A11: Main Accesses to the database and transactions

The objective of this artifact is to present the main accesses to the database.

Module M01: Administration

SQL001

```
SELECT movie.imagePath, movie.name
FROM (SELECT movieID, COUNT(*) AS purchases FROM PurchaseMovie GROUP BY
movieID) subquery
WHERE subquery.movieID = movie.movieID
ORDER BY subquery.purchases DESC
LIMIT 4;
```

SQL101.1

```
SELECT * FROM member;
```

SQL101.2

```
SELECT movieID, movie.name, format.name AS formatname, EXTRACT(YEAR FROM
releasedate) AS releasedate
    FROM movie, format
    WHERE movie.formatID = format.formatID;
```

SQL105

```
DELETE FROM member WHERE member.memberid = :idOfUserToDelete;
```

SQL106

```
DELETE FROM movie WHERE movie.movieid = :idOfMovieToDelete;
```

SQL107

```
INSERT INTO Movie
(classification,description,imagePath,name,numberOfDiscs,price,region,releas
eDate,runtime,stock,formatID,studioID,genreID) VALUES
(:classification,:description,:imagePath,:name,:numberOfDiscs,:price,:region
,:releaseDate,:runtime,:stock,:formatID,:studioID,:genreID);
```

SQL108

```
UPDATE Movie
    SET
(classification,description,imagePath,name,numberOfDiscs,price,region,releas
eDate,runtime,stock,formatID,studioID,genreID) =
(:classification,:description,:imagePath,:name,:numberOfDiscs,:price,:region
,:releaseDate,:runtime,:stock,:formatID,:studioID,:genreID)
    WHERE Movie.movieid = :idOfMovieToUpdate;
```

SQL109

```
INSERT INTO member(email, firstName, lastName, password) VALUES
    (:email,:firstName,:lastName,:password);
```

SQL110

```
UPDATE Member
    SET (bannedMember,email,firstName,lastName,password) =
    (:bannedMember,:email,:firstName,:lastName,:password)
    WHERE Member.memberid = :idOfMemberToUpdate;
```

Module M02: Authentication

SQL202

```
SELECT * FROM Member WHERE email = :email AND password = :password;
```

SQL203

```
INSERT INTO member(email, firstName, lastName, password) VALUES
    (:email,:firstName,:lastName,:password);
```

Module M03: User Details

SQL301

```
INSERT INTO billinginformation (cityid, countryid, postcodeid, address,
fullname) VALUES
    (:cityid, :countryid, :postcodeid, :address, :fullname);
```

SQL302

```
UPDATE billinginformation
    SET (cityid, countryid, postcodeid, address, fullname) =
    (:cityid, :countryid, :postcodeid, :address, :fullname)
    WHERE billibginformation.billinginformationid =
:billinginformationIdToEdit;
```

SQL303

```
INSERT INTO deliveryadress (cityid, countryid, postcodeid, address,
fullname) VALUES
    (:cityid, :countryid, :postcodeid, :address, :fullname);
```

SQL304

```
UPDATE deliveryadress
    SET (cityid, countryid, postcodeid, address, fullname) =
    (:cityid, :countryid, :postcodeid, :address, :fullname)
    WHERE deliveryadress.deliveryadressid = :deliveryadressIdToEdit;
```

SQL305

```
INSERT INTO paymentinformation (creditcardnumber, cvc, expirationdate)
VALUES
    (:creditcardnumber, :cvc, :expirationdate);
```

SQL306

```
UPDATE paymentinformation
    SET (creditcardnumber, cvc, expirationdate) =
    (:creditcardnumber, :cvc, :expirationdate)
    WHERE paymentinformation.paymentinformationid =
:paymentinformationIdToEdit;
```

SQL307

```
UPDATE Member
    SET (bannedMember,email,firstName,lastName,password) =
    (:bannedMember,:email,:firstName,:lastName,:password)
    WHERE Member.memberid = :idOfMemberToUpdate;
```

