Projektaufgabe Filmdatenbank

Teil der Prüfungsleistung im Modul

Datenbanken 1 (EDV-Nr. 113210)

Sommersemester 2019

Projektgruppe **08** mit den Gruppenteilnehmern:

Heinrich, Leon	37560	MI-7
Dubiel, Sebastian	37438	MI-7
Vu, Hai	37480	MI-7

Hiermit bestätigen die oben genannten Gruppenteilnehmer, dass die Projektaufgabe ohne Unterstützung dritter und unter Nennung aller Quellen und eingesetzten Hilfsmittel erstellt wurde. Es ist uns bekannt, dass ein Verstoß gem. § 17 Abs. 5 Satz 1 zum Nichtbestehen der kompletten Prüfungsleistung im Modul Datenbanken 1 (EDV-Nr. 113210) führt.

Stuttgart, den 17. Juli 2019

Heinrich, Leon

Dubiel, Sebastian

Vu, Hai

SYSTEMBESCHREIBUNG

Tabellen

Unsere Filmdatenbank haben wir mit insgesamt 9 Tabellen aufgebaut. Die zentrale Tabelle stellt dabei die Tabelle `movies` dar:
Diese beinhaltet den künstlichen Primärschlüssel `movie_id`, sowie die Nicht-Schlüsselattribute `title` (Film-Titel), `plot_outline` (Film-Zusammenfassung) und `release_date` (Veröffentlichungsdatum). Des weiteren enthält sie die Fremdschlüssel `m_comp_id` und `m_dir_id`, welche die (N:1)-Beziehungen zwischen der Tabelle `movies` und den Tabellen `prodCompany` bzw. `director` realisieren. Dies begründen wir damit, dass ein Film von genau einer Filmproduktionsgesellschaft produziert wird und genau einen Regisseur hat, jedoch eine Filmproduktionsgesellschaft/ein Regisseur einen oder mehrere Filme produzieren/inszeniert.

Die Tabelle `director` beschreibt dabei die Eigenschaften des Regisseurs durch `dir_socSecNum` (Sozialversicherungsnummer), `dir_name` (Name) und `dir_birthday` (Geburtsdatum). Die Sozialversicherungsnummer stellt den Primärschlüssel dar, welcher in diesem Fall, ein nicht-künstlicher Schlüssel ist.

Die andere von `movies` referenzierte Tabelle `prodCompany` umfasst wiederum alle Filmproduktionsgesellschaften und enthält neben der eindeutigen ID (`comp_id`) Informationen über den Namen der Filmproduktionsgesellschaft `(comp_name)` und den Fremdschlüssel `p_address_id`, welcher wiederum eine PK/FK-Beziehung (N:1) zur Tabelle `address` ermöglicht. Wir haben angenommen, dass sich evtl. mehrere Filmproduktionsgesellschaften/Tochterfirmen an einem Ort angesiedelt haben und somit eine Adresse zu mehreren Filmproduktionsgesellschaften zugeordnet werden kann.

In der Tabelle `address` sind dann genauere Informationen über die Adresse der Filmproduktionsgesellschaft/en festgehalten. Dazu gehören die Postleitzahl (`postCode`), Straße (`street`), Stadt/Ort (`city`), Land (`country`) und der Primärschlüssel `address_id`.

Hinzu kommt eine Tabelle `movieRole`. Diese beinhaltet Daten zur jeweiligen Rolle in einem Film. So sind hier neben der ID `role_id` auch der Name der Rolle (`role_name`) eingetragen. Zusätzlich wurden hier zwei Fremdschlüssel `mR_actor_id` und `mR_movie_id` definiert. `mR_movie_id` bezieht sich dabei auf die Tabelle `movies` (N:1) und `mR_actor_id` stellt eine Referenz zur Tabelle `actor` dar (N:1). Diese Beziehungen beruhen darauf, dass eine Rolle zu genau einem Film gehört (Filmfortsetzungen/mehrteilige Filme haben wir ausgeschlossen), jedoch in einem Film mehreren Rollen vorkommen. Andererseits wird eine

Rolle nur von einem Schauspieler belegt (Remakes ausgeschlossen), allerdings kann ein Schauspieler mehrere Rollen annehmen.

Die Daten aller beteiligten Schauspieler sind in der Tabelle `actor` gespeichert, wozu `actor_socSecNum` (Sozialversicherungsnummer), `actor_name` (Name) und `actor_birthday` (Geburtstag) zählen. `actor_socSecNum` ist hier der Primärschlüssel und ist somit zusammen mit `dir_socSecNum` der einzige nicht-künstliche Schlüssel in unserem Projekt.

Als nächstes ist die Tabelle `mov_gen` zu nennen. Diese setzt die einzige "logische" N:M-Beziehung in unserem Projekt in zwei 1:N-Beziehungen um. So sind neben der ID `mov_gen_id`, eine Referenz auf die Tabelle `movies (mg_movie_id)`, sowie auf die Tabelle `genre (mg_genre_id)` gegeben. Durch `mov_gen` konnte die Tatsache umgesetzt werden, dass ein Film mehrere Genres haben kann, aber auch ein Genre mehreren Filmen zugeordnet wird. `mg_movie_id` und `mg_genre_id` haben wir zudem kombiniert als `UNIQUE` definiert, um redundante Genre-Zuweisungen zum gleichen Film zu verhindern.

