19CSE202 - Database Management Systems

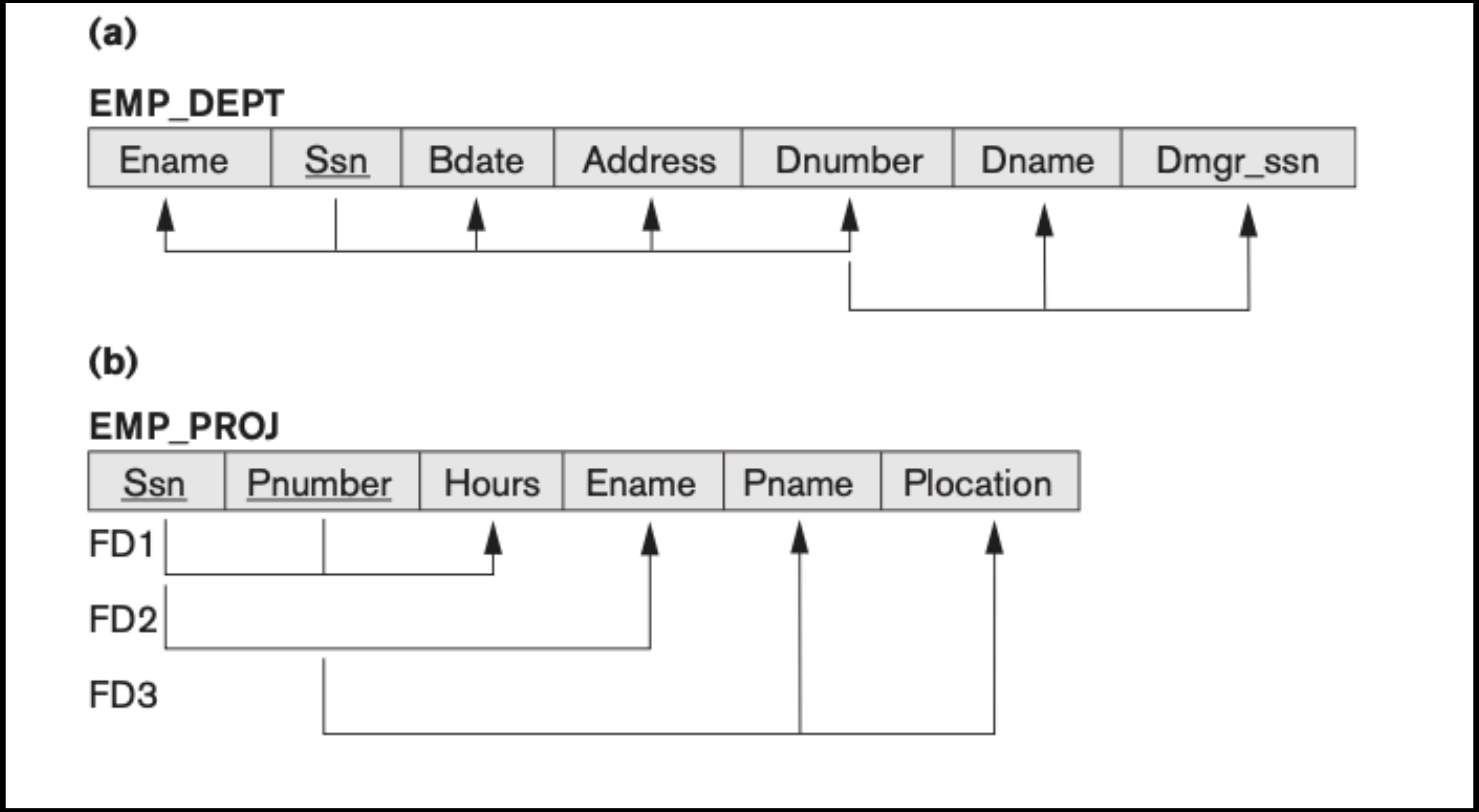
FUNCTIONAL DEPENDENCY THEORY - SESSION 1

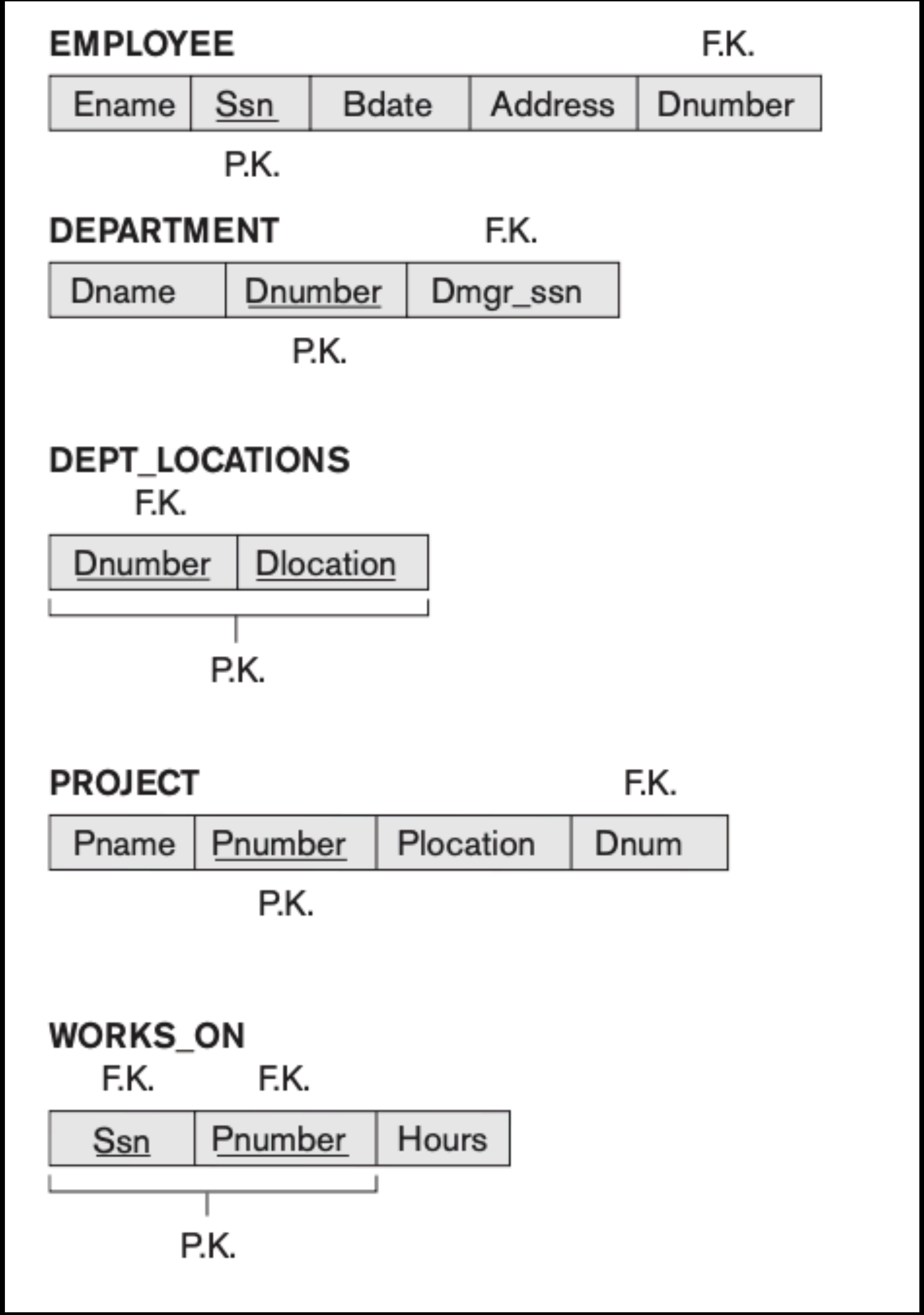
Ms. Vidhya. S
Assistant Professor (Sr. Gr.)
Department of Computer Science and Engineering
Amrita School of Engineering
Amrita Vishwa Vidyapeetham

Informal Design Guidelines for Relation Schemas

1. Semantics of the Relation Attributes

- ★ ***Guideline 1:*** Design a relation schema so that it is easy to explain its meaning. Do not combine attributes from multiple entity types and relationship types into a single relation.

