Data Wrangling and Data Analysis

Integrity Constraints + Database Design

Hakim Qahtan

Part of the slides were prepared by

Yannis Velegrakis

Department of Information and Computing Sciences

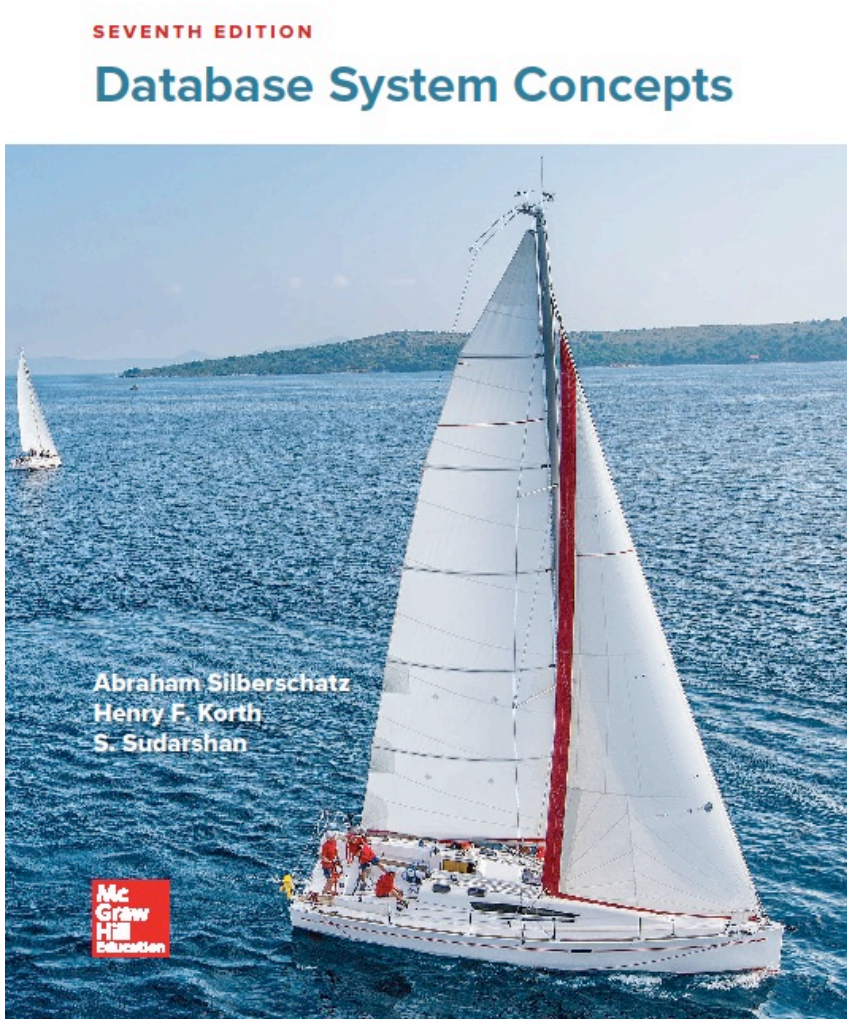
Utrecht University

Utrecht University

Reading Material for Today

Database System Concepts (7th Edition)

CH 3.2.1, 4.4, 6.1, 6.2



Utrecht University

Integrity Constraints

Integrity Constraints

- A constraint is a relationship among data elements that the DBMS is required to enforce
- Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.
- Examples:
  - Checking that an account must have a balance greater than \$1.00
  - A salary of a bank employee must be at least \$4.00 an hour
  - A customer must have a (non-null) phone number

Integrity Constraints (ICs)

- IC: condition that must be true for *any* instance of the database; e.g., *domain constraints.*
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.
- A *legal* instance of a relation is one that satisfies all specified ICs.
  - DBMS should not allow illegal instances.
- If the DBMS checks ICs, stored data is more faithful to real-world meaning.
  - Avoids data entry errors, too!

Where do ICs Come From?

- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations.

- We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance.
  - An IC is a statement about *all possible* instances!
  - From example, we know *name* is not a key, but the assertion that *id* is a key is given to us.

- Key and foreign key ICs are the most common; more general ICs supported too.

