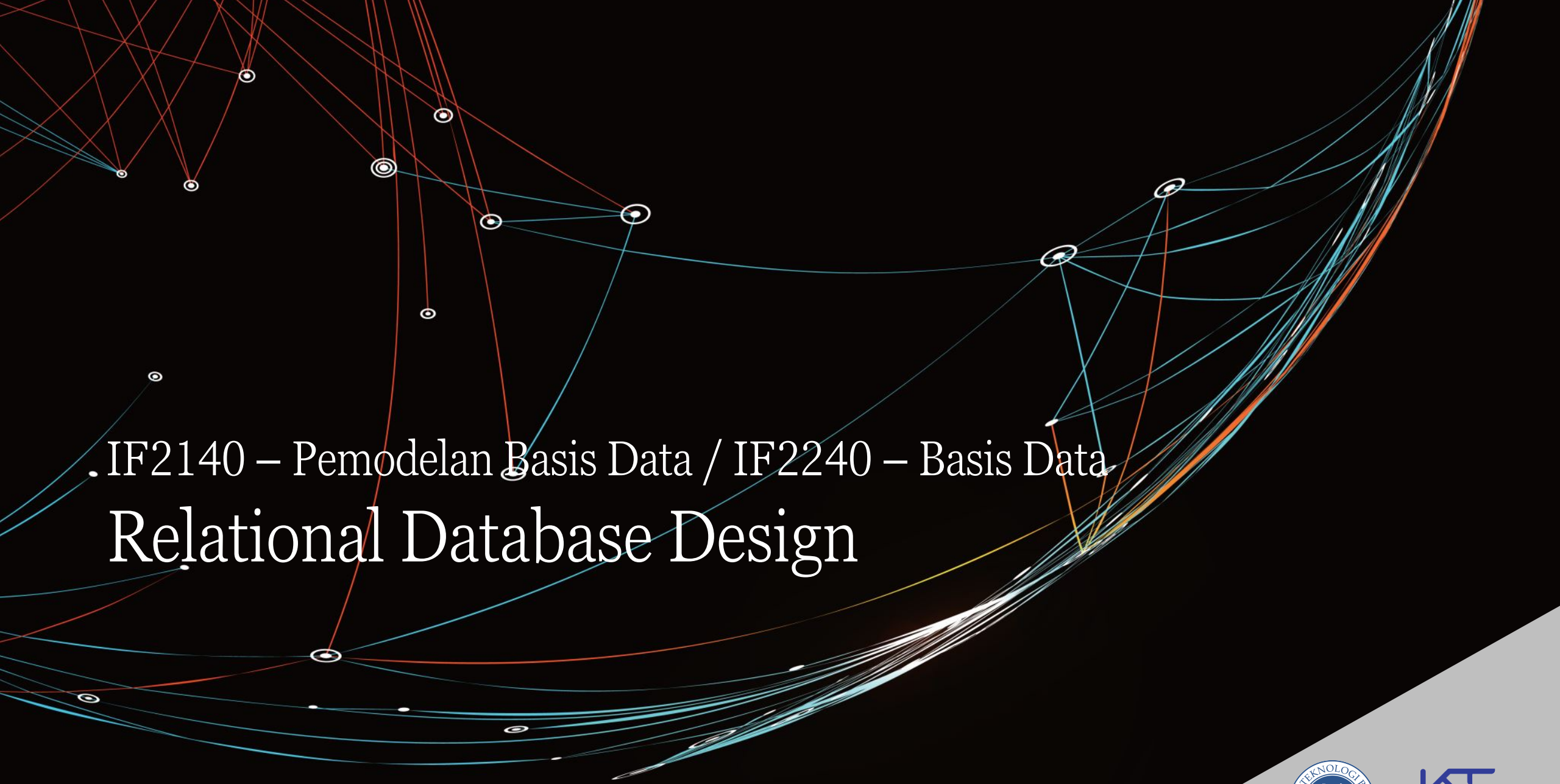
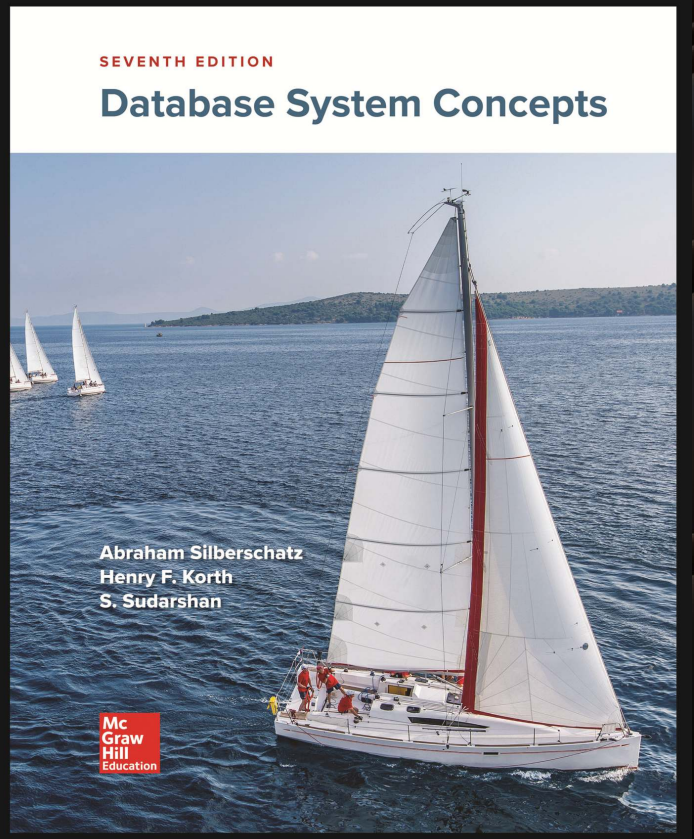


