Data Wrangling and Data Analysis
Integrity Constraints, Functional Dependencies and Denial Constraints

Hakim Qahtan

Department of Information and Computing Sciences

Utrecht University

Topics for Today

- Integrity Constraints
- Functional Dependencies
- Denial Constraints

Integrity Constraints

Integrity Constraints

- A constraint is a relationship among data elements that the DBMS is required to enforce

can be binary, multiple or one

- 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

e.g. customers bank account should be deleted but cannot be deleted without it being empty

Kinds of Integrity Constraints

- Primary key — enforces that all values in set of attributes should be unique
- Foreign-key, or referential-integrity — values have to appear in another table
- Value-based — like non-null or only a certain value
- Tuple-based — enforce integrity constraints in more than a single value

Single attribute key

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

each value should be unique in this data base

- Example:

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

when defining attribute - it has to be unique or PRIMARY KEY

UNIQUE doesn't have to be a primary key but referencing it is easier with Primary key

force specific field to have certain values

- You may also use student_id INTEGER PRIMARY KEY

Multiattribute key

not just one attribute is key

- You can also specify multiple attributes to be PRIMARY KEY prevents NULL VALUES
- 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)
    );
```

list the set of attributes that we want to use as primary key

keys for relationships r=are more likely to have to be multiple attributes like the enrolled table
students & courses - foreign keys of both attributes together can be primary key

Foreign Keys – Referential Integrity

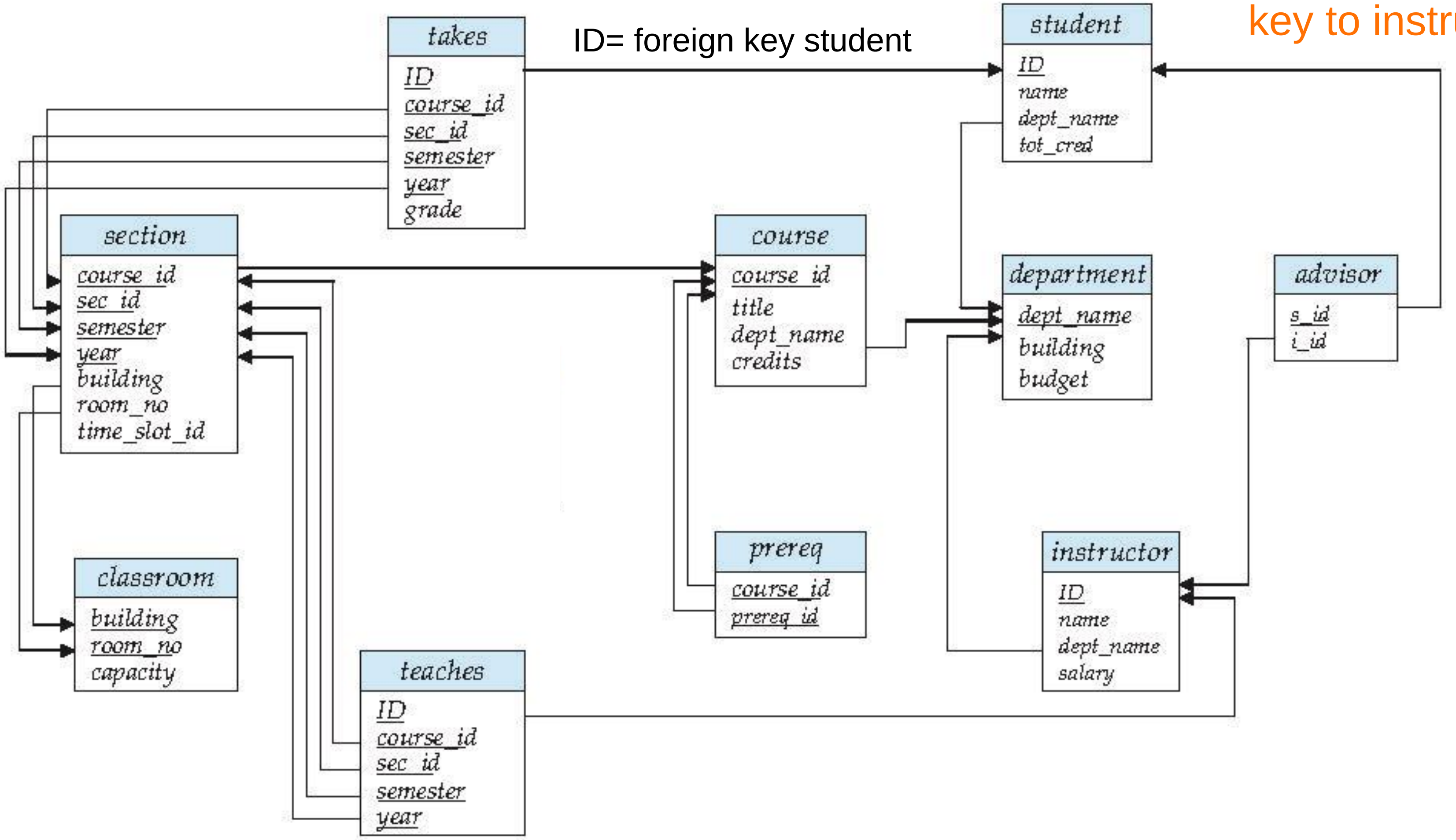
attribute in a relation that refers to a primary key in another relation

- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.
  - Example: If "Biology" is a department name appearing in one of the tuples in the *instructor* relation, then there exists a tuple in the *department* relation for "Biology".
- FOREIGN KEY = primary key in another relation
  - Let A be a set of attributes.
  - Let R and S be two relations that contain attributes A and A is the primary key of S.
    table
  - A is said to be a **foreign key** in R if for any values of A appearing in R these values also appear in S.

all the values in the set of attributes should also appear in the set of attributes of another relation

Foreign Keys – Referential Integrity (Cont.)

teaches ID is a primary key in teaches and is a foreign key to instructor



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

Foreign Keys – Referential Integrity (Cont.)

- Expressing foreign key – Use keyword REFERENCES
  - After the attribute