SQL308

```
BEGIN readMemberInfo;
SET TRANSACTION ISOLATION LEVEL READ COMMITED READ ONLY NOT DEFERRABLE;
--The isolation level above refers to the read of only the commited rows
before the first instruction.

SELECT * FROM member
    WHERE member.memberId = :memberIdToGet;

SELECT movie.name, purchase.price, purchase.dateOfPurchase,
purchasemovie.quantity
    FROM purchase, purchasemovie, movie
    WHERE purchase.purchaseID = purchasemovie.purchaseID AND
purchasemovie.movieID = movie.movieID AND purchase.memberID = $memberID;

COMMIT readMemberInfo;
```

Module M04: Movie

SQL401

```
SELECT * FROM movie
    WHERE movie.movieid = :movieId;
```

SQL402

```
INSERT INTO wishlist (memberid, movieid) VALUES
    (:memberid, :movieid);
```

SQL403

```
INSERT INTO MovieCart (cartid, movieid, quantity) VALUES
    (:cartid, :movieid, :quantity);
```

SQL405

```
INSERT INTO Review (title, description, rating, movieid, memberid) VALUES
    (:title, :description, :rating, :movieid, :memberid);
```

SQL406

```
DELETE Review WHERE Review.reviewid = :reviewIdToDelete;
```

Module M05: Purchase

SQL501

```
SELECT movie.name, moviecart.quantity, movie.price, cart.totalCost
    FROM movie, movieCart, cart
    WHERE moviecart.cartid = cart.cartid AND
        movie.movieid = moviecart.movieid AND
        cart.cartId = :cartIdToView;
```

SQL502

```
DELETE FROM moviecart WHERE movieId = :movieToDelete AND cartId =
:cartToDelete;
```

SQL503

```
BEGIN checkoutTrans;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE READ WRITE NOT DEFERRABLE;
--The isolation level defined above (serializable) guarantees the integrity
of the cart and purchase data. In the case of inconsistencies the
istructions of this transaction are rolled back.

INSERT INTO purchase (dateofpurchase, price, memberid) VALUES (DATE('NOW'),
:price, :memberid);

INSERT INTO purchasemovie (purchaseId, movieid, quantity) SELECT * FROM
moviecart WHERE cartId = :cartId;

DELETE FROM moviecart WHERE cartId = :cartToDelete;

COMMIT checkoutTrans;
```

Module M06: Shop

SQL602

```
--The block of code bellow refers to the different searches that the user
can make, which means that for each search, only one of the SELECT
statements bellow is used.

SELECT movie.imagePath, movie.name, movie.price, format.name
```

```
    FROM movie, format
    WHERE movie.genreID = :genreID AND format.formatID = movie.formatID;

SELECT movie.imagePath, movie.name, movie.price, format.name
    FROM movie, format
    WHERE format.formatID = movie.formatID
    ORDER BY movie.price;

SELECT movie.imagePath, movie.name, movie.price, format.name
    FROM movie
    WHERE movie.releaseDate BETWEEN :date1 AND :date2 AND format.formatID =
movie.formatID

SELECT movie.imagePath, movie.name
    FROM (SELECT movieID, COUNT(*) AS purchases FROM PurchaseMovie GROUP BY
movieID) subquery
    WHERE subquery.movieID = movie.movieID
    ORDER BY subquery.purchases DESC
    LIMIT 4;

SELECT movie.imagePath, movie.name, movie.price, format.name
    FROM movie, format
    WHERE format.formatID = movie.formatID AND movie.name LIKE
:searchString;
```

SQL603

```
SELECT movie.imagePath, movie.name, movie.price, format.name
FROM movie, format
WHERE (to_tsvector('english', movie.description) @@ to_tsquery('english',
:searchString) OR movie.name LIKE :searchString)
      AND format.formatID = movie.formatID;
```