Der referenzierte Primärschlüssel `genre_id` und der Name des Genres (`genre_name`) bilden dabei die Tabelle `genre`.

Unsere letzte Tabelle `movie_grosses` ist für das Speichern der Einnahmen eines Filmes vorgesehen, wobei wir uns nur auf Kinoeinnahmen (`movie_theater`) beschränkt haben. Des weiteren wird das zu den Einnahmen zugehörige Datum (`grossDate`) und eine ID (`grosses_id`) gespeichert. Zudem enthält sie auch den benötigten Fremdschlüssel (`g_movie_id`) um die Zuordnung zur Tabelle `movie` zu ermöglichen. Folglich kann ein Film mehrere Einnahmen generieren (N:1-Beziehung).

Datentypen

In unseren Tabellen haben wir alle geforderten Datentypen verwendet:
Einerseits haben wir zum Anlegen von IDs *ganze Zahlen* (`INTEGER`) verwendet, welche numerisch exakt sein sollen.
Andererseits haben wir den Datentyp `NUMBER` gewählt, da man durch die Angabe von zwei *Nachkommastellen* sehr gut Geldbeträge, in unserem Fall für die Kinoeinnahmen (`movie_theater`) darstellen kann.
Für *Zeichenketten mit variabler Länge* (vor allem Namen) haben wir den Datentyp `VARCHAR(x)`, mit einer für den Zweck jeweils ausreichenden Länge gewählt.
Bei *Zeichenketten* mit kurzer und *fester Länge* haben wir uns für den Datentyp `CHAR(x)` entschieden. Dazu zählen bei uns Sozialversicherungsnummern oder Postleitzahlen.

Um ein *Datum* in einer Tabelle zu implementieren, wie Geburtsdaten oder die Daten der jeweiligen Kinoverkaufszahlen (`grossDate`) haben wir standardmäßig den Datentyp `DATE` verwendet.

Aufbau SQL Script

Um ein völlig restart-fähiges Skript zu garantieren beginnen wir als erstes mit dem Löschen (`DROP`) aller `CONSTRAINTs`. Zuerst werden hierbei alle `FOREIGN KEYs`, die `UNIQUE CONSTRAINTs` und erst dann alle `PRIMARY KEYs` gedroppt. Diese Reihenfolge mussten wir wählen, damit sich beim Entfernen der PKs keine FKs mehr auf diese beziehen.
Nach dem Löschen aller `CONSTRAINTs` können alle Tabellen entfernt werden.
Danach werden zunächst alle Tabellen ohne ihre PK/FK-`CONSTRAINTs` erstellt, jedoch mit den zugehörigen Datentypen und Column-`CONSTRAINTS` `NULL / NOT NULL`. Erst im Anschluss erhalten sie dann die jeweils nötigen Table-`CONSTRAINTs`. Dies dient wiederum der Restart-Fähigkeit des Skriptes.
Dabei verteilen wir die `CONSTRAINTs` in umgekehrter Reihenfolge zum Löschen, also erst die PKs, dann `UNIQUE` und zuletzt die FKs.
Im Anschluss können die Tabellen dann befüllt werden (`INSERT`), wobei wir mindestens vier `INSERTs` pro Tabelle getätigt haben, teilweise auch mehr, um 1:N-Beziehungen zu verdeutlichen und mehrere Einträge mit gleichem FK zu speichern.
Vor einigen `INSERTs` sind `SEQUENCEs` und `TRIGGER` wiederzufinden. Hierbei gibt es zwei Arten von `TRIGGER`.
Die erste Art von `TRIGGER` (`<..>_incrementId`) ist dafür verantwortlich, IDs stets um 1 zu inkrementieren. Dies ist dafür nützlich, um viele Einträge automatisch mit gleichmäßig aufsteigenden und vor allem eindeutigen Werten zu versehen. Dazu haben wir zunächst alle IDs der Einträge in Tabellen, welche den `TRIGGER` verwenden, auf 0 gesetzt. Beim Ausführen des Programms werden diese dann durch den `TRIGGER` angepasst.
Der andere `TRIGGER` (`prevent_future_grosses`) überprüft vor dem Einfügen/Ändern von Einträgen in die/der Tabelle `movie_grosses` das Veröffentlichungsdatum des zugehörigen Filmes. Falls das Datum noch nicht gesetzt wurde (`NULL` ist) oder das Datum in der Zukunft liegt, jedoch Einnahmen zugewiesen werden (`movie_theater`), wird eine eigens erstellte Exception geworfen, die das Einfügen verhindert. In der Realität könnten dadurch Fehler bei der Interpretation der Einspielergebnisse verhindert werden.

Extra SQL-Skript

Getrennt vom restlichen SQL-Skript, haben wir unsere `SELECT/UPDATE`-Befehle, sowie die `TRIGGER`-Tests gespeichert

Select-Befehle

Hier haben wir neben dem Ausgeben aller Daten einer Tabelle, einfache Joins, Vergleichsoperationen, `LEFT OUTER JOINS`, als auch `SUM/COUNT`-Funktionen, kombiniert mit `GROUP BY` (Gruppierung) und `DESC` (absteigende Sortierung), verwendet.

Update-Befehle

Hier wurden einzelne Attributwerte ersetzt (u.a. auch Datumswerte), vertauscht oder auch durch Addition erhöht.