Kinds of Integrity Constraints

- Value-based
- Primary key
- Foreign-key, or referential-integrity
- Tuple-based

Value-Based Constraints – Data Types

- Specify the type of the data that can be entered in a specific field

```
CREATE TABLE test (
  id INTEGER PRIMARY KEY,
  full_name VARCHAR(30),        -- allocates a space for up to 30 characters
  dept_code CHAR(3),            -- allocates a space for exactly 3 characters
  dept_name VARCHAR(100)        -- allocates up to 100 characters
);
```

Test the database with the following queries

```
INSERT INTO test
VALUES ('kk', 'JH', 'CS', 'Computer Science')
```

```
INSERT INTO test
VALUES (123, 'JH', 'CS', 'Computer Science')
```

Basic Data Types in SQL

- CHAR(n) allocates a space for exactly X characters
- VARCHAR(n) allocates a space for up to X characters
- INT or INTEGER stores integer values
- SMALLINT stores integer values
- NUMERIC(p,d) numerical values with total p digits (plus a sign)
  - Out of the p digits, d are after the decimal point
- REAL stores floating point numbers with double precision
- FLOAT(n) stores floating point numbers with double precision

Basic Data Types in SQL

- CHAR(n) and VARCHAR(n)
  - CHAR(n) allocates a space for exactly `n’ characters – if the value is shorter, the remaining spaces are filled with the `space’ character
  - VARCHAR(n) allocates a space for up to `n’ characters – allocation is done during the runtime.

Value-Based Constraints – Data Types (Cont.)

- Most DBMSs use dynamic typing
  - Data of any type can (usually) be inserted into any column
  - You can put arbitrary length strings into integer columns, floating point numbers in Boolean columns, or dates in character columns
- Columns of type INTEGER/NUMERIC PRIMARY KEY cannot accept string
  - Error message will be printed if you try to put string into an INTEGER PRIMARY KEY column
  - If you put floating point value, some DBMSs store the integer part

Test the database with the following query

```
INSERT INTO test

VALUES (3.14, 'JH', 'CS', ‘Computer Science')
```

Primary Key Constraints

- A set of fields is a key for a relation if :
  1. No two distinct tuples can have same values in all key fields, and
  2. This is not true for any subset of the key.
  - Part 2 false? A superkey.
  - If there's >1 key for a relation, one of the keys is chosen (by DBA) to be the primary key.
- E.g., sid is a key for Students. (What about name?) The set {sid, gpa} is a superkey.

Single attribute key

- Use the PRIMARY KEY key or UNIQUE after the type in the declaration of the attribute
- Example:

```
CREATE TABLE test (
                sid     INTEGER UNIQUE,
                name    VARCHAR (30),
                major VARCHAR (30)
        );
```

- You may also use sid INTEGER PRIMARY KEY

Multiattribute key

- You can also specify multiple attributes to be PRIMARY KEY
- product_name and country of origin are the key for the sells relation
- Example:

```
CREATE TABLE sells (
                product_name CHAR(20),
                price REAL,
                country_of_origin VARCHAR (30),
                PRIMARY KEY (product_name, country_of_origin)
        );
```

Primary and Candidate Keys in SQL – What Could Go Wrong?

- Possibly many candidate keys (specified using UNIQUE), one of which is chosen as the *primary key*.
  - "For a given student and course, there is a single grade." vs. "Students can take only one course and receive a single grade for that course; further, no two students in a course receive the same grade."
  - Used carelessly, an IC can prevent the storage of database instances that arise in practice!

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid, cid) )

CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid),
   UNIQUE (cid, grade) )
```

Foreign Keys, Referential Integrity

- *Foreign key*: Set of fields in one relation that is used to `refer' to a tuple in another relation. (Must correspond to primary key of the second relation.) Like a `logical pointer'.
  - Let A be a set of attributes (could be one attribute).
  - Let R and S be two relations that contain attributes A
  - A is the primary key of S.
  - A is said to be a **foreign key** in R if for any r in R, r(A) in S.
- E.g. *sid* is a foreign key referring to Students:
  - Enrolled(*sid*: string, *cid*: string, *grade*: string)
  - If all foreign key constraints are enforced, *referential integrity* is achieved, i.e., no dangling references.

Foreign Keys in SQL

- Only students listed in the Students relation should be allowed to enroll for courses.