```
CREATE TABLE department (
        dept_name       VARCHAR(30) PRIMARY KEY,
        buildingVARCHAR(30) ,
        budget REAL
);

CREATE TABLE course (
        course_id       VARCHAR(8),
        title VARCHAR(20),
        dept_name VARCHAR(30) REFERENCES department(dep_name),
        credits  REAL
);
```

what table it is referencing to (not just the table, but also the column)

Foreign Keys – Referential Integrity (Cont.)

- Expressing foreign key – Use keyword REFERENCES
  - As a schema element

```
CREATE TABLE department (
   dept_name       VARCHAR(30) PRIMARY KEY,
   buildingVARCHAR(30) ,
   budget REAL
);

CREATE TABLE course (
   course_id          VARCHAR(8),
   title VARCHAR(20),
   dept_name VARCHAR(30),
   credits   REAL,
   FOREIGN KEY (dept_name) REFERENCES department(dep_name)
);
```

Referenced attributes must be declared PRIMARY KEY or UNIQUE

we can reference to unique attributes as well

separate line because new schema element; department will be referencing the department name
one can use different names for 'foreign key' like department instead of dept_name

Foreign Keys – Possible Violations

- R contains a foreign key from S then two violations are possible:
  - An insert update to R introduces values not in S

    INSERT INTO course

    VALUES ('Math-101', 'Calculus', 'Mathematics', 7.5)

    Where Mathematics is not in the department relation

  - A deletion or update to S causes some tuples of R to "dangle"

    DELETE FROM *department* WHERE *dept_name = 'Biology'*;

    Records in the course and instructor tables with 'Biology' in their dept_name will be affected

trying to insert a new record that has a foreign key that doesn't exist -> error can't insert

Foreign Keys Violations – Actions to Consider

- Let R = course and S = department
- An insert or update to course that introduces a nonexistent department must be rejected.
- A deletion or update to department that removes a dept_name value found in some tuples of course can be handled in three ways
  - Default : Reject the modification. reject the query - won't be deleted -
  - Cascade : Make the same changes in course.
    - Deleted dept_name: delete course tuple.
    - Updated dept_name : change value in course.

    apply the same change that happened in one table to all the other tables as well e.g. change Mathematics to Math everywhere else as well
  - Set NULL : Change the dept_name to NULL in the course tuples.

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 that you have to do yourself

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

query will be issued by the DBMS and will execute it by itself

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:

issue this query

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

or this

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

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

DBMS

Foreign Keys Violations – Choosing a Policy

- When we declare a foreign key, we may choose policies SET NULL or CASCADE independently for deletions and updates.
- Follow the foreign-key declaration by:
- ON [UPDATE, DELETE][SET NULL CASCADE]
- Two such clauses may be used.
- Otherwise, the default (reject) is used.

specify what you want to happen
e.g. on update do ... (set null/cascade/...)

if you don't specify -> reject as default policy

Foreign Keys Violations – Choosing a Policy (Cont.)