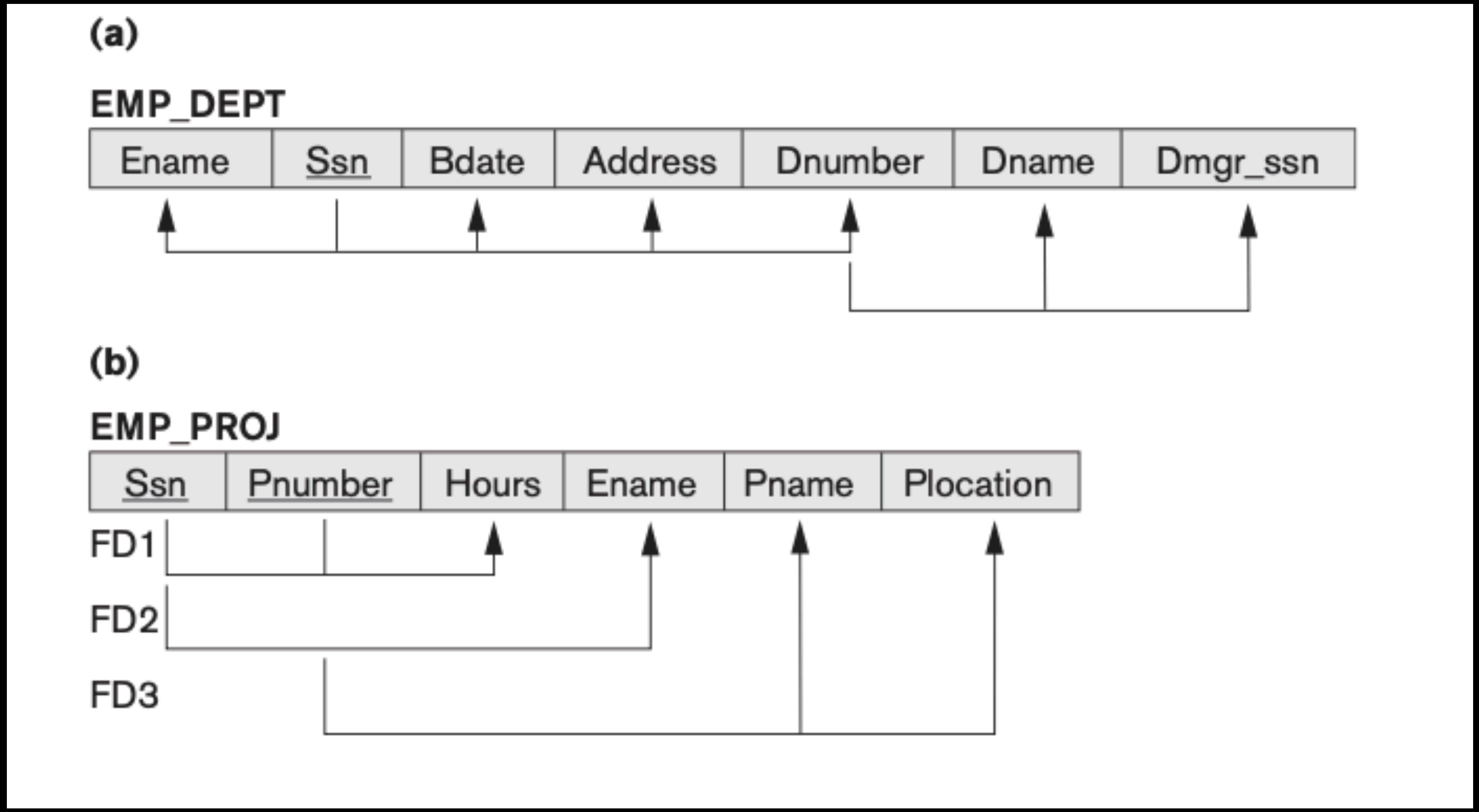




Informal Design Guidelines for Relation Schemas