```
CREATE TABLE Enrolled
  (esid CHAR(20),  cid CHAR(20),  grade CHAR(2),
   PRIMARY KEY  (esid,cid),
   FOREIGN KEY (esid) REFERENCES Students )
```

Referenced attributes must be PRIMARY KEY or UNIQUE in the original table (relation)

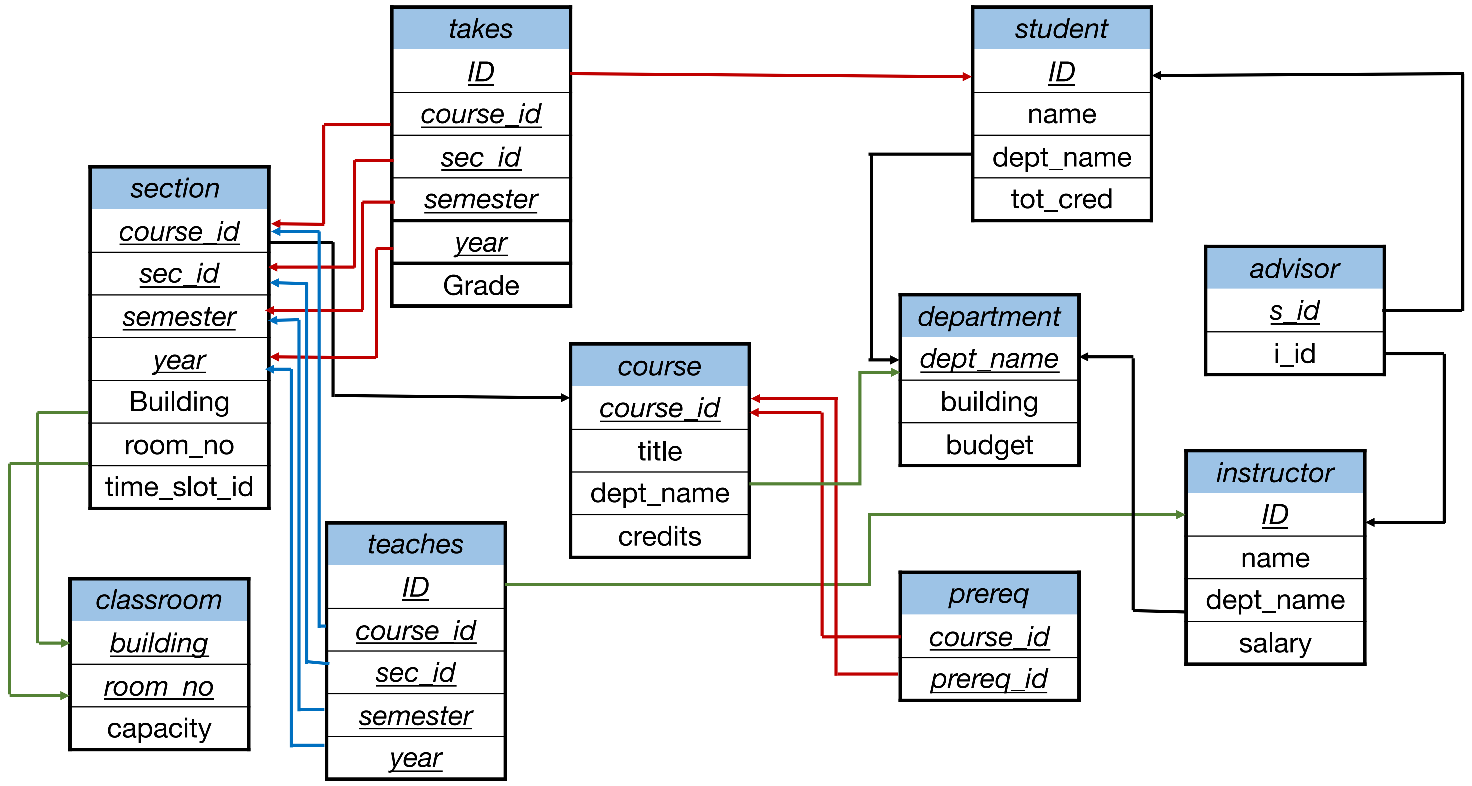
Enrolled

esid	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Foreign Keys – The University Database Example



dept_name is a foreign key in each of the course, student and instructor relations

Enforcing Referential Integrity

- Consider Students and Enrolled; *sid* in Enrolled is a foreign key that references Students.
- What should be done if an Enrolled tuple with a non-existent student id is inserted? (*Reject it!*)
- What should be done if a Students tuple is deleted?
- Similar if primary key of Students tuple is updated.

Referential Integrity in SQL

- SQL/92 and SQL:1999 support all 4 options on deletes and updates.
  - Default is NO ACTION (*delete/update is rejected*)
  - CASCADE (Make the same changes to all tuples that refer to the updated/deleted tuple)
  - SET NULL / SET DEFAULT (sets foreign key value of referencing tuple to NULL or a default value)