```
CREATE TABLE course (
  course_id     VARCHAR(8),
  title VARCHAR(20),
  dept_name VARCHAR(30),
  credits         REAL,
  FOREIGN KEY (dept_name) REFERENCES department(dep_name)
  ON DELETE SET NULL
  ON UPDATE CASCADE
);
```

what's the policy?

on delete set the values to null and when updated cascade - change the value to the same new value as course

enrolled sid and cid are foreign keys - one can separate them

NOT NULL

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

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

12= length of number
2 = decimals

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),          -- up to 30 characters
   dept_code CHAR(3),              -- exactly 8 characters
   dept_name VARCHAR(100)          -- up to 100 characters
);
```

also considers UP TO that character

Test the database with the following query

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

what would happen if I put integer value in it
kk is not accepted because it is a string and not an integer
- won't be executed

Value-Based Constraints – Data Types (Cont.)

- Most DBMSs use dynamic typing

located dynamically - DBS finds out the best types for that value and converts it and then accepts the value

  - 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

but if you try to insert a floating point it will take out the integer part of it an use it for the db

Test the database with the following query

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

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.

so if I want to refer to another relation I have to do that in a subquery

HELPS WITH ASSIGNMENT

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)
);
```

when updating - check if semester is one of those strings

no decimal places allowed and it has to be bigger than 1990 but some programs still accept it

Tuple-Based Check

two checks in one line as a schema element

- 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))
);
```

Timing of Checks

when updating the db

- 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

like subquery - I don't want a time slot ID that doesn't exist in the other table
difference to FOREIGN KEY:

- 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 pretty much any database

not all of them are supported by all data management systems

Functional Dependency (FD)

Functional Dependence (FD)

if two attributes or two sets of attributes - one value in first column should be connected with one single value in other table

- Functional dependence (FD): the values of a set of attributes X determine the values of another set of attributes Y
  - Denoted by $X \longrightarrow Y$
  - If two records has the same set of values for the attributes in X the they should have the same set of values for the attributes in Y
  - In the instructor relation, dept_name is functionally dependent on name ($name \longrightarrow dept\_name$)
  - Given the instructor name, I can find *one and only one value* of dept_name
- Constraints on the set of legal relation instances
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes

only a single record for the department in this case
the name can only determine ONE department

Functional Dependence

R = (A, B, C, D, E)
X = A,B
Y = C, D

- Let R be a relation with attributes (A,B, C, D, E)

$$X \subseteq R, \ Y \subseteq R$$

- The functional dependency

$$X \longrightarrow Y$$

holds on R if and only if whenever two tuples t_1, t_2 of R agree on the attributes of X, they also agree on the attributes of Y. That is

$$t_1[X] = t_2[X] \implies t_1[Y] = t_2[Y]$$

- Examples:
  - The capital determines the country
  - The country determines the Internet domain

when Netherlands it should be NL

ID	name	dept_name	salary
22322	Einstein	Physics	95000
33452	Gold	Physics	87000
21212	Wu	Finance	90000
10101	Brandt	Comp. Sci.	82000
43521	Katz	Comp. Sci.	75000
98531	Kim	Biology	78000
58763	Crick	Elec. Eng.	80000
52187	Mozart	History	65000
32343	El Said	History	86000

$$name \longrightarrow dept\_name$$

department name is dependent on name but not the other way around

Alternative Definition of the Keys

- K is a superkey for relation R if and only if $\mathrm{K} \longrightarrow R$
  - This is the *uniqueness* property of "key"
- K is a candidate key for R if and only if
  - $\mathrm{K} \longmapsto R$, and

no duplicates because it might have different values for other attributes

  - there is no $X \subset K, X \nrightarrow R$
    - make sure key has minimum set of attributes (*minimality*)

no subset that can determine all the attributes - smallest possible

- Question
  - When this definition will not hold?