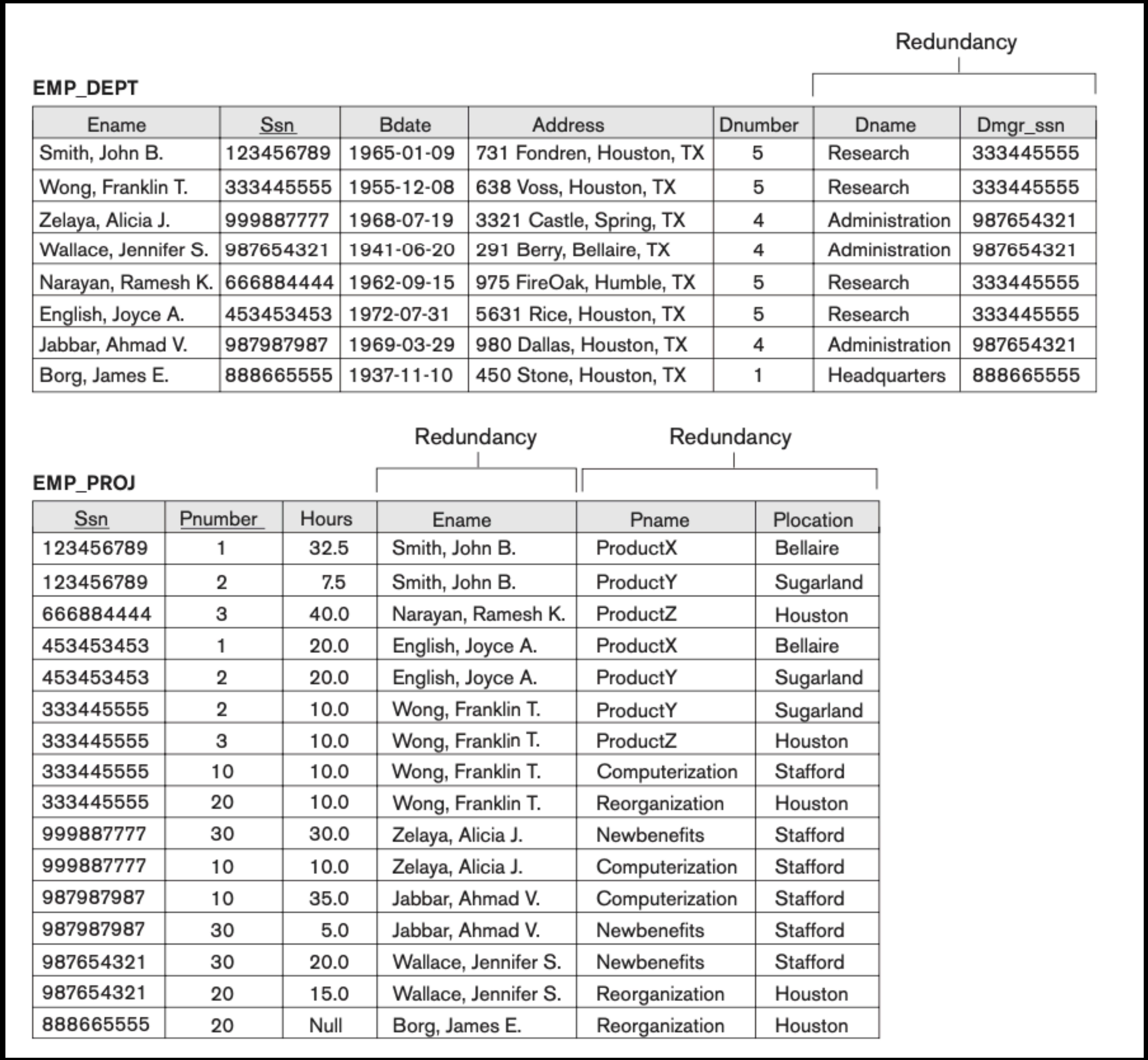
2. Redundant Information in Tuples and Update Anomalies