```
CREATE TABLE Enrolled
  (sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid,cid),
   FOREIGN KEY (sid)
    REFERENCES Students
          ON DELETE CASCADE
          ON UPDATE SET DEFAULT )
```

Foreign Keys Violations – Actions to Consider – Cascade

- Delete the Mathematics department from department
  - Then delete all tuples from course that have dept_name = ‘Mathematics’
- Update the Mathematics tuple by changing the ‘Mathematics’ to ‘Math’
  - Then change all records in course with dept_name = ‘Mathematics’ to dept_name = ‘Math’
- Example:

```
UPDATE department
SET dept_name = 'Math'
WHERE dept_name = 'Mathematics';
```

```
UPDATE course
SET dept_name = 'Math'
WHERE dept_name = ‘Mathematics';
```

Foreign Keys Violations – Actions to Consider – Set NULL

- Delete the Mathematics tuple from department:
  - Change all tuples of course that have dept_name = ’Mathematics’ to have dept_name = NULL.
- Update the Mathematics tuple by changing Mathematics’ to ‘Math’:
  - Same change as for deletion.
- Example:

```
DELETE FROM department
WHERE dept_name = 'Mathematics';
```

```
UPDATE department
SET dept_name = 'Math'
WHERE dept_name = 'Mathematics';
```

```
UPDATE course
SET dept_name = NULL
WHERE dept_name = ‘Mathematics';
```

The check clause

- Defines constraints on the values of a particular attribute.
- Syntax: CHECK(<condition>)
- The condition may use the name of the attribute, but any other relation or attribute name must be in a subquery.

The check clause

- Example: ensure that semester is one of Fall, Winter, Spring or Summer and year is greater than 1990:

```
CREATE TABLE section (
   course_id VARCHAR (8),
   sec_id VARCHAR (8) NOT NULL,
   semester VARCHAR (6) CHECK (semester IN ('Fall', 'Winter', 'Spring',
        'Summer')),
   year NUMERIC (4,0) CHECK (year > 1990),
   building VARCHAR (15),
   room_number VARCHAR (7),
   time_slot_id VARCHAR (4),
   PRIMARY KEY (course_id, sec_id, semester, year)
);
```

Tuple-Based Check

- CHECK (<condition>) may be added as a relation-schema element.
- The condition may refer to any attribute of the relation.
  - But other attributes or relations require a subquery.
- Checked on insert or update only.