IF2140 – Pemodelan Basis Data / IF2240 – Basis Data

Relational Database Design

KNOWLEDGE & SOFTWARE ENGINEERING



Sumber

- Silberschatz, Korth, Sudarshan: "Database System Concepts", 7th Edition
  - Chapter 7: Relational Database Design

Capaian

- Mahasiswa dapat menghasilkan desain basis data relasional yang baik berdasarkan prinsip-prinsip yang diberikan

Outline

Features of Good Relational Design

Functional Dependencies

Decomposition Using Functional Dependencies

Normal Forms

Functional Dependency Theory

Algorithms for Decomposition using Functional Dependencies

Database-Design Process

Features of Good Relational Designs

Suppose we combine *instructor* and *department* into *in_dep,* which represents the natural join on the relations *instructor* and *department*

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

There is repetition of information

Need to use null values (if we add a new department with no instructors)

A Combined Schema Without Repetition

Not all combined schemas result in repetition of information

Consider combining relations

- *sec_class(sec_id, building, room_number)* and
- *section(course_id, sec_id, semester, year)*

into one relation

- *section(course_id, sec_id, semester, year, building, room_number)*

No repetition in this case

Decomposition

- The only way to avoid the repetition-of-information problem in the *in_dep* schema is to decompose it into two schemas – instructor and *department* schemas.
- Not all decompositions are good. Suppose we decompose

  employee(ID, name, street, city, salary)