- *Guideline 2:* Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations. If any anomalies are present, note them clearly and make sure that the programs that update will operate correctly



Informal Design Guidelines for Relation Schemas

1. Insertion Anomalies
2. Deletion Anomalies
3. Modification Anomalies


Redundancy
Redundancy
Redundancy

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 FireOak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation
123456789	1	32.5	Smith, John B.	ProductX	Bellaire
123456789	2	7.5	Smith, John B.	ProductY	Sugarland
666884444	3	40.0	Narayan, Ramesh K.	ProductZ	Houston
453453453	1	20.0	English, Joyce A.	ProductX	Bellaire
453453453	2	20.0	English, Joyce A.	ProductY	Sugarland
333445555	2	10.0	Wong, Franklin T.	ProductY	Sugarland
333445555	3	10.0	Wong, Franklin T.	ProductZ	Houston
333445555	10	10.0	Wong, Franklin T.	Computerization	Stafford
333445555	20	10.0	Wong, Franklin T.	Reorganization	Houston
999887777	30	30.0	Zelaya, Alicia J.	Newbenefits	Stafford
999887777	10	10.0	Zelaya, Alicia J.	Computerization	Stafford
987987987	10	35.0	Jabbar, Ahmad V.	Computerization	Stafford
987987987	30	5.0	Jabbar, Ahmad V.	Newbenefits	Stafford
987654321	30	20.0	Wallace, Jennifer S.	Newbenefits	Stafford
987654321	20	15.0	Wallace, Jennifer S.	Reorganization	Houston
888665555	20	Null	Borg, James E.	Reorganization	Houston