```
CREATE TABLE section (
   course_id VARCHAR (8),
   sec_id VARCHAR (8) NOT NULL,
   semester VARCHAR (6),
   year NUMERIC (4,0),
   building VARCHAR (15),
   room_number VARCHAR (7),
   time_slot_id VARCHAR (4),
   PRIMARY KEY (course_id, sec_id, semester, year),
   CHECK (semester IN ('Fall', 'Winter', 'Spring', 'Summer') AND (year > 1990))
);
```

Utrecht University

Timing of Checks

- Attribute-based checks are performed only when a value for that attribute is inserted or updated.
  - Example: CHECK (year >= 1990) checks every new year and rejects the modification (for that tuple) if the year is before 1990

Complex Check Clauses

- CHECK (*time_slot_id* IN (SELECT *time_slot_id* FROM *time_slot*))
  - why not use a foreign key here?
- Every section has at least one instructor teaching the section.
  - how to write this?
- Unfortunately: subquery in check clause not supported by a lot of database management systems

NOT NULL

- **NOT NULL**
  - Declare *name* and *budget* to be NOT NULL

    name VARCHAR(20) NOT NULL
    budget NUMERIC(12,2) NOT NULL

Logical Database Design

Utrecht University

DB Design

- A mechanism (methodology) for ensuring that each relation in the database is good.
  - Entity-Relationship model
    - Models an organization (enterprise) as a set of entities and relationships
    - Represented using ER-diagram
  - Normalization theory
    - Formalize what is a bad design

Entities

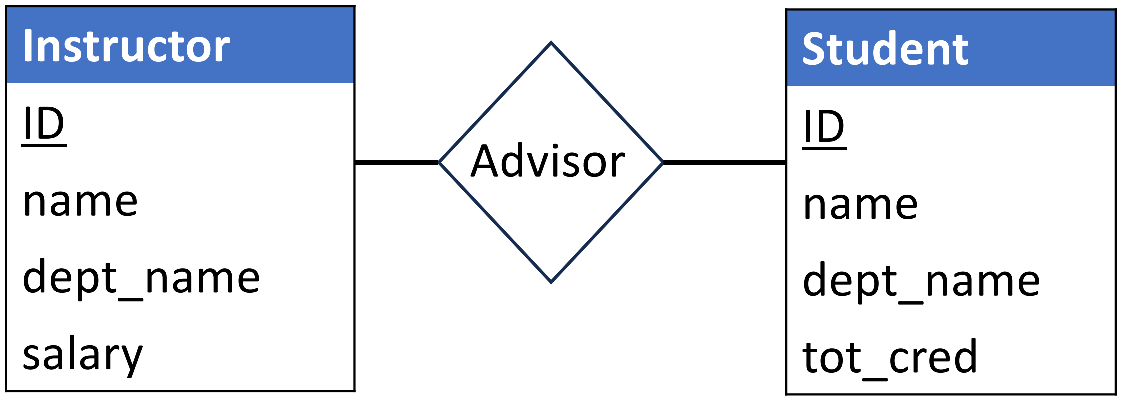
- An entity: an existing “object” that is distinguishable from other objects
  - Example: Specific person, product, event, plant