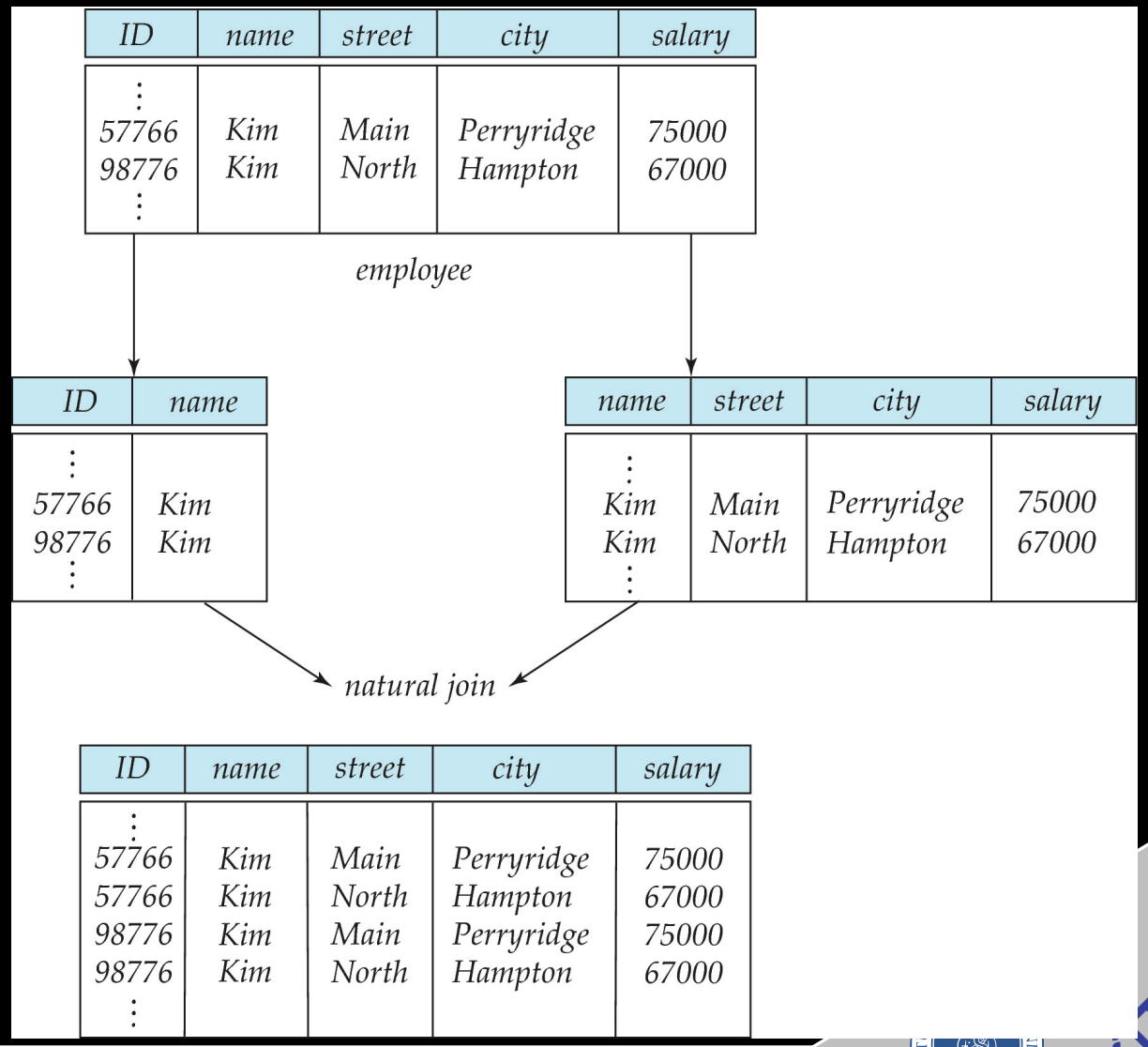
  into

  employee1 (*ID*, *name*)

  employee2 (*name*, *street*, *city*, *salary*)

  The problem arises when we have two employees with the same name
- The next slide shows how we lose information -- we cannot reconstruct the original *employee* relation -- and so, this is a lossy decomposition

A Lossy Decomposition



Lossless Decomposition

- Let R be a relation schema and let R_1 and R_2 form a decomposition of R . That is R = R_1 U R_2
- We say that the decomposition is a lossless decomposition if there is no loss of information by replacing R with the two relation schemas R_1 U R_2
- Formally,