Informal Design Guidelines for Relation Schemas

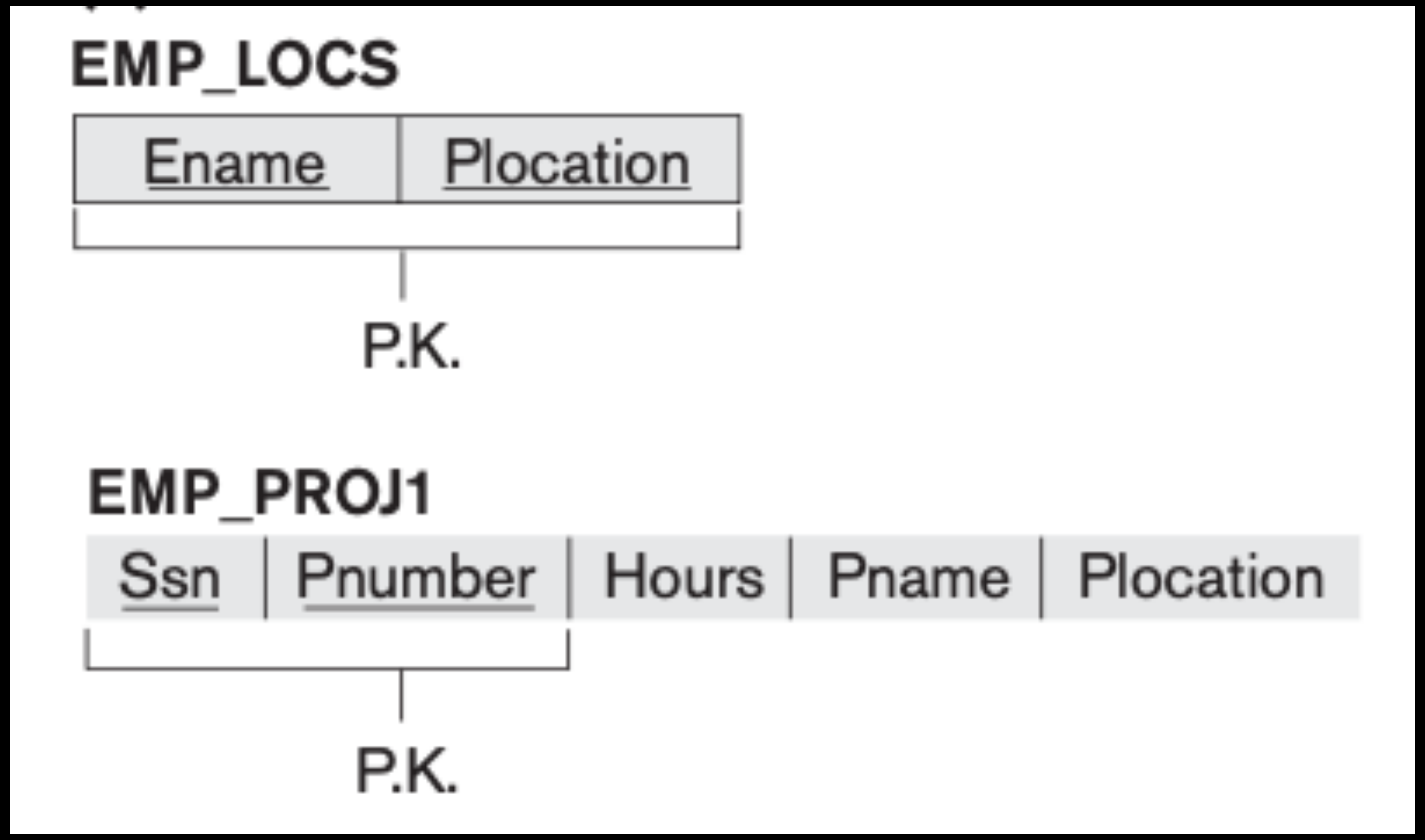
3. NULL Values in Tuples

   - *Guideline 3:* As far as possible, avoid placing attributes in a base relation whose values may frequently be NULL. If NULLs are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation.
     - The attribute *does not apply* to this tuple. For example, *Visa_status* may not apply to U.S. students.
     - The attribute value for this tuple is *unknown*. For example, the *Date_of_birth* may be unknown for an employee.
     - The value is *known but absent*; that is, it has not been recorded yet. For exam- ple, the *Home_Phone_Number* for an employee may exist, but may not be available and recorded yet.

Informal Design Guidelines for Relation Schemas

4. Generation of Spurious Tuples
   - *Guideline 4:* Design relation schemas so that they can be joined with equality conditions on attributes that are appropriately related (primary key, foreign key) pairs in a way that guarantees that no spurious tuples are generated. Avoid relations that contains matching attributes that are not (foreign key, primary key) combinations because joining on such attributes may produce spurious tuples.