- Entity set: a set of entities that share the same properties
  - Set of students, plants, activities, trees
- An entity is represented by a set of attributes
  - students = (ID, name, dept_name, tot_cred).
- Each entity should differ from the others in at least a value of one attribute

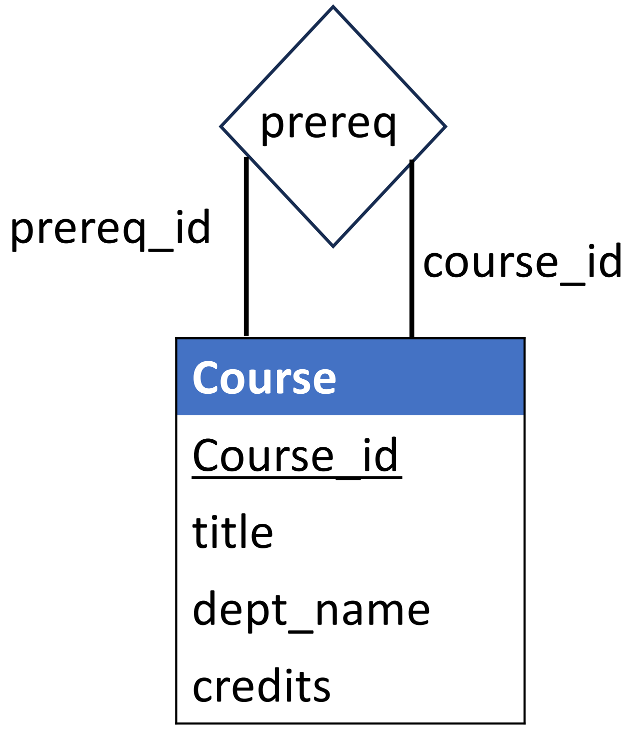
Relationships

- A relationship: an association between several entities
  - Robin is an Advisor of Mark
  - Tom works in the ICS department
  - Fred is enrolled in INFOMDWR
- A relationship set is a set of relationships of the same type
  - Advisor = {(Robin, Advisor of, Mark), (Katz, advisor of, Shankar), (Lola, advisor of, Tim)}

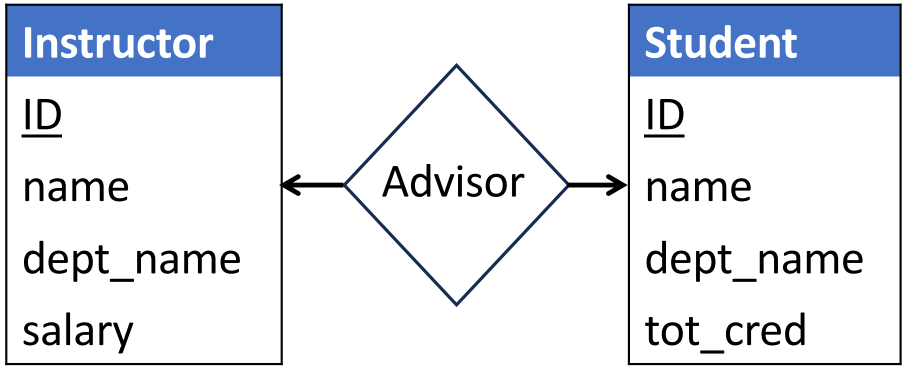
ER-Diagram

- An entity is represented as a **rectangle**
- A relationship is represented as a **diamond**



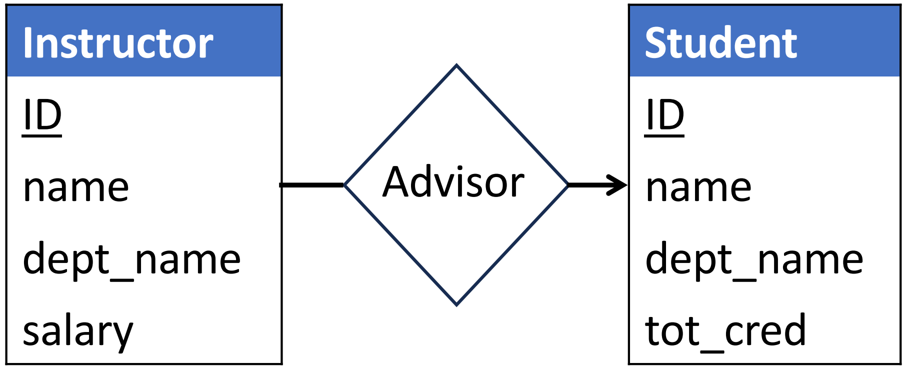


Relationship Cardinality



Arrows represent 1-1 relationship

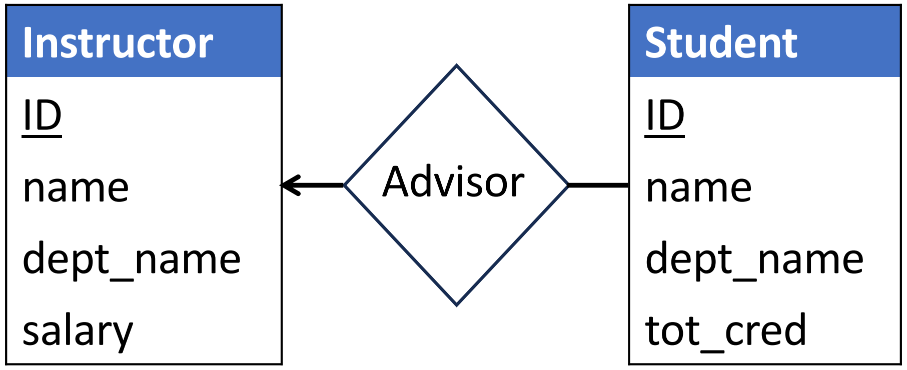
- A student can have at most one advisor
- An instructor can supervise at most one student

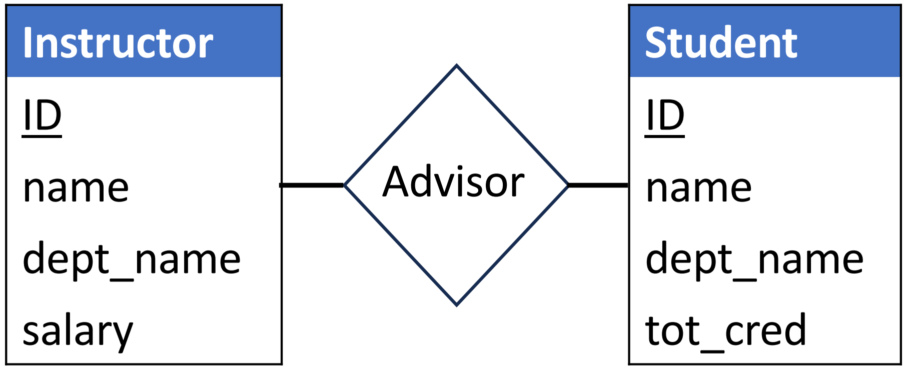


Line-Arrow represent many-1 relationship

- A student can have zero or more advisors
- An instructor can supervise at most one student

Relationship Cardinality





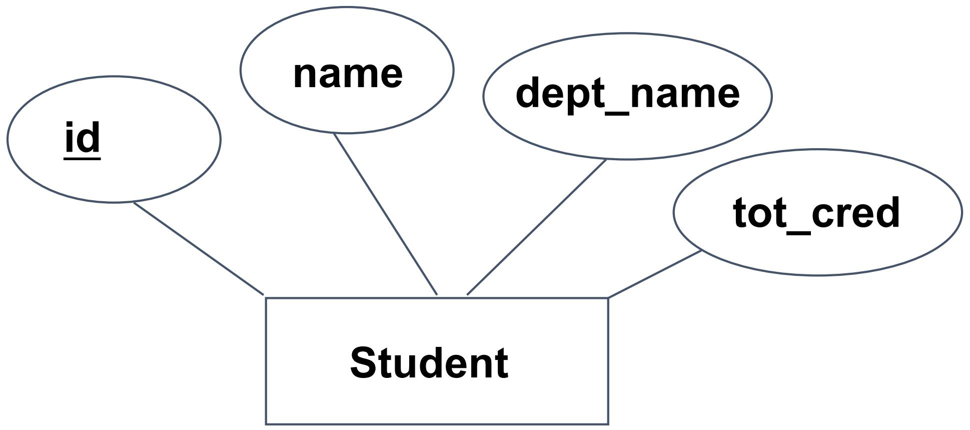
Arrow-Line represent 1-many relationship

- A student can have at most one advisor
- An instructor can supervise zero or more students

Lines represent many-many relationship

- A student can have zero or more advisors
- An instructor can supervise zero or more students

Logical DB Design: Entities as Tables