$$\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$$

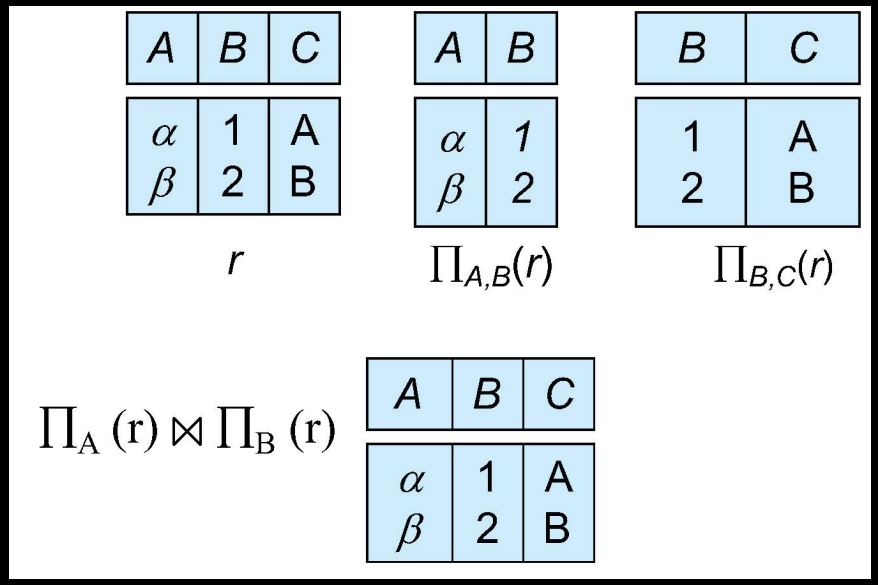
- And, conversely a decomposition is lossy if

$$r \subset \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$$

Example of Lossless Decomposition

Decomposition of $R = (A, B, C)$

- $R_1 = (A, B)$
- $R_2 = (B, C)$



Normalization Theory

- Decide whether a particular relation R is in "good" form.
- In the case that a relation R is not in "good" form, decompose it into set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - Each relation is in good form
  - The decomposition is a lossless decomposition
- Our theory is based on:
  - Functional dependencies
  - Multivalued dependencies

Functional Dependencies

- There are usually a variety of constraints (rules) on the data in the real world.
- For example, some of the constraints that are expected to hold in a university database are:
  - Students and instructors are uniquely identified by their ID.
  - Each student and instructor has only one name.
  - Each instructor and student is (primarily) associated with only one department.
  - Each department has only one value for its budget, and only one associated building.

Functional Dependencies (Cont.)

- An instance of a relation that satisfies all such real-world constraints is called a legal instance of the relation;
- A legal instance of a database is one where all the relation instances are legal instances
- Constraints on the set of legal relations.
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.
- A functional dependency is a generalization of the notion of a *key*.

Functional Dependencies Definition

Let R be a relation schema

$$\alpha \subseteq R \quad and \quad \beta \subseteq R$$

The functional dependency

$$\alpha \rightarrow \beta$$

holds on R if and only if for any legal relations $r(R)$, whenever any two tuples t_1 and t_2 of r agree on the attributes α, they also agree on the attributes β. That is,

$$t_1[\alpha] = t_2[\alpha] \quad \Rightarrow \quad t_1[\beta] = t_2[\beta]$$

Example: Consider $r(A,B)$ with the following instance of r.

1	4
1	5
3	7

On this instance, $B \rightarrow A$ hold; $A \rightarrow B$ does NOT hold

Closure of a Set of Functional Dependencies

- Given a set F set of functional dependencies, there are certain other functional dependencies that are logically implied by F.
  - If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
  - etc.
- The set of all functional dependencies logically implied by F is the closure of F.
- We denote the *closure* of F by F^+.

Keys and Functional Dependencies

- K is a superkey for relation schema R if and only if $K \rightarrow R$
- K is a candidate key for R if and only if
  - $K \rightarrow R$, and
  - for no $\alpha \subset K$, $\alpha \rightarrow R$
- Functional dependencies allow us to express constraints that cannot be expressed using superkeys. Consider the schema:

  in_dep (*ID, name, salary, dept_name, building, budget*).

  We expect these functional dependencies to hold:

  dept_name→ *building*

  ID → *building*

  but would not expect the following to hold:

  dept_name → *salary*

Use of Functional Dependencies

- We use functional dependencies to:
  - 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 *F* of functional dependencies, we say that *r* satisfies *F*.
  - To specify constraints on the set of legal relations
    - We say that *F* holds on *R* if all legal relations on *R* satisfy the set of functional dependencies *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 → *ID*.

Trivial Functional Dependencies

- *A* functional dependency is trivial if it is satisfied by all instances of a relation
- Example*:*
  - *ID, name* $\rightarrow$ *ID*
  - *name* $\rightarrow$ *name*
- In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$