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
- 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

Kinds of Integrity Constraints

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

Single attribute key

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

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

- You may also use student_id INTEGER PRIMARY KEY

Multiattribute key

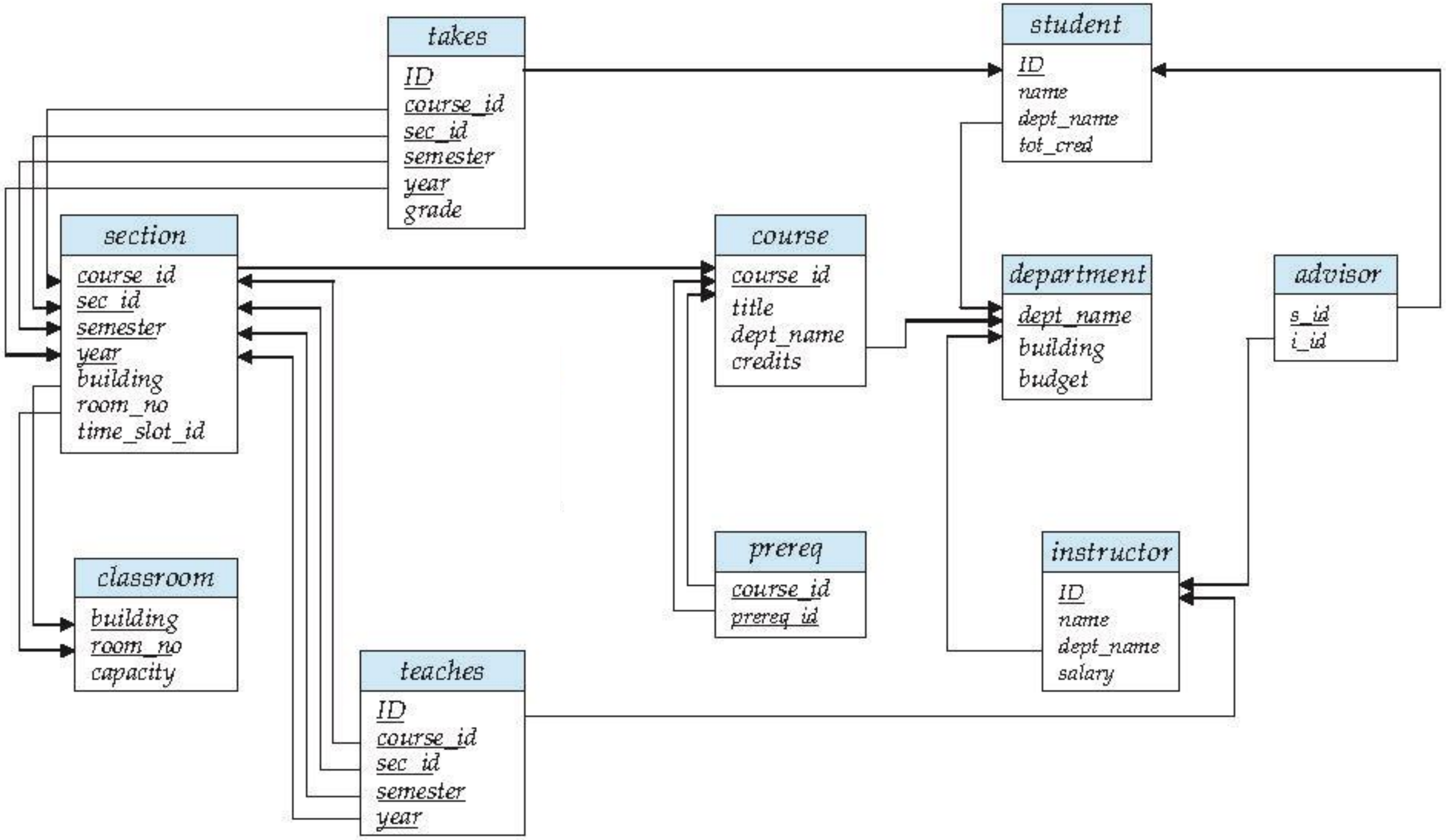
- You can also specify multiple attributes to be PRIMARY KEY
- product_name and country of origin are the key for the sells relation
- Example:

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

Foreign Keys – Referential Integrity

- 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
  - 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.
  - 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.

Foreign Keys – Referential Integrity (Cont.)



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
);
```

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

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

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.
  - Cascade : Make the same changes in course.
    - Deleted dept_name: delete course tuple.
    - Updated dept_name : change value in course.
  - 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 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 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';
```

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.

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
);
```

NOT NULL

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

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

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
);
```

Test the database with the following query

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

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')
  ```

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))
);
```

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 pretty much any database

Functional Dependency (FD)

Functional Dependence (FD)

- 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

Functional Dependence

- 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

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

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

$$name \longrightarrow dept\_name$$

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
  - there is no $X \subset K, X \nrightarrow R$
    - make sure key has minimum set of attributes (*minimality*)
- Question
  - When this definition will not hold?

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 $\rightarrow$ dept_name
  name, dept_name $\rightarrow$ 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 $\mathcal{F}$.
- We denote the closure of $\mathcal{F}$ by $\mathcal{F}^+$.
- We can find all of $\mathcal{F}^+$ by applying Armstrong's Axioms:
  - if $X \subseteq Y$, then $Y \longrightarrow X$ (*reflexivity*)
  - if $\mathrm{X} \longrightarrow Y$, then $\mathrm{AX} \longrightarrow AY$ (*augmentation*)
  - if $X \longrightarrow Y$ and $Y \longrightarrow W$, then $X \longrightarrow W$ (*transitivity*)

  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 $\mathrm{X} \longrightarrow Y, \forall Y$
- What about the right hand side (RHS) of the dependency?
  - $\mathrm{X} \longrightarrow Y \implies X \longrightarrow B \qquad \forall B \in Y$
    - We can restrict the RHS to have only a single attribute

Examples of Armstrong's Axioms

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

  name $\to$ name
  name, dept_name $\to$ name
  name, dept_name $\to$ dept_name

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

  name $\to$ dept_name
  name, salary $\to$ dept_name

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

  name $\to$ dept_name implies
  id $\to$ dept_name

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

$X \longrightarrow Y$ 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)

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

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

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

Use of FDs

- 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

- Exercise: If you know that in the goal relation

  $$player \rightarrow teamid$$

  Write python script to check for violations

Conditional Functional Dependencies (CFDs)

- 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

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

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

- 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:

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

Denial Constraints (DCs) – Examples

- Expressing functional dependency as a DC – consider *zip* $\rightarrow$ *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

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

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

- Uniqueness constraint

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

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.