CMPE321

Project 3

Report

Ahmet Fırat Gamsız - 2020400180

Yunus Emre Özdemir - 2020400153

**Step 1: Non-Trivial Functional Dependencies (FDs)**

**User**

{username} -> {username, password, name, surname}

**Player**

{username} -> {username, date\_of\_birth, height, weight}

**Coach**

{username} -> {username, nationality}

**Jury**

{username} -> {username, nationality}

**Position**

{position\_ID} -> {position\_ID, position\_name}

**Channel**

{channel\_ID} -> {channel\_ID, channel\_name}

**Team**

{team\_ID} -> {team\_ID, team\_name, channel\_ID}

**Stadium**

{stadium\_id} -> {stadium\_id, stadium\_name, stadium\_country}

**Date\_Time**

-

**Match\_Sessions**

{session\_ID} -> {session\_ID, team\_ID, stadium\_id, date\_of, timeslot}

{stadium\_id, date\_of, timeslot} -> {session\_ID, team\_ID, stadium\_id, date\_of, timeslot }

**Can\_Play**

-

**Registered**

-

**In\_Contract**

{team\_ID} -> {team\_ID, username, contract\_start, contract\_finish} TODO

**Rates**

{session\_ID} -> {session\_ID, username, rating}

**Player\_Plays\_In**

{username, session\_ID} -> {username, session\_ID ,position\_ID}

**Step 2: BCNF Checking**

Every non-trivial functional dependency in every table is only key constraint (unique identifies candidate keys, so they are also taken as key constraint). Since key constraints do not affect BCNF, all tables are in BCNF. Some tables don’t have any non-trivial FD, because they only consist of primary keys.

**Step 3:** **3NF Checking**

Since all tables are in BCNF they are also in 3NF.

**Step 4: Refinement**

Since our tables were already in BCNF we didn’t need to conduct refinement. However, we included the DBManager table (which is also in BCNF) in designs.

**TRIGGERS**

We included following triggers to capture the constraints that we were not able to handle in Project 1. Other constraints were handled in backend and frontend. For the implementation we used the scheme that was given by the TA rather than our own design. These triggers are also from the implementation; therefore, there are differences in table names and attributes.

DELIMITER //

DROP TRIGGER IF EXISTS check\_overlap; //

CREATE TRIGGER check\_overlap

BEFORE INSERT ON MatchSession

FOR EACH ROW

BEGIN

DECLARE m\_count *INT*;

SELECT COUNT(\*) INTO m\_count

FROM MatchSession

WHERE (time\_slot = NEW.time\_slot OR

time\_slot = NEW.time\_slot + 1 OR

time\_slot - 1 = NEW.time\_slot) AND

stadium\_ID = NEW.stadium\_ID AND

*date* = NEW.date;

IF m\_count > 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Overlapping session';

END IF;

END;

This trigger checks for slot overlap. It prevents matches happen in same stadium, same date and overlapping slot.

DELIMITER //

DROP TRIGGER IF EXISTS channel\_name\_integrity; //

CREATE TRIGGER channel\_name\_integrity

BEFORE INSERT ON Team

FOR EACH ROW

BEGIN

DECLARE m\_count *INT*;

SELECT COUNT(\*) INTO m\_count

FROM Team

WHERE (channel\_ID = NEW.channel\_ID AND channel\_name != NEW.channel\_name) OR

(channel\_ID != NEW.channel\_ID AND channel\_name = NEW.channel\_name);

IF m\_count > 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Channel name integrity violation';

END IF;

END; //

This trigger check for the channel names. Channel IDs must always match the channel name.

DELIMITER //

DROP TRIGGER IF EXISTS check\_stadium; //

CREATE TRIGGER check\_stadium

BEFORE INSERT ON MatchSession

FOR EACH ROW

BEGIN

DECLARE m\_count *INT*;

SELECT COUNT(\*) INTO m\_count

FROM MatchSession

WHERE ((stadium\_ID = NEW.stadium\_ID) AND (stadium\_name != NEW.stadium\_name or stadium\_country != NEW.stadium\_country));

IF m\_count > 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Stadium integrity violation';

END IF;

END; //

Since stadium\_ID’s define the stadium name and country, this constraint provides stadiums with same ID to have same name and country.

DELIMITER //

DROP TRIGGER IF EXISTS rating\_change\_limit; //

CREATE TRIGGER rating\_change\_limit

BEFORE UPDATE ON MatchSession

FOR EACH ROW

BEGIN

DECLARE m\_count *INT*;

SELECT COUNT(\*) INTO m\_count

FROM MatchSession

WHERE rating IS NOT NULL AND rating != NEW.rating AND session\_ID = NEW.session\_ID;

IF m\_count > 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Rating already set!';

END IF;

END; //

Juries can rate only once and they cannot edit or delete. This constraint locks the rating entered.

DELIMITER //

DROP TRIGGER IF EXISTS team\_existence\_trigger; //

CREATE TRIGGER team\_existence\_trigger

BEFORE INSERT ON PlayerTeams

FOR EACH ROW

BEGIN

DECLARE m\_count *INT*;

SELECT COUNT(\*) INTO m\_count

FROM Team

WHERE team\_ID = NEW.team;

IF team\_count = 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Team does not exist!';

END IF;

END; //

Player must play on teams existing. This constraint prevents adding any entry for non-existing teams.

DELIMITER //

DROP TRIGGER IF EXISTS position\_existance\_trigger; //

CREATE TRIGGER position\_existance\_trigger

BEFORE INSERT ON PlayerPositions

FOR EACH ROW

BEGIN

DECLARE m\_count *INT*;

SELECT COUNT(\*) INTO m\_count

FROM Position

WHERE position\_ID = NEW.position;

IF m\_count = 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Position does not exist!';

END IF;

END; //

Players must play on positions existing. This constraint prevents adding any entry for non-existing positions.

ALTER TABLE MatchSession

ADD *CONSTRAINT* timeslot\_limit

*CHECK* (0 <= time\_slot AND time\_slot <= 3);

Since only available time slots are from 0 to 3 (matches starting at 4 can’t last two slots), this check prevents any other entries.