```
CREATE TABLE Student
   (id CHAR(20),
   name CHAR(20),
   dept_name CHAR(20),
   tot_cred INTEGER,
   PRIMARY KEY  (id)
 );
```

Logical DB Design: Relationships as Tables

- A many-many relationship between two entities is expressed as a stand-alone table that is linked to the entities through foreign keys.
- That table has a Foreign Key referencing each PK in the tables of the entities to which it is linked, plus some additional descriptive attributes.
- The Primary Key of the table is the union of the Primary Keys in the participating entities

```
CREATE TABLE takes(
  id  CHAR(20),
  course_id  VARCHAR(7),
  sec_id VARCHAR(8),
  semester VARCHAR(6),
  year numeric(4,0),
  Grade numeric(2,1),
  PRIMARY KEY (id, course_id, sec_id,
      semester, year),
  FOREIGN KEY (id) REFERENCES student,
  FOREIGN KEY (course_id, sec_id, semester,
      year) REFERENCES section
);
```

Examples of Relationships as Tables

- Manage relationship

```
CREATE TABLE  Manages(
  i_id  CHAR(11),
  dept_name CHAR(20),
  since  DATE,
 d_id CHAR (11) PRIMARY KEY,
  FOREIGN KEY (i_id) REFERENCES instructor,
  FOREIGN KEY (d_id) REFERENCES department)
```

- Since each department has a unique manager, we could instead combine Manages and Departments.

