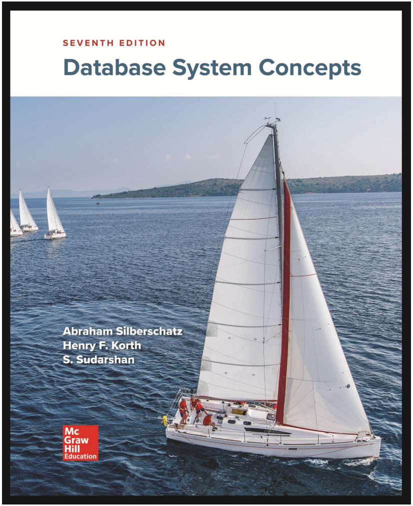
Database System Concepts, 7^{th} Edition
Chapter 7: Relational Database Design

Silberschatz, Korth and Sudarshan

April 23, 2025

Database System Concepts



Content has been extracted from *Database System Concepts*, Seventh Edition, by Silberschatz, Korth and Sudarshan. Mc Graw Hill Education. 2019.
Visit `https://db-book.com/`.

Plan

- Features of Good Relational Design
- Functional Dependencies
- Decomposition Using Functional Dependencies
- Normal Forms
- Functional Dependency Theory

Features of Good Relational Designs

classroom(<u>*building*</u>, <u>*room_number*</u>, *capacity*)
department(<u>*dept_name*</u>, *building*, *budget*)
course(<u>*course_id*</u>, *title*, *dept_name*, *credits*)
instructor(<u>*ID*</u>, *name*, *dept_name*, *salary*)
section(<u>*course_id*</u>, <u>*sec_id*</u>, <u>*semester*</u>, <u>*year*</u>, *building*, *room_number*, *time_slot_id*)
teaches(<u>*ID*</u>, <u>*course_id*</u>, <u>*sec_id*</u>, <u>*semester*</u>, <u>*year*</u>)
student(<u>*ID*</u>, *name*, *dept_name*, *tot_cred*)
takes(<u>*ID*</u>, <u>*course_id*</u>, <u>*sec_id*</u>, <u>*semester*</u>, <u>*year*</u>, *grade*)
advisor(<u>*s_ID*</u>, *i_ID*)
time_slot(<u>*time_slot_id*</u>, <u>*day*</u>, <u>*start_time*</u>, *end_time*)
prereq(<u>*course_id*</u>, <u>*prereq_id*</u>)

Figure 7.1 Database schema for the university example.

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

Figure 7.2 The *in_dep* relation.

- There is repetition of information
- Need to use `null` values (if we add a new department with no instructors)

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

ID	*name*	*street*	*city*	*salary*
⋮				
57766	*Kim*	*Main*	*Perryridge*	*75000*
98776	*Kim*	*North*	*Hampton*	*67000*
⋮				

employee

ID	*name*
⋮	
57766	*Kim*
98776	*Kim*
⋮	

name	*street*	*city*	*salary*
⋮			
Kim	*Main*	*Perryridge*	*75000*
Kim	*North*	*Hampton*	*67000*
⋮			

natural join

ID	*name*	*street*	*city*	*salary*
⋮				
57766	*Kim*	*Main*	*Perryridge*	*75000*
57766	*Kim*	*North*	*Hampton*	*67000*
98776	*Kim*	*Main*	*Perryridge*	*75000*
98776	*Kim*	*North*	*Hampton*	*67000*
⋮				

Figure 7.3 Loss of information via a bad decomposition.

Lossless Decomposition

- Let R be a relation schema and let R_1 and R_2 form a decomposition of R. That is $R = R_1 \bigcup R_2$.
- We say that the decomposition is a **lossless decomposition** if there is no loss of information by replacing R with the two relation schemas $R_1 \bigcup R_2$.
- 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 $R = (A, B, C)$:

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

A	B	C
α	1	A
β	2	B

r

A	B
α	1
β	2

$\Pi_{A,B}(r)$

B	C
1	A
2	B

$\Pi_{B,C}(r)$

$\Pi_A (r) \bowtie \Pi_B (r)$

A	B	C
α	1	A
β	2	B

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 $\{R_1, R_2, \cdots, R_n\}$ such that,
  - Each relation is in good form.
  - The decomposition is a lossless decomposition.
- Our theory is based on:
  1. functional dependencies.
  2. multivalued dependencies.

Plan

Features of Good Relational Design

Functional Dependencies

Decomposition Using Functional Dependencies

Normal Forms

Functional Dependency Theory

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.
- 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

$$\alpha \subseteq R \text{ and } \beta \subseteq R$$

- 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,

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

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

A	**B**
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 $A \rightarrow C$.
- The set of **all** functional dependencies logically implied by F is the **closure** of F.
- We denote the closure of F by F^+.