Testen von Triggern

Die Increment-`TRIGGER` haben wir durch `SELECT`-Abfragen getestet, welche aufeinanderfolgende Entitäten mit der geforderten ID (die größer als 0 ist) ausgibt, wodurch die fortlaufende Inkrementierung durch den `TRIGGER` bestätigt wird.
Den `TRIGGER prevent_future_grosses` haben wir durch einen `INSERT`-Befehl getestet, welcher einem Film, dessen Veröffentlichungsdatum noch nicht angegeben wurde, positive Kinoverkaufszahlen zuordnet, woraufhin wie erwartet die Exception geworfen wird.

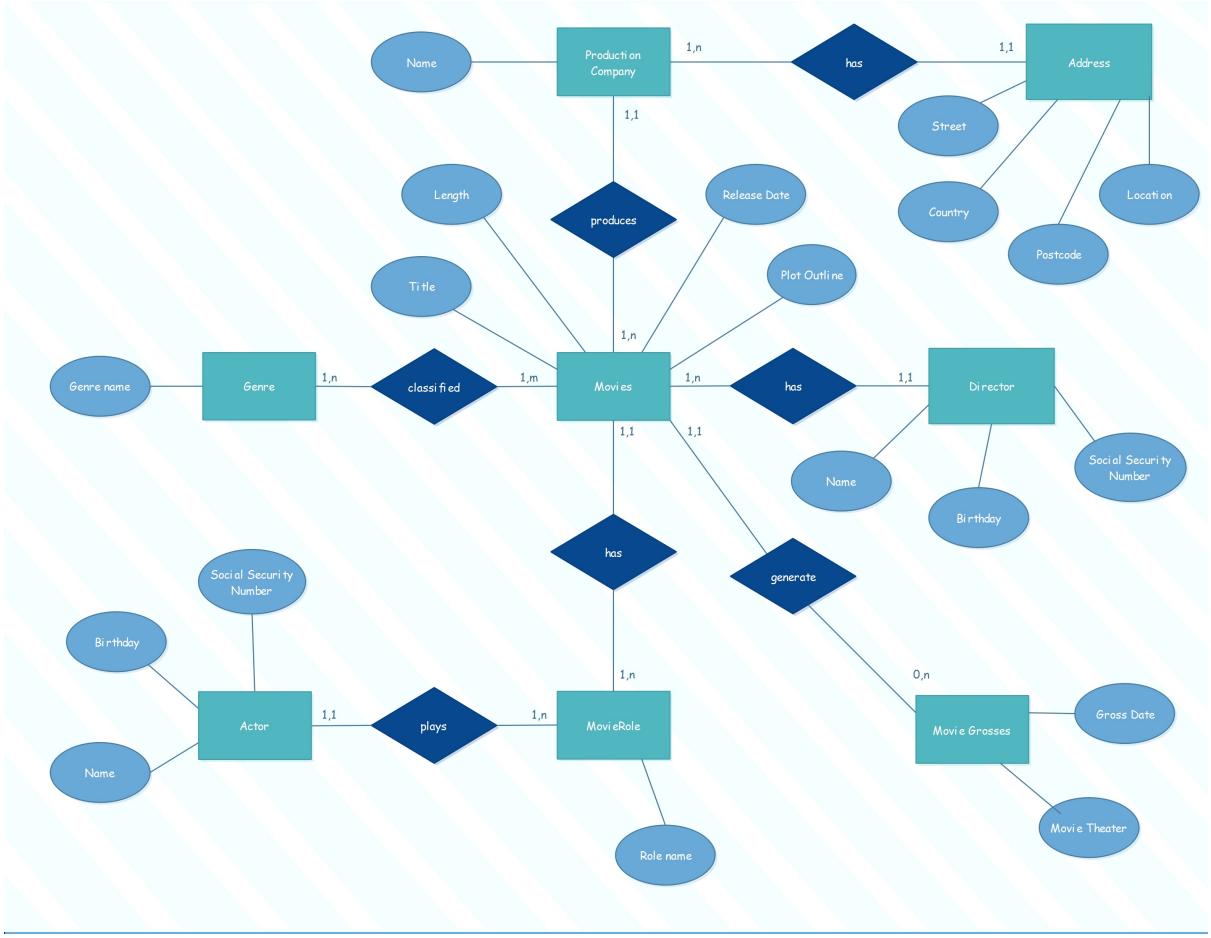
Nutzung von Indizes

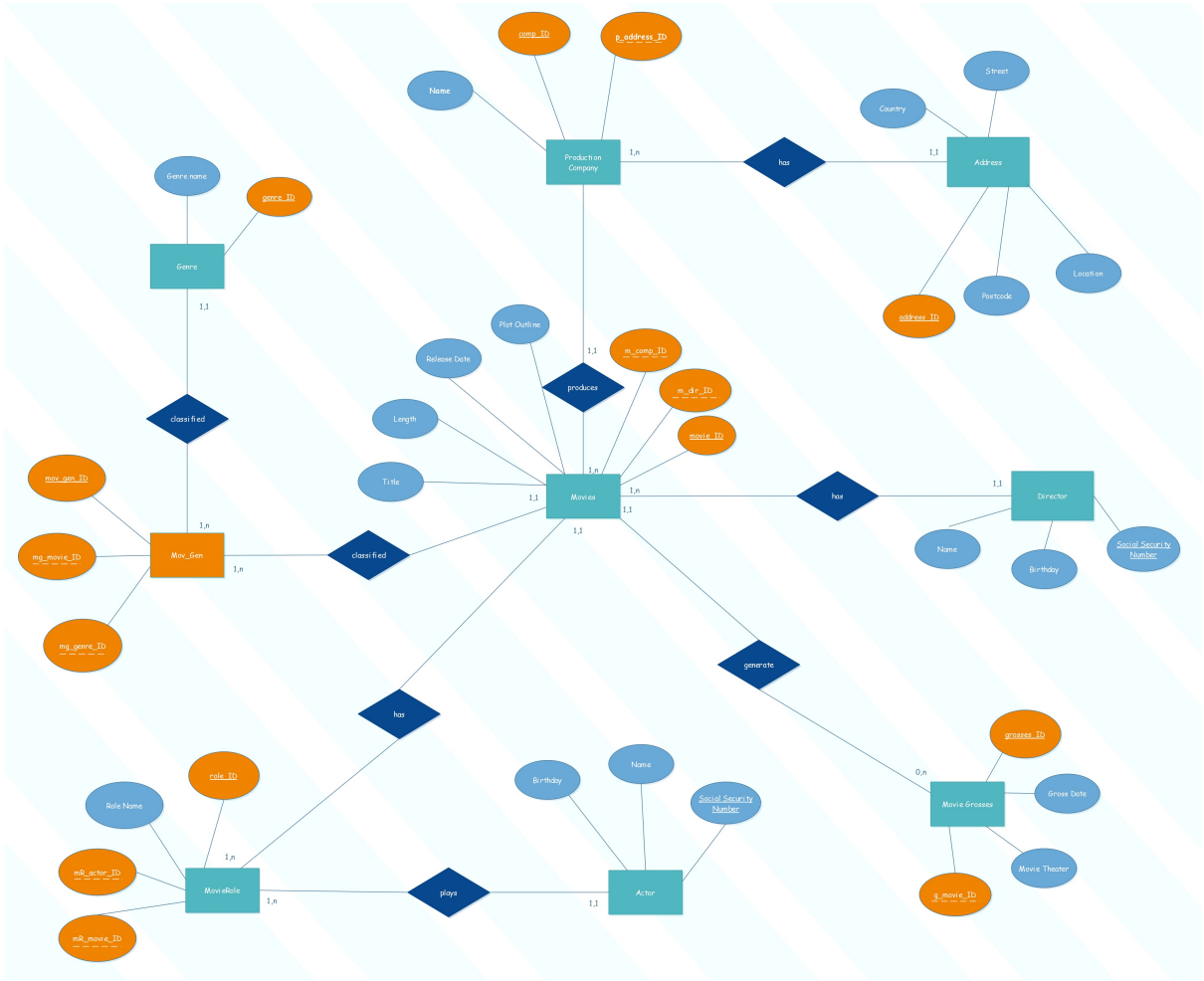
Wir haben uns gegen eine Index entschieden, da wir in jeder Tabelle durch die Verwendung eines Primärschlüssels eine implizite Indexierung der Daten vornehmen.
Ein expliziter Index würde sich jedoch bspw. in der Tabelle `movie_grosses` eignen, da diese durch fortlaufendes Eintragen von Umsätzen sehr schnell wachsen würde und hier außerdem wiederholte bestimmte Lesevorgänge vorkommen würden.

Quellen

Lediglich für den Increment-`TRIGGER` haben wir Hilfe aus dem Internet genutzt (https://stackoverflow.com/questions/11296361/how-to-create-id-with-auto-increment-on-oracle). Ansonsten haben wir für das gesamte SQL-Skript ausschließlich die in der VL und den Übungen gewonnenen Kenntnisse verwendet.