```
CREATE TABLE  Dept_Mgr(
  d_id CHAR(11),
  dept_name  CHAR(20),
  budget  REAL,
  i_id  CHAR(11),
  since  DATE,
  PRIMARY KEY  (did),
  FOREIGN KEY (i_id) REFERENCES instructor)
```

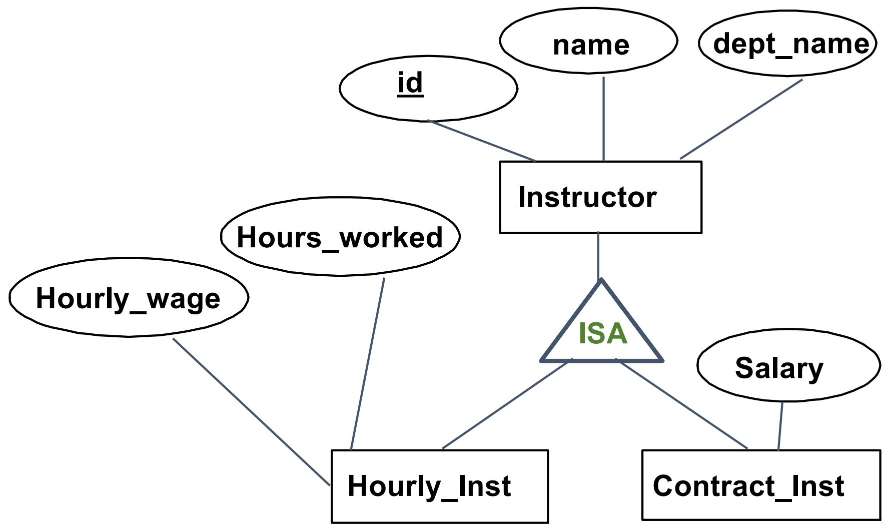
Utrecht University

Primary Keys for Relationships in the Relational Models

- For 1-1 relationship, one of the primary keys of the participating entities can be considered as the primary key of the relationship
- For 1-many and many-1 relationships, the primary of many entity is considered as the primary key of the relationship
- For many-many, the primary key of the relationship is the union of the attributes in the primary keys of the participating entities

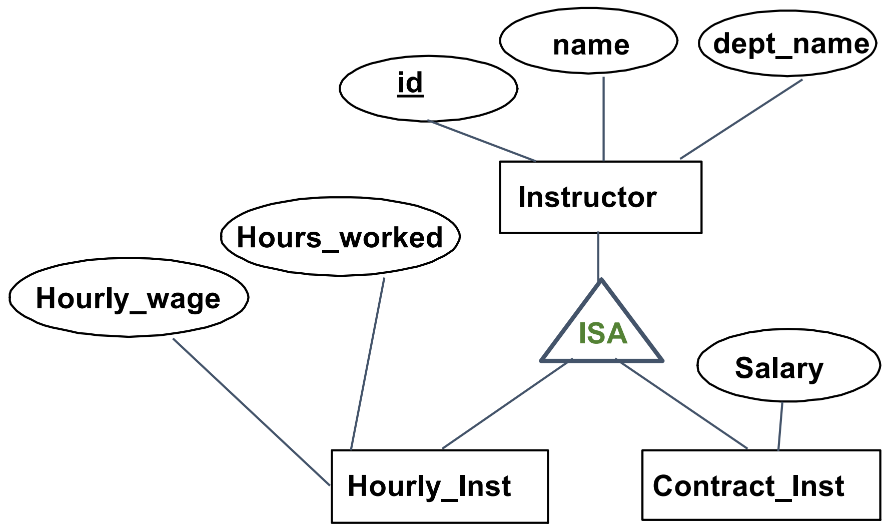
What are the options to define these kinds of relationships?

The ISA Hierarchy example



- **Entity hierarchies:** called class hierarchies organize a group of entity sets into a parent/child hierarchy using as criterion their generality/specificity
- **Overlap Constraints:** specify that the children of an entity do/don’t overlap e.g. Hourly_Inst/Contract_Inst don’t overlap whereas cartoons and drama overlap in movies database

The ISA Hierarchy example



- **Covering Constraints:** instances of the children of an entity include all instances of their parent (i.e., cover it).
  - Hourly_Inst and Contract_Inst cover all instances from the super class Instructor

Utrecht University

Translating ISA Hierarchies to Relations

- ***General approach:***
  - 3 relations: Instructor, Hourly_Inst and Contract_inst.
    - *Hourly_Inst*: Every employee is recorded in Instructor.
    - For hourly instructors, extra info recorded in Hourly_Inst (*hourly_wages*, *hours_worked*, *id*); must delete Hourly_Inst tuple if referenced Instructor tuple is deleted).
    - Queries involving all instructors easy, those involving just Hourly_Inst require a join to get some attributes.

Translating ISA Hierarchies to Relations

- Alternative: Just Hourly_Inst and Contract_Inst.
  - *Hourly_Inst*: *id, name, dept_name, hourly_wages, hours_worked.*
  - *Contract_Inst*: *id, name, dept_name, salary.*
  - Each instructor must be in one of these two subclasses (Coverage).

Views

- A *view* is just a relation, but we store a *definition*, rather than a set of tuples.

```
CREATE VIEW ActiveStudents (name, grade)
        AS SELECT S.name, T.grade
        FROM Students S, takes T
        WHERE S.id = T.id and S.tot_cred < 31
```

- Views can be dropped using the DROP VIEW command.
  - How to handle DROP TABLE if there's a view on the table?
    - DROP TABLE command has options to let the user specify this.

Utrecht University

Views and Security

- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
  - Given `ActiveStudents', but not `student' or `takes', we can find students who are enrolled, but not the *course_id's* of the courses they are taking.

Relational Model: Summary

- A tabular representation of data.
- Simple and intuitive, currently the most widely used.
- Integrity constraints can be specified by the DBA, based on application semantics. DBMS checks for violations.
    - Two important ICs: primary and foreign keys
    - In addition, we *always* have domain constraints.

Thank You

- Summarize what you learned today in 2-minutes

Utrecht University