could it happen that if we have duplicate records that there would be duplicates for all the records in R????

e.g. city determines the country but country can't determine the city

Functional Dependencies

- Functional dependencies allow us to express constraints that cannot be expressed using superkeys.
- Example: Consider the department relation:

  We expect the following set of functional dependencies to hold:

  id $\rightarrow$ name id determines the nam - left side is determinate and right is Dependant

  id $\rightarrow$ dept_name for each value of id there's a single department name

  name, dept_name $\rightarrow$ salary name and department will determine the salary

  but would not expect the following to hold:

  salary $\rightarrow$ name

Closure of a Set of Functional Dependencies

- Given a set of functional dependencies $\mathcal{F}$, there are certain other functional dependencies that are logically implied by $\mathcal{F}$.
- The set of all functional dependencies *logically implied* by $\mathcal{F}$ is the closure of $\mathcal{F}$.

set of functional dependencies

- We denote the closure of $\mathcal{F}$ by $\mathcal{F}^+$. logical dependencies that can be found
- We can find all of $\mathcal{F}^+$ by applying Armstrong's Axioms:
  - if $X \subseteq Y$, then $Y \longrightarrow X$ (*reflexivity*) each item is related to itself if X is a subset of Y then Y determines X
  - if $\mathrm{X} \longrightarrow Y$, then $\mathrm{AX} \longrightarrow AY$ (*augmentation*) if I add same value - dependencies it holds
  - if $X \longrightarrow Y$ and $Y \longrightarrow W$, then $X \longrightarrow W$ (*transitivity*)

  duplicates in X are duplicates in Y

  these rules are sound and complete. A is a set of attributes (could be single attribute)

Trivial Functional Dependencies

- **Trivial** – FDs that can be derived using Armstrong's Axioms are called trivial – trivial FDs always hold. Examples:
  - If Y is a subset of X, then the FD $\mathrm{X} \longrightarrow Y$ is called a trivial FD.
  - If X is a key candidate then $\mathrm{X} \longrightarrow Y, \forall Y$ will always determine Y
- What about the right hand side (RHS) of the dependency?
  - $\mathrm{X} \longrightarrow Y \;\Longrightarrow X \longrightarrow B \qquad \forall B \in Y$ X determines every attribute in B
    - We can restrict the RHS to have only a single attribute

right hand side -->

I have a primary key - will it determine attribute B? Yes because primary key can have no duplicate

Examples of Armstrong's Axioms

COLUMNS DETERMINE OTHER COLUMNS

- if $X \subseteq Y$, then $Y \longrightarrow X$ (*reflexivity*)

  name → name single attribute
  name, dept_name → name
  name, dept_name → dept_name

- if $\mathrm{X} \longrightarrow Y$, then $\mathrm{AX} \longrightarrow AY$ (*augmentation*)

  name → dept_name
  name, salary → dept_name , salary

- if $X \longrightarrow Y$ and $Y \longrightarrow W$, then $X \longrightarrow W$ (*transitivity*)

  id → name

  name → dept_name implies
  id → dept_name

  id= primary key

More Derived FDs

$X \longrightarrow Y$ and $X \longrightarrow W$ then $X \longrightarrow YW$

$X \longrightarrow YW$ then $X \longrightarrow Y$ and $X \longrightarrow W$ we saw this earlier X determines all attributes of Y

$X \longrightarrow Y$ and $WY \longrightarrow Z$ then $XW \longrightarrow Z$

- Can we prove the correctness of $X \longrightarrow Y$ and $WY \longrightarrow Z$ then $XW \longrightarrow Z$?

  $X \longrightarrow Y$ and $WY \longrightarrow Z$ (given) yes

  $X \longrightarrow Y$ then $XW \longrightarrow YW$

  $XW \longrightarrow YW$ and $YW \longrightarrow Z$ then $XW \longrightarrow Z$

- Exercise: can you prove if $X \longrightarrow Y$ then $XW \longrightarrow Y$?