Logisches & Physisches ER-Diagramm





(Primärschlüssel = unterstrichen, Fremdschlüssel = gestrichelt)

```
--- Movie-database ---

-- DROP CONSTRAINTs --

ALTER TABLE mov_gen DROP CONSTRAINT mov_gen_fk2;
ALTER TABLE mov_gen DROP CONSTRAINT mov_gen_fk1;
ALTER TABLE movieRole DROP CONSTRAINT movieRole_fk2;
ALTER TABLE movieRole DROP CONSTRAINT movieRole_fk1;
ALTER TABLE movie_grosses DROP CONSTRAINT movies_grosses_fk1;
ALTER TABLE movies DROP CONSTRAINT movies_fk2 ;
ALTER TABLE movies DROP CONSTRAINT movies_fk1 ;
ALTER TABLE prodCompany DROP CONSTRAINT prodCompany_fk1 ;

ALTER TABLE mov_gen DROP CONSTRAINT mov_gen_unique;
ALTER TABLE movieRole DROP CONSTRAINT movieRole_unique;

ALTER TABLE mov_gen DROP CONSTRAINT mov_gen_pk;
ALTER TABLE movieRole DROP CONSTRAINT movieRole_pk;
ALTER TABLE movie_grosses DROP CONSTRAINT  movie_gross_pk ;
ALTER TABLE movies DROP CONSTRAINT movies_pk;
ALTER TABLE prodCompany DROP CONSTRAINT prodCompany_pk ;
ALTER TABLE director DROP CONSTRAINT director_pk;
ALTER TABLE actor DROP CONSTRAINT actor_pk ;
ALTER TABLE genre DROP CONSTRAINT genre_pk ;
ALTER TABLE address DROP CONSTRAINT address_pk ;


-- DROP TABLEs --

DROP TABLE mov_gen;
DROP TABLE movieRole;
DROP TABLE movie_grosses;
DROP TABLE movies;
DROP TABLE prodCompany;
DROP TABLE director;
DROP TABLE actor;
DROP TABLE genre;
DROP TABLE address;


-- CREATE TABLEs --

CREATE TABLE address
(
    address_id INTEGER,
    postCode   CHAR(5) NOT NULL,
    street     VARCHAR(40) NOT NULL,
    city   VARCHAR(40) NOT NULL,
    country    VARCHAR(40) NOT NULL
);

CREATE TABLE genre
(
    genre_id   INTEGER,
    genre_name VARCHAR(40) NOT NULL
);


CREATE TABLE actor
(
    actor_socSecNum      CHAR(15) ,
    actor_name     VARCHAR(40) NOT NULL,
    actor_birthday DATE
);

CREATE TABLE director
(
```

```
    dir_socSecNum    CHAR(15),
    dir_name     VARCHAR(40) NOT NULL,
    dir_birthday DATE
);

CREATE TABLE prodCompany
(
    comp_id INTEGER,
    comp_name   VARCHAR(40) NOT NULL,
    p_address_id INTEGER NOT NULL
);

CREATE TABLE movies
(
    movie_id INTEGER,
    title    VARCHAR(40) NOT NULL,
    m_comp_id INTEGER NOT NULL,
    m_dir_id CHAR(15) NOT NULL,
    plot_outline VARCHAR(500),
    release_date DATE
);

CREATE TABLE movie_grosses
(
    grosses_id INTEGER,
    g_movie_id INTEGER NOT NULL ,
    grossDate DATE NOT NULL,
    movie_theater NUMBER(15,2) NOT NULL
);

CREATE TABLE movieRole
(
    role_id   INTEGER,
    role_name VARCHAR(40) NOT NULL,
    mR_actor_id CHAR(15) NOT NULL,
    mR_movie_id INTEGER NOT NULL

);

CREATE TABLE mov_gen
(
    mov_gen_id INTEGER,
    mg_movie_id INTEGER NOT NULL,
    mg_genre_id INTEGER NOT NULL
);


-- ADD CONSTRAINTs --

ALTER TABLE address ADD CONSTRAINT address_pk PRIMARY KEY (address_id);
ALTER TABLE genre ADD CONSTRAINT genre_pk PRIMARY KEY (genre_id);
ALTER TABLE actor ADD CONSTRAINT actor_pk PRIMARY KEY (actor_socSecNum);
ALTER TABLE director ADD CONSTRAINT director_pk PRIMARY KEY
(dir_socSecNum);
ALTER TABLE prodCompany ADD CONSTRAINT prodCompany_pk PRIMARY KEY
(comp_id);
ALTER TABLE movies ADD CONSTRAINT movies_pk PRIMARY KEY (movie_id);
ALTER TABLE movie_grosses ADD CONSTRAINT  movie_gross_pk PRIMARY KEY
(grosses_id);
ALTER TABLE movieRole ADD CONSTRAINT movieRole_pk PRIMARY KEY (role_id);
ALTER TABLE mov_gen ADD CONSTRAINT mov_gen_pk PRIMARY KEY (mov_gen_id);

ALTER TABLE mov_gen ADD CONSTRAINT mov_gen_unique UNIQUE (mg_movie_id,
mg_genre_id);
ALTER TABLE movieRole ADD CONSTRAINT movieRole_unique UNIQUE (mR_actor_id,
mR_movie_id);
```

```
ALTER TABLE prodCompany ADD CONSTRAINT prodCompany_fk1 FOREIGN KEY
(p_address_id) REFERENCES address (address_id);
ALTER TABLE movies ADD CONSTRAINT movies_fk1 FOREIGN KEY (m_comp_id)
REFERENCES  prodCompany (comp_id);
ALTER TABLE movies ADD CONSTRAINT movies_fk2 FOREIGN KEY (m_dir_id)
REFERENCES   director (dir_socSecNum);
ALTER TABLE movie_grosses ADD CONSTRAINT movies_grosses_fk1 FOREIGN KEY
(g_movie_id) REFERENCES movies (movie_id);