Plan

Keys and Functional Dependencies

- K is a superkey for relation schema R if and only if $K \rightarrow R$.
- K is a primary 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:

$$dept\_name \rightarrow building$$
$$ID \rightarrow building$$

  but would not expect the following to hold:

$$dept\_name \rightarrow salary$$

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

$$name \rightarrow ID.$$

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 $\beta \subseteq \alpha$.

Lossless Decomposition

- We can use functional dependencies to show when certain decomposition are lossless.
- For the case of $R = (R_1, R_2)$, we require that for all possible relations r on schema R

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

- A decomposition of R into R_1 and R_2 is lossless decomposition if at least one of the following dependencies is in F^+:

$$R_1 \bigcap R_2 \rightarrow R_1$$

$$R_1 \bigcap R_2 \rightarrow R_2$$

- The above functional dependencies are a sufficient condition for lossless join decomposition; the dependencies are a necessary condition only if all constraints are functional dependencies.

Example[1]

- $R = (A, B, C)$
$$F = \{A \rightarrow B \\ B \rightarrow C\}$$
- $R_1 = (A, B)$; $R_2 = (B, C)$
Lossless decomposition:
$R_1 \bigcap R_2 = \{B\}$ and $B \rightarrow BC$ holds.
- $R_1 = (A, B)$; $R_2 = (A, C)$
Lossless decomposition:
$R_1 \bigcap R_2 = \{A\}$ and $A \rightarrow AC$ holds.

[1] Note: $B \rightarrow BC$ is a shorthand notation for $B \rightarrow \{B, C\}$.

Dependency Preservation

- Testing functional dependency constraints each time the database is updated can be costly,
- It is useful to design the database in a way that constraints can be tested efficiently.
- If testing a functional dependency can be done by considering just one relation, then the cost of testing this constraint is low.
- When decomposing a relation it is possible that it is no longer possible to do the testing without having to perform a Cartesian Produced.
- A decomposition that makes it computationally hard to enforce functional dependency is said to be NOT **dependency preserving**.

Dependency Preservation Example

- Consider a schema:

  `dept_advisor(s_ID, i_ID, department_name)`

- With function dependencies:

$$i\_ID \rightarrow dept\_name$$
$$s\_ID, dept\_name \rightarrow i\_ID$$

- In the above design we are forced to repeat the department name once for each time an instructor participates in a *dept_advisor* relationship.
- To fix this, we need to decompose *dept_advisor*.
- Any decomposition will not include all the attributes in:

$$s\_ID, dept\_name \rightarrow i\_ID$$

- Thus, the composition NOT be dependency preserving.

Plan

Features of Good Relational Design

Functional Dependencies

Decomposition Using Functional Dependencies

Normal Forms

Functional Dependency Theory

Boyce-Codd Normal Form

- A relation schema R is in BCNF with respect to a set F of functional dependencies if for all functional dependencies in F^+ of the form:

  $$\alpha \rightarrow \beta$$

  where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
  - α is a superkey for R.

Boyce-Codd Normal Form (Cont.)

- Example schema that is not in BCNF:

  `in_dep (ID, name, salary, dept_name, building, budget)`

  because :
  - $dept\_name \rightarrow building, budget$
    - holds on `in_dep`
    - but...
  - *dept_name* is not a superkey.
- When decompose `in_dept` into `instructor` and `department`:
  - `instructor` is in BCNF.
  - `department` is in BCNF.

Decomposing a Schema into BCNF

- Let R be a schema that is not in BCNF. Let $\alpha \rightarrow \beta$ be the Fuctional Dependency that causes a violation of BCNF.
- We decompose R into:
  - $(\alpha \bigcup \beta)$
  - $(R - (\beta - \alpha))$
- In our example of `in_dep`,
  - $\alpha = dept\_name$
  - $\beta = building, budget$

  and `in_dep` is replaced by
  - $(\alpha \bigcup \beta) = (dept\_name, building, budget)$
  - $(R - (\beta - \alpha)) = (ID, name, salary, dept\_name)$

BCNF and Dependency Preservation

- It is not always possible to achieve both BCNF and dependency preservation.
- Consider a schema:

  `dept_advisor(s_ID, i_ID, department_name)`

- With function dependencies:

$$\begin{aligned} i\_ID \rightarrow &\quad dept\_name \\ s\_ID, dept\_name \rightarrow &\quad i\_ID \end{aligned}$$

- `dept_advisor` is not in BCNF:
  - i_ID is not a superkey.
- Any decomposition of `dept_advisor` will not include all the attributes in:

$$s\_ID, dept\_name \rightarrow \quad i\_ID$$

- Thus, the composition is NOT be dependency preserving.

