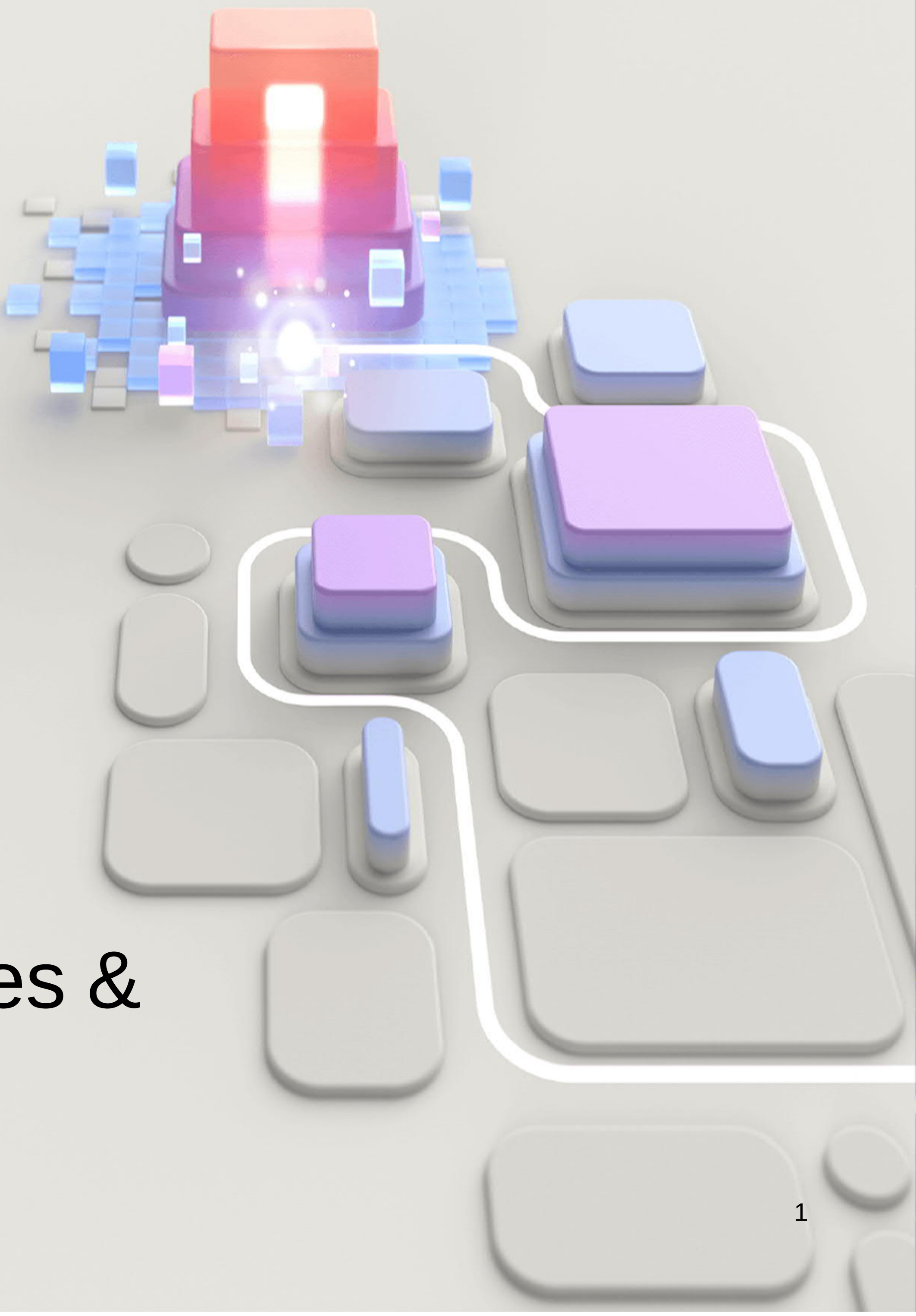


Functional Dependencies & Table Decomposition

Goal…

- Given a database schema, how do you judge whether or not the design is good?

- How do you ensure it does not have redundancy or anomaly problems?