ALTER TABLE movieRole ADD CONSTRAINT movieRole_fk2 FOREIGN KEY
(mR_movie_id) REFERENCES movies (movie_id);
ALTER TABLE movieRole ADD CONSTRAINT movieRole_fk1 FOREIGN KEY
(mR_actor_id) REFERENCES actor (actor_socSecNum);
ALTER TABLE mov_gen ADD CONSTRAINT mov_gen_fk1 FOREIGN KEY (mg_movie_id)
REFERENCES movies (movie_id);
ALTER TABLE mov_gen ADD CONSTRAINT mov_gen_fk2 FOREIGN KEY (mg_genre_id)
REFERENCES genre (genre_id);


-- INSERTs --

INSERT INTO address (address_id, postCode, street, city, country) VALUES
(1,13629,'Langestraße 24','Berlin','Germany');
INSERT INTO address (address_id, postCode, street, city, country) VALUES
(2,91522,'New York Street 56','New York','USA');
INSERT INTO address (address_id, postCode, street, city, country) VALUES
(3,90187,'Saas Street 31','Los Angeles','USA');
INSERT INTO address (address_id, postCode, street, city, country) VALUES
(4,92364,'Old Town Road 42','Dublin','Ireland');


DROP SEQUENCE genre_incrementId;
CREATE SEQUENCE genre_incrementId
    MINVALUE 0
    START WITH 1;
CREATE OR REPLACE TRIGGER genre_incrementId_trigger
    BEFORE INSERT
    ON genre
    FOR EACH ROW
BEGIN
    SELECT genre_incrementId.nextval
    INTO :new.genre_id
    FROM DUAL;
END;

INSERT INTO genre (genre_id, genre_name) VALUES (0,'Action');
INSERT INTO genre (genre_id, genre_name) VALUES (0,'Drama');
INSERT INTO genre (genre_id, genre_name) VALUES (0,'Adventure');
INSERT INTO genre (genre_id, genre_name) VALUES (0,'Thriller');
INSERT INTO genre (genre_id, genre_name) VALUES (0,'Music');
INSERT INTO genre (genre_id, genre_name) VALUES (0,'Fantasy');
INSERT INTO genre (genre_id, genre_name) VALUES (0,'Science-Fiction');
INSERT INTO genre (genre_id, genre_name) VALUES (0,'Horror');
INSERT INTO genre (genre_id, genre_name) VALUES (0,'Western');


INSERT INTO actor (actor_socSecNum, actor_name, actor_birthday) VALUES ('65
051195 S 003', 'Debastian Subiel', to_date('05.11.1995', 'DD.MM.YYYY'));
INSERT INTO actor (actor_socSecNum, actor_name, actor_birthday) VALUES ('18
200492 L 402', 'Heon Leinrich', to_date('20.04.1992', 'DD.MM.YYYY'));
INSERT INTO actor (actor_socSecNum, actor_name, actor_birthday) VALUES ('33
021070 H 753', 'Marta Higgins', to_date('02.10.1970', 'DD.MM.YYYY'));
INSERT INTO actor (actor_socSecNum, actor_name, actor_birthday) VALUES ('20
060659 J 697', 'Isabelle Freljord', to_date('06.06.1959', 'DD.MM.YYYY'));


INSERT INTO director (dir_socSecNum, dir_name, dir_birthday) VALUES ('25
240660 M 367', 'Steven Spielberg', to_date('24.06.1960', 'DD.MM.YYYY'));
```

```
INSERT INTO director (dir_socSecNum, dir_name, dir_birthday) VALUES ('10
120387 F 477', 'Tom Fischer', to_date('12.03.1987', 'DD.MM.YYYY'));
INSERT INTO director (dir_socSecNum, dir_name, dir_birthday) VALUES ('15
010779 J 367', 'Mark Jones', to_date('01.07.1979', 'DD.MM.YYYY'));
INSERT INTO director (dir_socSecNum, dir_name, dir_birthday) VALUES ('24
171219 H 127', 'Vai Hu', to_date('24.12.1993', 'DD.MM.YYYY'));


INSERT INTO prodCompany (comp_id, comp_name, p_address_id) VALUES (1, 'The
Dalt Wisney Company', 2);
INSERT INTO prodCompany (comp_id, comp_name, p_address_id) VALUES (2,
'Warner Sis.', 4);
INSERT INTO prodCompany (comp_id, comp_name, p_address_id) VALUES (3,
'Parodont Pictures', 3);
INSERT INTO prodCompany (comp_id, comp_name, p_address_id) VALUES (4, '21st
Century Fox', 1);


INSERT INTO movies (movie_id, title, m_comp_id, m_dir_id, plot_outline,
release_date)
VALUES (1,'Bames Jond, 700 Silver Ear',1,'25 240660 M 367','The Queen was
threatened and Jond has to save her in a great thrilling adventure!
',TO_DATE('12.05.1987','DD.MM.YYYY'));
INSERT INTO movies (movie_id, title, m_comp_id, m_dir_id, plot_outline,
release_date)
VALUES (2,'Bragon Doll the Movie',2,'10 120387 F 477','Son Goku wants to
bekome the king of the pirates! Can he do
it?',TO_DATE('23.08.2013','DD.MM.YYYY'));
INSERT INTO movies (movie_id, title, m_comp_id, m_dir_id, plot_outline,
release_date)
VALUES (3,'Adventure in the mountains',3,'15 010779 J 367','Anton had a