Third Normal Form

- A relation schema R is in **third normal form (3NF)** if for all:

$$\alpha \to \beta \text{ in } F^+$$

  at least one of the following holds:
  - $\alpha \to \beta$ is trivial (i.e., $\beta \in \alpha$).
  - α is a superkey for R.
  - Each attribute A in $\beta - \alpha$ is contained in a candidate key[2] for R.
- If a relation is in BCNF it is in 3NF (since in BCNF one of the first two conditions above must hold).
- Third condition is a minimal relaxation of BCNF to ensure dependency preservation (will see why later).

[2] NOTE: each attribute may be in a different candidate key.

3NF Example

- Consider a schema:

  `dept_advisor(s_ID, i_ID, dept_name)`

- With function dependencies:

$$i\_ID \rightarrow dept\_name$$
$$s\_ID, dept\_name \rightarrow i\_ID$$

- Two candidate keys = $\{s\_ID, dept\_name\}$, $\{s\_ID, i\_ID\}$
- We have seen before that `dept_advisor` is not in BCNF.
- R, however, is in 3NF:
  - s_ID, $dept\_name$ is a superkey,
  - $i\_ID \rightarrow dept\_name$ and i_ID is NOT a superkey, but:
    - $dept\_name - i\_ID = dept\_name$ and
    - $dept\_name$ is contained in a candidate key.

Redundancy in 3NF

- Consider the schema R below, which is in 3NF:
  - $R = (J, K, L)$,
  - $F = \{JK \rightarrow L, L \rightarrow K\}$,
  - And an instance table:

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
`null`	l_2	k_2

- What is wrong with the table?

Redundancy in 3NF

- Consider the schema R below, which is in 3NF:
  - $R = (J, K, L)$,
  - $F = \{JK \rightarrow L, L \rightarrow K\}$,
  - And an instance table:

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
`null`	l_2	k_2

- What is wrong with the table?
  - Repetition of information,
  - Need to use `null` values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J)

Comparison of BCNF and 3NF

- Advantages to 3NF over BCNF. It is always possible to obtain a 3NF design without sacrificing losslessness or dependency preservation.
- Disadvantages to 3NF.
  - We may have to use `null` values to represent some of the possible meaningful relationships among data items.
  - There is the problem of repetition of information.

Goals of Normalization

- Let R be a relation scheme with a set F of functional dependencies.
- Decide whether a relation scheme R is in "good" form.
- In the case that a relation scheme R is not in "good" form, decompose it into a set of relation scheme $\{R_1, R_2, \cdots, R_n\}$ such that:
  - Each relation scheme is in good form,
  - The decomposition is a lossless decomposition,
  - Preferably, the decomposition should be dependency preserving.

How good is BCNF?

- There are database schemas in BCNF that do not seem to be sufficiently normalized...
- Consider a relation:

  `inst_info(ID, child_name, phone)`

  where an instructor may have more than one phone and can have multiple children.
- Instance of `inst_info`

ID	**child_name**	**phone**
99999	David	512-555-1234
99999	David	512-555-4321
99999	William	512-555-1234
99999	William	512-555-4321

How good is BCNF? (Cont.)

- There are no non-trivial functional dependencies and therefore the relation is in BCNF.
- Insertion anomalies – i.e., if we add a phone 981-992-3443 to 99999, we need to add two tuples:

  (99999, David, 981-992-3443)
  (99999, William, 981-992-3443)

Higher Normal Forms

- It is better to decompose `inst_info` into:
  - `inst_child`

ID	child_name
99999	David
99999	William

  - `inst_phone`

ID	phone
99999	512-555-1234
99999	512-555-4321

- This suggests the need for higher normal forms, such as Fourth Normal Form (4NF), which we shall see later

Plan

Functional-Dependency Theory Roadmap

- We now consider the formal theory that tells us which functional dependencies are implied logically by a given set of functional dependencies.
- We then develop algorithms to generate lossless decompositions into BCNF and 3NF.
- We then develop algorithms to test if a decomposition is dependency-preserving.

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^+.

Closure of a Set of Functional Dependencies

- We can compute F^+, the closure of F, by repeatedly applying **Armstrong's Axioms**:
  - **Reflexive rule**: if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$,
  - **Augmentation rule**: if $\alpha \rightarrow \beta$, then $\gamma\alpha \rightarrow \gamma\beta$,
  - **Transitivity rule**: if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$.
- These rules are:
  - **sound** – generate only functional dependencies that actually hold, and
  - **complete** – generate all functional dependencies that hold.

