# Database Management Systems

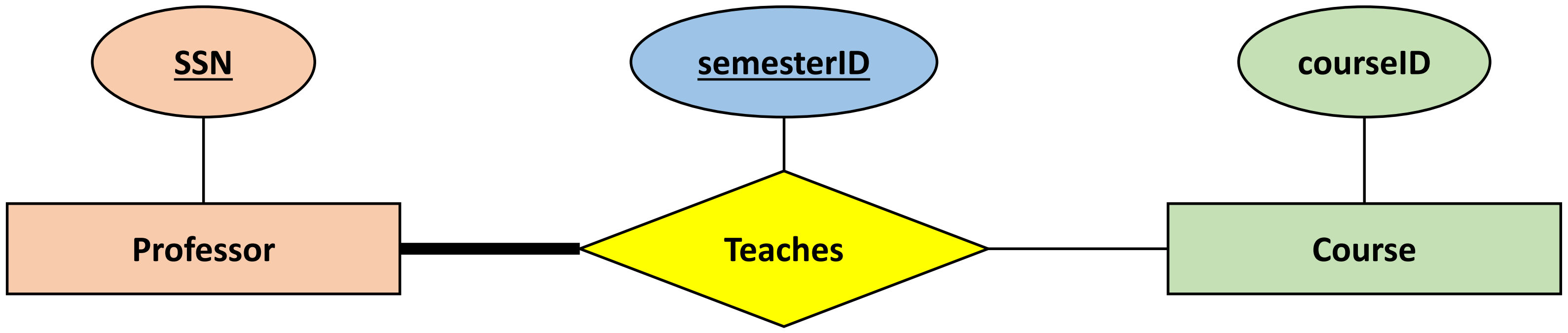
Ehsan Noei
e.noei@utoronto.ca

UNIVERSITY OF TORONTO

## Integrity Constraints (Review)

- An IC describes conditions that every *legal instance* of a relation must satisfy.
  - Inserts/deletes/updates that violate IC's are disallowed.
  - Can be used to ensure application semantics (e.g., *sid* is a key), or prevent inconsistencies (e.g., *sname* has to be a string, *age* must be < 200)
- *Types of IC's*: Domain constraints, primary key constraints, foreign key constraints, general constraints.

## Total Participation



## One Solution

```
START TRANSACTION;
BEGIN;
    SET FOREIGN_KEY_CHECKS=0;

    CREATE TABLE A (B INT(11),C INT (11),PRIMARY KEY (B),
    FOREIGN KEY (B) REFERENCES C(D));

    CREATE TABLE C (D INT(11),E INT (11),PRIMARY KEY (D),
    FOREIGN KEY (D) REFERENCES A(B));

    SET FOREIGN_KEY_CHECKS=1;
COMMIT;
```

## General Constraints

```
CREATE TABLE  Sailors
     ( sid  INTEGER,
     sname  CHAR(10),
     rating  INTEGER,
     age  REAL,
     PRIMARY KEY  (sid),
     CHECK  ( rating >= 1
          AND rating <= 10 )
```

# General Constraints

- Useful when more general ICs than keys are involved.
- Can use queries to express constraint.
- Constraints can be named.

```
CREATE TABLE  Reserves
        ( sname  CHAR(10),
        bid  INTEGER,
        day  DATE,
        PRIMARY KEY  (bid,day),
        CONSTRAINT  noInterlakeRes
        CHECK  ('Interlake' <>
                ( SELECT  B.bname
                FROM  Boats B
                WHERE  B.bid=bid)))
```

## Constraints Over Multiple Relations

```
CREATE TABLE  Sailors
   ( sid  INTEGER,
   sname  CHAR(10),
   rating  INTEGER,
   age  REAL,
   PRIMARY KEY  (sid),
   CHECK
   ( (SELECT COUNT (S.sid) FROM Sailors S)
   + (SELECT COUNT (B.bid) FROM Boats B) < 100 ))
```

## Constraints Over Multiple Relations

```
CREATE ASSERTION  smallClub
CHECK
( (SELECT COUNT (S.sid) FROM Sailors S)
+ (SELECT COUNT (B.bid) FROM Boats B) < 100 )
```

## Trigger

- Trigger: procedure that starts automatically if specified changes occur to the DBMS
- Three parts:
  - Event (activates the trigger)
  - Condition (tests whether the trigger should run)
  - Action (what happens if the trigger runs)

## The trigger acts as an accumulator, summing the values inserted into one of the columns of the table.

```
CREATE TABLE account (
acct_num INT(11),
amount INT(11)
);

CREATE TRIGGER ins_sum
BEFORE INSERT ON account
FOR EACH ROW SET @sum = @sum + NEW.amount;
```

## The trigger acts as an accumulator, summing the values inserted into one of the columns of the table.

```
SET @sum = 0;

INSERT INTO account
VALUES(137,14.98),(141,1937.50),(97,-100.00);

SELECT @sum AS 'Total amount inserted';
```

## The trigger acts as an accumulator, summing the values inserted into one of the columns of the table.

```
SET @sum = 0;

INSERT INTO account
VALUES(137,14.98),(141,1937.50),(97,-100.00);

SELECT @sum AS 'Total amount inserted';
```

| Total amount inserted |
|---|
| 1852.48 |

## An UPDATE trigger that checks the new value to be used for updating each row, and modifies the value to be within the range from 0 to 100.

```
CREATE TRIGGER upd_check BEFORE UPDATE ON
account
    FOR EACH ROW
    BEGIN
        IF NEW.amount < 0 THEN
                SET NEW.amount = 0;
        ELSEIF NEW.amount > 100 THEN
                SET NEW.amount = 100;
        END IF;
    END;
```

## Drop Trigger

- DROP TRIGGER ins_sum;