beautiful life in the mountains, but suddenly everything
changed!',TO_DATE('08.04.2015','DD.MM.YYYY'));
INSERT INTO movies (movie_id, title, m_comp_id, m_dir_id, plot_outline,
release_date)
VALUES (4,'Patrick Starts darkest stories',4,'24 171219 H
127',NULL,TO_DATE('23.10.2019','DD.MM.YYYY'));
INSERT INTO movies (movie_id, title, m_comp_id, m_dir_id, plot_outline,
release_date)
VALUES (5,'The One and Only Movie',1,'15 010779 J 367',NULL,NULL);


CREATE OR REPLACE TRIGGER prevent_future_grosses
    BEFORE INSERT OR UPDATE ON movie_grosses
    REFERENCING OLD AS OLD NEW AS NEW
    FOR EACH ROW
DECLARE
    relDate DATE;
    refId INT;
    invalid_grosses exception;
    PRAGMA EXCEPTION_INIT(invalid_grosses, -20550);
BEGIN
    SELECT m.release_date INTO relDate FROM movies m WHERE :NEW.g_movie_id
= m.movie_id;
    SELECT m.movie_id INTO refId FROM movies m WHERE :NEW.g_movie_id =
m.movie_id;
    IF (relDate > SYSDATE OR relDate IS NULL) AND :NEW.g_movie_id = refId
AND :NEW.movie_theater > 0 THEN
        RAISE_APPLICATION_ERROR(-20550, 'Grosses for a movie entered which
has not been published yet!');
    END IF;
END;

DROP SEQUENCE grosses_incrementId;
CREATE SEQUENCE grosses_incrementId
    MINVALUE 0
    START WITH 1;
```

```
CREATE OR REPLACE TRIGGER grosses_incrementId_trigger
    BEFORE INSERT
    ON movie_grosses
    FOR EACH ROW
BEGIN
    SELECT grosses_incrementId.nextval
    INTO :NEW.grosses_id
    FROM DUAL;
END;

INSERT INTO movie_grosses (grosses_id, g_movie_id, grossDate,
movie_theater) VALUES
(0, 1, TO_DATE('12.05.1987','DD.MM.YYYY'),10648270.00);
INSERT INTO movie_grosses (grosses_id, g_movie_id, grossDate,
movie_theater) VALUES
(0, 1, TO_DATE('13.05.1987','DD.MM.YYYY'),23046879.00);
INSERT INTO movie_grosses (grosses_id, g_movie_id, grossDate,
movie_theater) VALUES
(0, 1, TO_DATE('14.05.1987','DD.MM.YYYY'),31164579.00);
INSERT INTO movie_grosses (grosses_id, g_movie_id, grossDate,
movie_theater) VALUES
(0, 1, TO_DATE('15.05.1987','DD.MM.YYYY'),30134687.00);
INSERT INTO movie_grosses (grosses_id, g_movie_id, grossDate,
movie_theater) VALUES
(0, 1, TO_DATE('16.05.1987','DD.MM.YYYY'),29637540.00);
INSERT INTO movie_grosses (grosses_id, g_movie_id, grossDate,
movie_theater) VALUES
(0, 2, TO_DATE('23.08.2013','DD.MM.YYYY'),1046978.00);
INSERT INTO movie_grosses (grosses_id, g_movie_id, grossDate,
movie_theater) VALUES
(0, 2, TO_DATE('24.08.2013','DD.MM.YYYY'),2405367.00);
INSERT INTO movie_grosses (grosses_id, g_movie_id, grossDate,
movie_theater) VALUES
(0, 2, TO_DATE('25.08.2013','DD.MM.YYYY'),4603161.00);
INSERT INTO movie_grosses (grosses_id, g_movie_id, grossDate,
movie_theater) VALUES
(0, 2, TO_DATE('26.08.2013','DD.MM.YYYY'),4401349.00);
INSERT INTO movie_grosses (grosses_id, g_movie_id, grossDate,
movie_theater) VALUES
(0, 2, TO_DATE('27.08.2013','DD.MM.YYYY'),5000134.00);
INSERT INTO movie_grosses (grosses_id, g_movie_id, grossDate,
movie_theater) VALUES
(0, 3, TO_DATE('08.04.2015','DD.MM.YYYY'),6064371.00);
INSERT INTO movie_grosses (grosses_id, g_movie_id, grossDate,
movie_theater) VALUES
(0, 3, TO_DATE('09.04.2015','DD.MM.YYYY'),8643077.00);
INSERT INTO movie_grosses (grosses_id, g_movie_id, grossDate,
movie_theater) VALUES
(0, 3, TO_DATE('10.04.2015','DD.MM.YYYY'),9640225.00);
INSERT INTO movie_grosses (grosses_id, g_movie_id, grossDate,
movie_theater) VALUES
(0, 3, TO_DATE('11.04.2015','DD.MM.YYYY'),11376660.00);
INSERT INTO movie_grosses (grosses_id, g_movie_id, grossDate,
movie_theater) VALUES
(0, 3, TO_DATE('12.04.2015','DD.MM.YYYY'),9613664.00);


DROP SEQUENCE movieRole_incrementId;
CREATE SEQUENCE movieRole_incrementId
    MINVALUE 0
    START WITH 1;
CREATE OR REPLACE TRIGGER movieRole_incrementId_trigger
    BEFORE INSERT
    ON movieRole
    FOR EACH ROW
BEGIN
    SELECT movieRole_incrementId.nextval
```

```
    INTO :new.role_id
    FROM DUAL;
END;

INSERT INTO movieRole (role_id, role_name, mR_actor_id, mR_movie_id) VALUES
(0, 'Bames Jond', '65 051195 S 003', 1);