- To ensure your database schema is in a good form we use:
  - Functional Dependencies
  - Normalization Rules

What is Normalization

- Normalization is a set of rules to systematically achieve a good design

- **If these rules are followed, then the DB design is guarantee to avoid several problems:**
  - Inconsistent data
  - *Anomalies*: insert, delete and update
  - Redundancy: which wastes storage, and often slows down query processing

Problem I: Insert Anomaly

Student

sNumber	sName	pNumber	pName
s1	Dave	p1	MM
s2	Greg	p2	ER

Student Info   Professor Info

Question: Could we insert a professor without student?
Note: We cannot insert a professor who has no students.

Insert Anomaly: We are not able to insert “valid” value/(s)

Problem II: Delete Anomaly

Student

sNumber	sName	pNumber	pName
s1	Dave	p1	MM
s2	Greg	p2	ER

Student Info

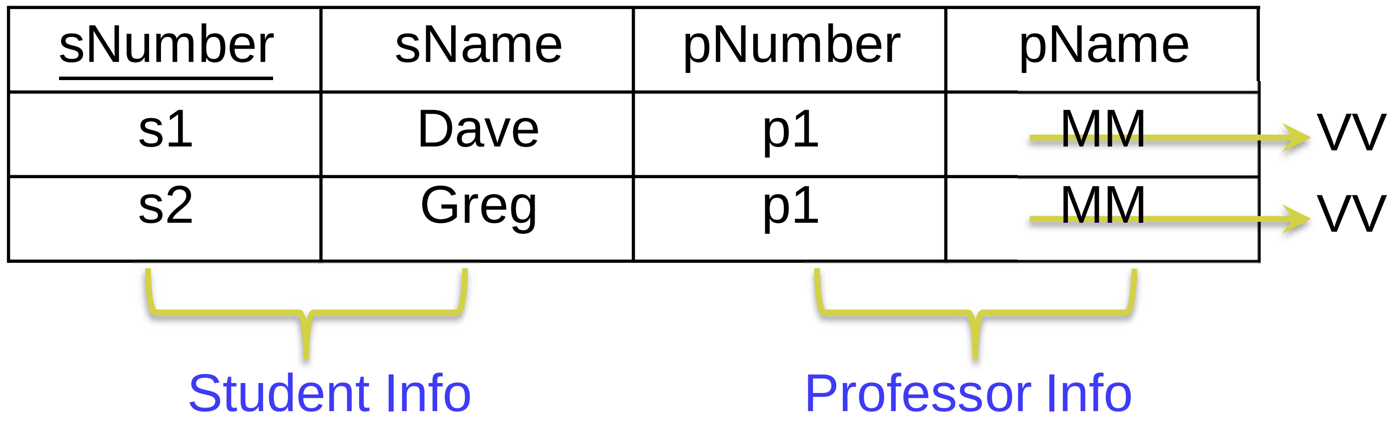
Professor Info

Question: Can we delete a student and keep a professor info ?
Note: We cannot delete a student that is the only student of a professor.

Delete Anomaly: We are not able to perform a delete without losing some "valid" information.

Problem III: Update Anomaly

Student



sNumber	sName	pNumber	pName
s1	Dave	p1	MM $\rightarrow$ VV
s2	Greg	p1	MM $\rightarrow$ VV

Student Info
Professor Info

Question: Can we simply update a professor's name ?
Note: To update the name of a professor, we have to update in multiple tuples.

Update Anomaly: To update a value, we have to update multiple rows. Update anomalies are due to redundancy.

Problem IV: Inconsistency

Student

sNumber	sName	pNumber	pName
s1	Dave	p1	MM
s2	Greg	p1	MM $\rightarrow$ VV

Student Info   Professor Info

What if the name of professor p1 is updated in one place and not the other!!!

Inconsistent Data: The same object has multiple values.
Inconsistency is due to redundancy.