  X determines Y then XW will determine XY

  if a set of attributes determines another set of attribute then it will determine every subset from the RHS

Use of FDs

definitions
satisfies
holds on

- We use functional dependencies to:
  - Test relations to see if they are legal under a given set of functional dependencies.
    - If a relation r is legal under a set $\mathcal{F}$ of functional dependencies, we say that r **satisfies** $\mathcal{F}$.
  - Specify constraints on the set of legal relations
    - We say that $\mathcal{F}$ **holds on** R if all legal relations on R satisfy the set of functional dependencies $\mathcal{F}$.
- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.
  - For example, a specific instance of *instructor* may, by chance, satisfy

$$name \rightarrow id$$

Use of FDs (Cont.)

- We use functional dependencies to:
  - Detect inconsistencies in the data
  - For example, if we are given that the instructor name in a university can determine the department i.e. in the department relation we have:

    $$name \rightarrow dept\_name$$

    Then the highlighted records violate this FD

    When discovering an FD violation,

    each value can be considered as the

    source of violation

id	name	dept_name	salary
22322	Einstein	Physics	95000
33452	Gold	Physics	87000
21212	Wu	Finance	90000
10101	Einstein	Comp. Sci.	82000
43521	Katz	Comp. Sci.	75000
98531	Kim	Biology	78000
58763	Crick	Elec. Eng.	80000

inconsistent = valuating functional dependancy

- Exercise: If you know that in the goal relation

  $$player \rightarrow teamid$$

  Write python script to check for violations

Conditional Functional Dependencies (CFDs)

in some cases FD doens't hold on to all records in the relation

- In the UK, zip code uniquely determines the street
- The constraint may not hold for other countries
- This constraints can be expressed as follows

  $([country = 44, zip] \rightarrow street)$
- It expresses a fundamental part of the semantics of the data
- It can NOT be expressed as an FD
  - It does not hold on the entire relation; instead, it holds on tuples representing UK customers only

country	area-code	phone	street	city	zip
44	131	1234567	Mayfield	Liverpool	EH4 8LE
44	131	3456789	Crichton	Manchester	EH4 8LE
01	908	3456789	Mountain Ave	NYC	07974

one of them is wrong because two streets for same zip

zip determines street. so if country equals 44 then zip code will determine the streets because in GB the zip determines the street

Denial Constraints

Denial Constraints (DCs)

- A denial constraint (DC) expresses that a set of predicates cannot be true together for any combination of tuples in a relation

"all of these predicates can't be true together"

$$\forall t_i, t_j, \ldots \in R : \neg(p_1 \wedge \cdots p_m)$$

- Each predicate expresses a relationship between two cells, or between a cell and a constant
- Example:
  - if two employees are working in the same state, then the tax should be proportional to the income – we express that like:

$$\forall t_i, t_j \in R : \neg((t_i.state = t_j.state)$$
$$\wedge (t_i.income > t_j.income)$$
$$\wedge (t_i.taxRate < t_j.taxRate))$$

state attribute

employee number i with more income than employee number J ... can't be true together rewatch mt

Denial Constraints (DCs) – Examples

- Expressing functional dependency as a DC – consider *zip* → *city*

$$\forall t_i, t_j \in R :\ \neg((t_i.zip = t_j.zip) \wedge (t_i.city \neq t_j.city))$$

- Order dependency

$$\forall t_i, t_j \in R :\ \neg((t_i.date \leq t_j.date) \wedge (t_i.population > t_j.population)$$

means that they can't be true at the same time

$$\forall t_i \in R :\ \neg((t_i.openingTime \leq t_i.closingTime)$$

- Uniqueness constraint

$$\forall t_i, t_j \in R :\ \neg(t_i.id = t_j.id)$$

for every tubple ti and tj it cannot be that two values (specific attribute) are equal

Demo

- Examples of ICs, FDs, DCs
- Applying constraints on the data
- Given a constraint, check if the data satisfy that constraint or not
- Working on tools for FDs discovery such as Fdep
- Working on Nadeef (a data cleaning tool that allow for creating and applying set of rules or constraints).

Further Reading Material

- Section 4.4 of the Database System Concepts Book
- Section 8.3 of the Database System Concepts Book
- Bleifuß, Tobias, Sebastian Kruse, and Felix Naumann. Efficient Denial Constraint Discovery with Hydra. Proceedings of the VLDB Endowment (PVLDB). 11(3):311-323, 2017.