INSERT INTO movieRole (role_id, role_name, mR_actor_id, mR_movie_id) VALUES
(0, 'Queen', '33 021070 H 753', 1);
INSERT INTO movieRole (role_id, role_name, mR_actor_id, mR_movie_id) VALUES
(0, 'Gong Suko', '33 021070 H 753', 2);
INSERT INTO movieRole (role_id, role_name, mR_actor_id, mR_movie_id) VALUES
(0, 'Anton aus Tirol', '18 200492 L 402' , 3);
INSERT INTO movieRole (role_id, role_name, mR_actor_id, mR_movie_id) VALUES
(0, 'Patrick Start', '65 051195 S 003', 4);
INSERT INTO movieRole (role_id, role_name, mR_actor_id, mR_movie_id) VALUES
(0, 'Sandy', '20 060659 J 697', 4);
INSERT INTO movieRole (role_id, role_name, mR_actor_id, mR_movie_id) VALUES
(0, 'Gary', '33 021070 H 753', 4);
INSERT INTO movieRole (role_id, role_name, mR_actor_id, mR_movie_id) VALUES
(0, 'Der Echte', '18 200492 L 402', 5);


DROP SEQUENCE mov_gen_incrementId;
CREATE SEQUENCE mov_gen_incrementId
    MINVALUE 0
    START WITH 1;
CREATE OR REPLACE TRIGGER mov_gen_incrementId_trigger
    BEFORE INSERT
    ON mov_gen
    FOR EACH ROW
BEGIN
    SELECT mov_gen_incrementId.nextval
    INTO :new.mov_gen_id
    FROM DUAL;
END;

INSERT INTO mov_gen (mov_gen_id, mg_movie_id, mg_genre_id) VALUES (0, 1,
3);
INSERT INTO mov_gen (mov_gen_id, mg_movie_id, mg_genre_id) VALUES (0, 1,
2);
INSERT INTO mov_gen (mov_gen_id, mg_movie_id, mg_genre_id) VALUES (0, 1,
5);
INSERT INTO mov_gen (mov_gen_id, mg_movie_id, mg_genre_id) VALUES (0, 2,
7);
INSERT INTO mov_gen (mov_gen_id, mg_movie_id, mg_genre_id) VALUES (0, 3,
6);
INSERT INTO mov_gen (mov_gen_id, mg_movie_id, mg_genre_id) VALUES (0, 3,
3);
INSERT INTO mov_gen (mov_gen_id, mg_movie_id, mg_genre_id) VALUES (0, 4,
8);
INSERT INTO mov_gen (mov_gen_id, mg_movie_id, mg_genre_id) VALUES (0, 4,
1);
INSERT INTO mov_gen (mov_gen_id, mg_movie_id, mg_genre_id) VALUES (0, 5,
4);


COMMIT;
```

```
-- SELECTs --

SELECT * FROM movies;

SELECT m.title, a.actor_name, mR.role_name FROM movies m, actor a,
movieRole mR WHERE m.movie_id = mR.mR_movie_id AND mR.mR_actor_id =
a.actor_socSecNum;

SELECT m.title, release_date FROM movies m WHERE release_date <
to_date('23.03.2015', 'DD.MM.YYYY');

SELECT m.title, sum(g.movie_theater) AS Sum_Of_Grosses FROM movies m,
movie_grosses g WHERE m.movie_id = g.g_movie_id GROUP BY m.title ORDER BY
Sum_Of_Grosses DESC;

SELECT m.title, COUNT(*) AS Number_of_genres FROM movies m, genre g,
mov_gen mg WHERE m.movie_id = mg.mg_movie_id AND mg.mg_genre_id =
g.genre_id GROUP BY m.title ORDER BY Number_of_genres DESC;

SELECT m.title, m.release_date, mg.grossDate, mg.movie_theater FROM movies
m LEFT OUTER JOIN movie_grosses mg ON m.movie_id = mg.g_movie_id;


--UPDATEs --

UPDATE movies SET plot_outline = 'A bunch of scary stories told by your
favourite character' WHERE movie_id = 4;

UPDATE genre SET genre_name = 'Adventure' WHERE genre_id = 4;
UPDATE genre SET genre_name = 'Thriller' WHERE genre_id = 3;

UPDATE movies SET release_date = to_date('23.12.2018', 'DD.MM.YYYY') WHERE
movie_id = 1;

UPDATE movie_grosses SET movie_theater =+ 10000.00 WHERE g_movie_id = 1;


COMMIT;


-- TRIGGER-tests --

-- IncrementId-TRIGGER --

SELECT * FROM movie_grosses WHERE grosses_id = 3;
SELECT * FROM movie_grosses WHERE grosses_id = 4;
SELECT * FROM movie_grosses WHERE grosses_id = 5;


-- TRIGGER: "prevent_future_grosses" --

INSERT INTO movie_grosses(grosses_id, g_movie_id, grossDate, movie_theater)
VALUES
(0, 5, SYSDATE, 66754.00);
```

Bewertungsschema (bitte als letzte Seite der Ausarbeitung einbinden):

Kriterium		**max.**	**ist**	**Kommentar**
Relative Qualität		15		
Spezifikation / Systembeschreibung		15		
ER-Modell inkl. formaler Korrektheit im Sinn der Normalisierung		10		
Sinnvolle Nutzung von Normierungsdaten		5		
Lauffähigkeit		5		
Restartfähigkeit		5		
Vollständigkeit der Testdaten		5		
Umfang der Umsetzung des Datenmodells		-	-	
	Anzahl der Tabellen	5		
	Vollständige Nutzung der Datentypen	5		
	Sinnvolle Nutzung von Randbedingungen	5		
	Sinnvolle Nutzung von Indexierung	5		
Summe				