Schema Normalization

- **Following the normalization rules, we avoid**
  - Insert anomaly
  - Delete anomaly
  - Update anomaly
  - Inconsistency

Functional Dependencies

Usage of Functional Dependencies

- Discover all dependencies between attributes
- Identify the keys of relations
- Enable good (Lossless) decomposition of a given relation

Functional Dependencies (FDs)

Student

<u>sNumber</u>	sName	address
1	Dave	144FL
2	Greg	320FL

Suppose we have the FD: *sNumber → address*

That is, there is a functional dependency from ***sNumber*** to ***address***

Meaning: A student number determines the student address

Or: For any two rows in the Student relation with the same value for sNumber, the value for address must be same.

Functional Dependencies (FDs)

- 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*
- FD: $\underbrace{A1,A2,\dots An}_{\text{L.H.S}} \;\square\; \underbrace{B1, B2,\dots Bm}_{\text{R.H.S}}$

Functional Dependencies (FDs)

- The basic form of a FDs

A1,A2,…An □ *B1, B2,…Bm*

L.H.S R.H.S

>> The values in the L.H.S uniquely determine the values in the R.H.S attributes
(when you lookup the DB)

>> It does not mean that L.H.S values *compute* the R.H.S values

Examples:

SSN →→ personName, personDoB, personAddress

DepartmentID, CourseNum →→ CourseTitle, NumCredits

personName $\not\rightarrow$ **personAddress**

FD and Keys

Student

<u>sNumber</u>	sName	address
1	Dave	144FL
2	Greg	320FL

Primary Key : <sNumber>

Questions :

•Does a primary key implies functional dependencies? Which ones ?
•Does unique keys imply functional dependencies? Which ones ?
•Does a functional dependency imply keys ? Which ones ?

Observation :

Any key (primary or candidate) or superkey of a relation R functionally determines all attributes of R.

Functional Dependencies & Keys

- ***K* is a superkey** for relation schema *R* if and only if
  - $K \rightarrow R$ -- *K determines all attributes of R*
- ***K* is a candidate key** for *R* if and only if
  - $K \rightarrow R$, and
  - No $\alpha \subset K$, $\alpha \rightarrow R$

Keys imply FDs, and FDs imply keys

Example I

Student(SSN, Fname, Mname, Lname, DoB, address, age, admissionDate)

- **If you know that SSN is a key, Then**
  - SSN →→ Fname, Mname, Lname, DoB, address, age, admissionDate

- **If you know that (Fname, Mname, Lname) is a key, Then**
  - $\underbrace{\text{Fname, Mname, Lname}}$ →→ SSN, DoB, address, age, admissionDate

Need to know all of L.H.S to determine any of the R.H.S

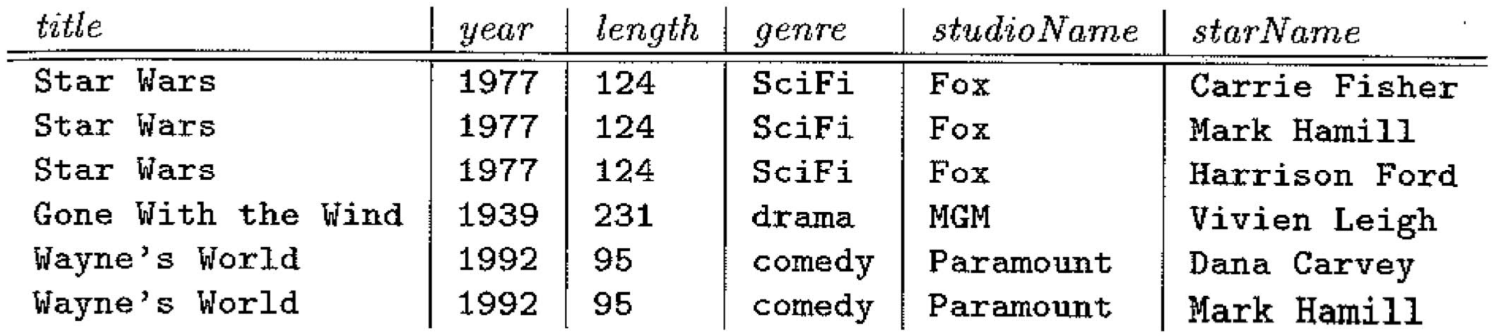
Example II

Student(SSN, Fname, Mname, Lname, DoB, address, age, admissionDate)