Informal Design Guidelines for Relation Schemas



EMP_LOCS

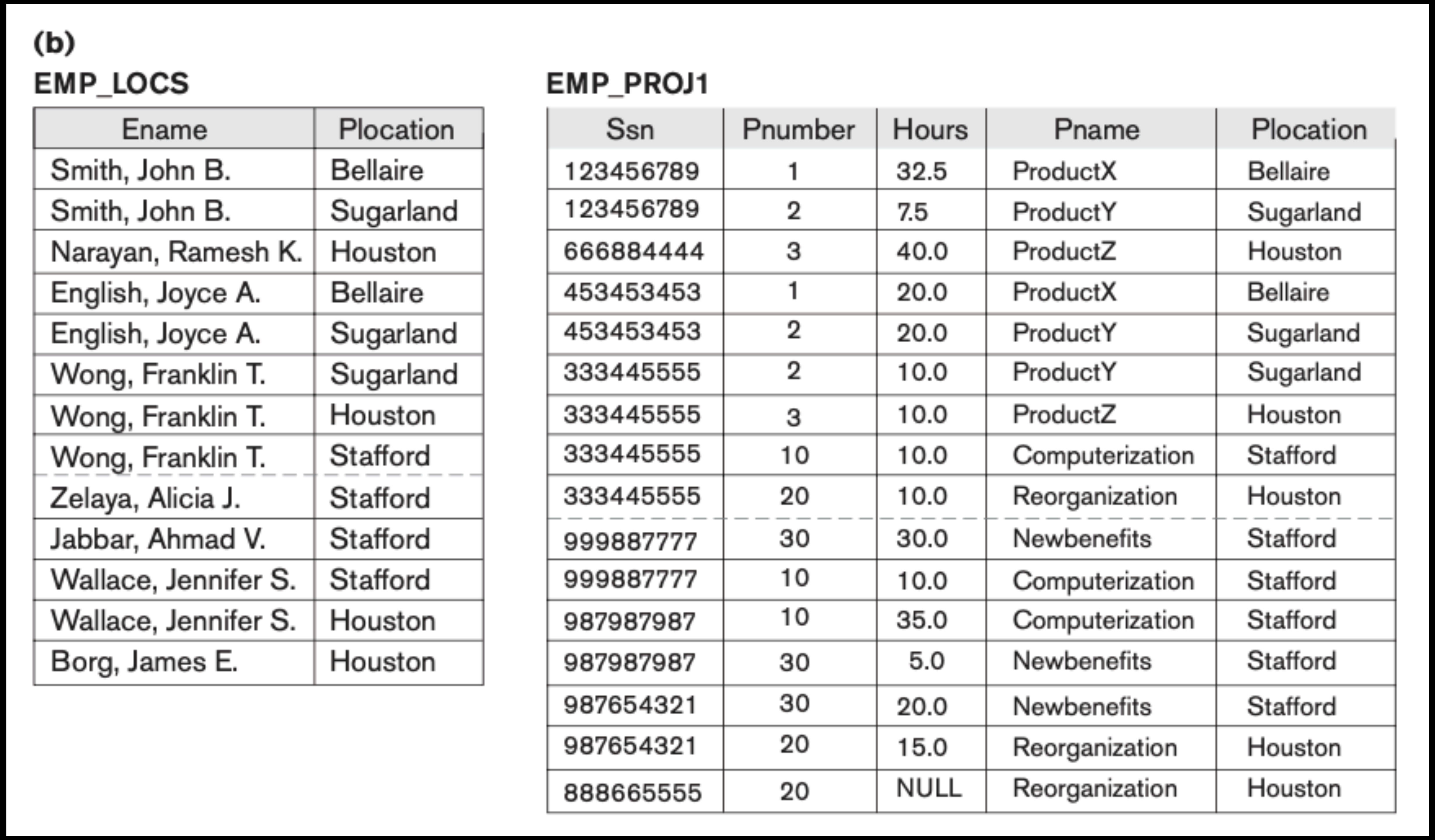
Ename	Plocation

P.K.

EMP_PROJ1

Ssn	Pnumber	Hours	Pname	Plocation

P.K.