Example of F^+

- $R = (A, B, C, G, H, I)$

$$
\begin{aligned}
F = \{A &\rightarrow B \\
A &\rightarrow C \\
CG &\rightarrow H \\
CG &\rightarrow I \\
B &\rightarrow H\}
\end{aligned}
$$

- Some members of F^+
  - $A \rightarrow H$ by transitivity from $A \rightarrow B$ and $B \rightarrow H$.
  - $AG \rightarrow I$ by augmenting $A \rightarrow C$ with G, to get $AG \rightarrow CG$ and then transitivity with $CG \rightarrow I$.
  - $CG \rightarrow HI$ by augmenting $CG \rightarrow I$ to infer $CG \rightarrow CGI$, and augmenting of $CG \rightarrow H$ to infer $CGI \rightarrow HI$, and then transitivity.

Closure of a Set of Functional Dependencies (Cont.)

- Additional rules:
  - **Union rule**: If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds.
  - **Decomposition rule**: If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds.
  - **Pseudotransitivity rule**: If $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds.
- The above rules can be inferred from Armstrong's axioms.

Procedure for Computing F^+

- To compute[3] the closure of a set of functional dependencies F:

$F^+ = F$
apply the reflexivity rule /* Generates all trivial dependencies */
repeat
    for each functional dependency f in F^+
        apply the augmentation rule on f
        add the resulting functional dependencies to F^+
    for each pair of functional dependencies f_1 and f_2 in F^+
        if f_1 and f_2 can be combined using transitivity
            add the resulting functional dependency to F^+
until F^+ does not change any further

Figure 7.7 A procedure to compute F^+.

[3]NOTE: We shall see an alternative procedure for this task later.

Closure of Attribute Sets

- Given a set of attributes α, define the closure of α under F (denoted by α^+) as the set of attributes that are functionally determined by α under F.
- Algorithm to compute α^+, the closure of α under F:

```
result := α;
repeat
    for each functional dependency β → γ in F do
        begin
            if β ⊆ result then result := result ∪ γ;
        end
until (result does not change)
```

Figure 7.8 An algorithm to compute α^+, the closure of α under F.

Example of Attribute Set Closure

- $R = (A, B, C, G, H, I)$

$$F = \{A \rightarrow B \\ A \rightarrow C \\ CG \rightarrow H \\ CG \rightarrow I \\ B \rightarrow H\}$$

- $(AG)^+$
  1. result = AG.
  2. result = $ABCG(A \rightarrow C$ and $A \rightarrow B)$.
  3. result = $ABCGH(CG \rightarrow H$ and $CG \subseteq AGBC)$.
  4. result = $ABCGHI(CG \rightarrow I$ and $CG \subseteq AGBCH)$.
- Is AG a candidate key?
  1. Is AG a super key?
     1.1 Does $AG \rightarrow R$? == Is $R \supseteq (AG)^+$.
  2. Is any subset of AG a superkey?
     2.1 Does $A \rightarrow R$? == Is $R \supseteq (A)^+$.
     2.2 Does $G \rightarrow R$? == Is $R \supseteq (G)^+$.
     2.3 In general: check for each subset of size $n - 1$.

Uses of Attribute Closure

There are several uses of the attribute closure algorithm:

- Testing for superkey
  - To test if α is a superkey, we compute α^+, and check if it contains all attributes of R.
- Testing functional dependencies
  - To check if a functional dependency $\alpha \to \beta$ holds (or, in other words, is in F^+), just check if $\beta \subseteq \alpha^+$.
  - That is, we compute α^+ by using attribute closure, and then check if it contains β.
- Computing closure of F
  - For each $\gamma \subseteq R$, we find the closure $\gamma +$, and for each $S \subseteq \gamma +$, we output a functional dependency $\gamma \to S$.

Canonical Cover

- When a user updates a relation, the database system must ensure that all functional dependencies in the set F are still satisfied.
- If any FD is violated, the update is rolled back.
- To minimize the effort of checking for violations, the system can test a simplified set of F with the same closure as the given set.
- This simplified set is termed the **canonical cover**.
- To define canonical cover we must first define **extraneous attributes**:
  - An attribute of a functional dependency in F is extraneous if we can remove it without changing F^+.