- **If you know that SSN →→ Fname, Mname, Lname, DoB, address, age, admissionDate**
  - Then, we infer that SSN is a candidate key

- **If you know that Fname, Mname, Lname →→ SSN, DoB, address, age, admissionDate**
  - Then, we infer that (Fname, Mname, Lname) is a key. **Is it Candidate or super key???**
  - **Does any pair of attributes together form a key??**
    - **If no →→** (Fname, Mname, Lname) is a candidate key (minimal)
    - **If yes →→** (Fname, Mname, Lname) is a super key

Example III



title	*year*	*length*	*genre*	*studioName*	*starName*
Star Wars	1977	124	SciFi	Fox	Carrie Fisher
Star Wars	1977	124	SciFi	Fox	Mark Hamill
Star Wars	1977	124	SciFi	Fox	Harrison Ford
Gone With the Wind	1939	231	drama	MGM	Vivien Leigh
Wayne's World	1992	95	comedy	Paramount	Dana Carvey
Wayne's World	1992	95	comedy	Paramount	Mark Hamill

- Does this FD hold?
  - Title, year →→ length, genre, studioName

YES

- Does this FD hold?
  - Title, year →→ starName

NO

- What is a key of this relation instance?
  - {title, year, starName}
  - Is it candidate key?

>> For this instance →→ not a candidate key
(title, starName) can be a key

Important Definitions

Lossless Decomposition

Lossless decomposition is a concept in database normalization that ensures when you break a relation (table) into two or more smaller relations, you can reconstruct the original relation without losing any data by performing a join.

Natural Join

Natural join (⋈) is a binary operator that is written as (*R* ⋈ *S*) where *R* and *S* are relations (set of tuples → table, as discussed earlier).

The result of the natural join is the set of all combinations of tuples in *R* and *S* that are equal on their common attribute names.

Lossy & Lossless Decomposition

Decomposing Relations

StudentProf

sNumber	sName	pNumber	pName
s1	Dave	p1	MM
s2	Greg	p2	MM

FDs: pNumber □ pName

Lossless

Student

sNumber	sName	pNumber
s1	Dave	p1
s2	Greg	p2

Professor

pNumber	pName
p1	MM
p2	MM

Lossy

Student

sNumber	sName	pName
S1	Dave	MM
S2	Greg	MM

Professor

pNumber	pName
p1	MM
p2	MM

Lossless vs. Lossy Decomposition

- Assume R is divided into R1 and R2

- **Lossless Decomposition**
  - *R1 natural join R2* should create exactly R

- **Lossy Decomposition**
  - *R1 natural join R2* adds more records (or deletes records) from R

Lossless Decomposition

StudentProf

sNumber	sName	pNumber	pName
s1	Dave	p1	MM
s2	Greg	p2	MM

FDs: pNumber pName

Lossless

Student

sNumber	sName	pNumber
s1	Dave	p1
s2	Greg	p2

Professor

pNumber	pName
p1	MM
p2	MM

Student & Professor are **lossless decomposition** of **StudentProf**
(Student ⋈ Professor = StudentProf)

Lossy Decomposition

StudentProf

sNumber	sName	pNumber	pName
s1	Dave	p1	MM
s2	Greg	p2	MM

FDs: pNumber □ pName

Lossy

Student

sNumber	sName	pName
S1	Dave	MM
S2	Greg	MM

Professor

pNumber	pName
p1	MM
p2	MM

Student & Professor are **lossy decomposition** of **StudentProf**
(Student ⋈ Professor != StudentProf)

Goal: Ensure Lossless Decomposition

- **How to ensure lossless decomposition?**

- **Answer:**
  - The common columns must be candidate key in one of the two relations

Back to our example

StudentProf

sNumber	sName	pNumber	pName
s1	Dave	p1	MM
s2	Greg	p2	MM

FDs: pNumber pName

pNumber is candidate key

Lossless

Student

sNumber	sName	pNumber
s1	Dave	p1
s2	Greg	p2

Professor

pNumber	pName
p1	MM
p2	MM

pName is not candidate key

Lossy

Student

sNumber	sName	pName
S1	Dave	MM
S2	Greg	MM

Professor

pNumber	pName
p1	MM
p2	MM