 4

- Business tier (continue)
  - @Stateless UserService
    - public User checkCredentials(String usrn, String pwd)
    - public Usertype getUsertype(String type)
    - public boolean checkDuplicatedMail(String username)
    - public boolean checkDuplicatedUsername(String username)
    - public void registerUser (String username, String password, String type, String mail)

UML sequence diagrams (for salient events, CreateOrder)