Extraneous Attributes

- Removing an attribute from the left side of a functional dependency could make it a stronger constraint.
  - For example, if we have $AB \rightarrow C$ and remove B, we get the possibly stronger result $A \rightarrow C$. It may be stronger because $A \rightarrow C$ logically implies $AB \rightarrow C$, but $AB \rightarrow C$ does not, on its own, logically imply $A \rightarrow C$.
- But, depending on what our set F of functional dependencies happens to be, we may be able to remove B from $AB \rightarrow C$ safely.
  - For example, suppose that: $F = \{AB \rightarrow C, A \rightarrow D, D \rightarrow C\}$
  - Then we can show that F logically implies $A \rightarrow C$, making extraneous in $AB \rightarrow C$.

Extraneous Attributes (Cont.)

- Removing an attribute from the right side of a functional dependency could make it a weaker constraint.
  - For example, if we have $AB \to CD$ and remove C, we get the possibly weaker result $AB \to D$. It may be weaker because using just $AB \to D$, we can no longer infer $AB \to C$.
- But, depending on what our set F of functional dependencies happens to be, we may be able to remove C from $AB \to CD$ safely.
  - For example, suppose that: $F = \{AB \to CD, A \to C\}$.
  - Then we can show that even after replacing $AB \to CD$ by $AB \to D$, we can still infer $AB \to C$ and thus $AB \to CD$.

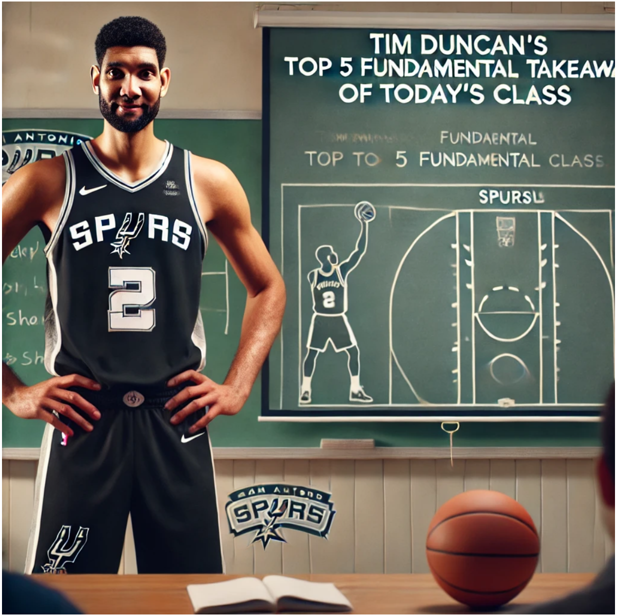
Extraneous Attributes

- An attribute of a functional dependency in F is extraneous if we can remove it without changing F^+.
- Consider a set F of functional dependencies and the functional dependency $\alpha \to \beta$ in F.
  - **Remove from the left side**: Attribute A is extraneous in α if:
    - $A \in \alpha$ and
    - F logically implies $(F - \{\alpha \to \beta\}) \bigcup \{(\alpha - A) \to \beta\}$.
  - **Remove from the right side**: Attribute A is extraneous in β if:
    - $A \in \beta$ and
    - The set of functional dependencies $(F - \{\alpha \to \beta\}) \bigcup \{\alpha \to (\beta - A)\}$ logically implies F.

Note: implication in the opposite direction is trivial in each of the cases above, since a "stronger" functional dependency always implies a weaker one

End of Chapter 7.

TDT5FTOTTC

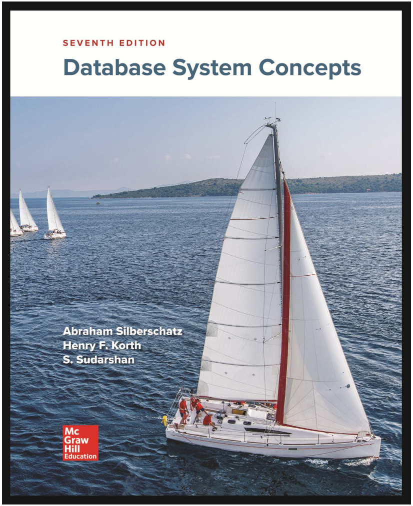

TIM DUNCAN'S
TOP 5 FUNDAMENTAL TAKEAW
OF TODAY'S CLASS
FUNDAENTAL
TOP TO 5 FUNDAMENTAL CLASS
SPURSL
SPURS
2
SPURS

Top 5 Fundamental Takeaways

Top 5 Fundamental Takeaways

5

Database System Concepts



Content has been extracted from *Database System Concepts*, Seventh Edition, by Silberschatz, Korth and Sudarshan. Mc Graw Hill Education. 2019.
Visit `https://db-book.com/`.