(b)

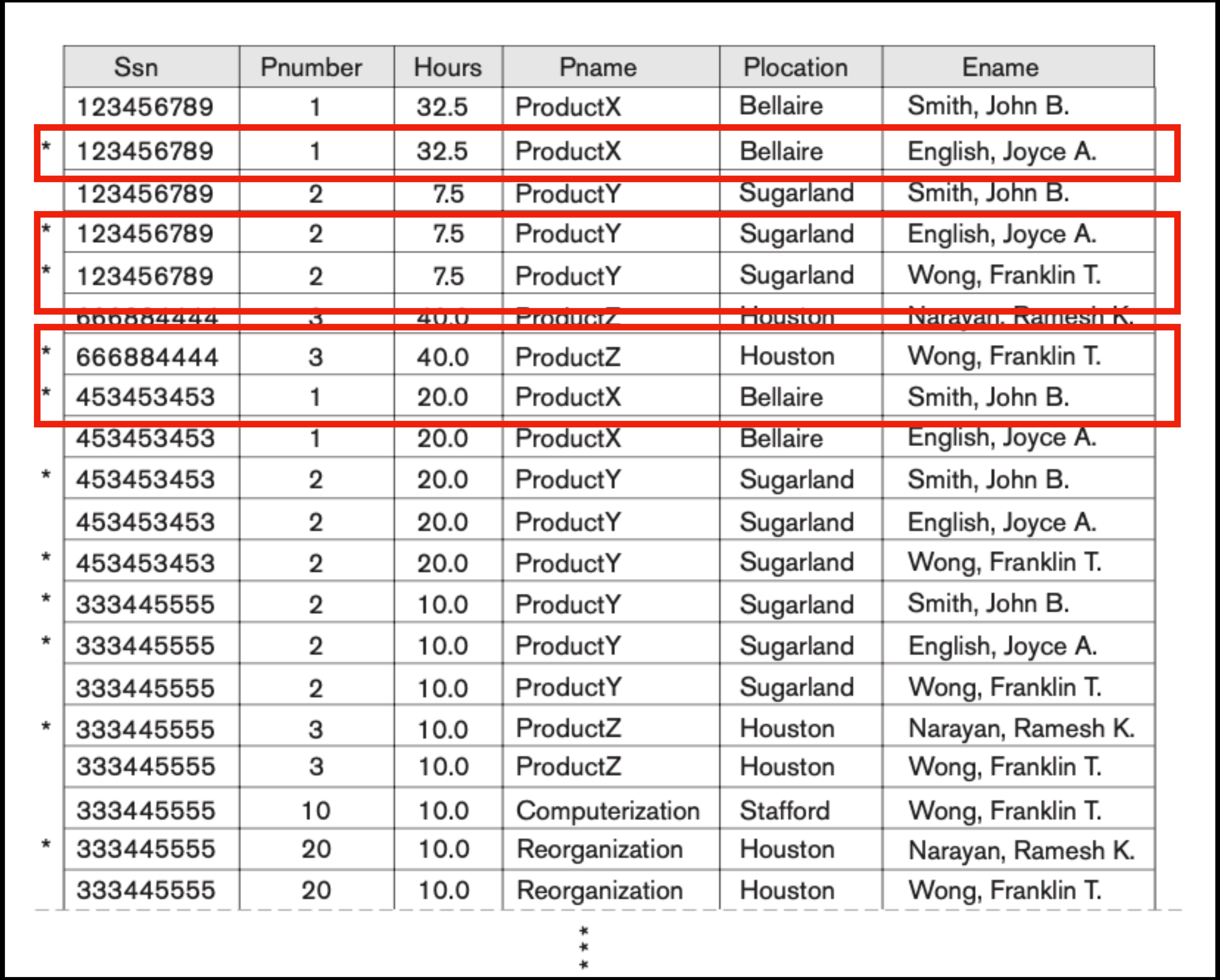
EMP_LOCS

Ename	Plocation
Smith, John B.	Bellaire
Smith, John B.	Sugarland
Narayan, Ramesh K.	Houston
English, Joyce A.	Bellaire
English, Joyce A.	Sugarland
Wong, Franklin T.	Sugarland
Wong, Franklin T.	Houston
Wong, Franklin T.	Stafford
Zelaya, Alicia J.	Stafford
Jabbar, Ahmad V.	Stafford
Wallace, Jennifer S.	Stafford
Wallace, Jennifer S.	Houston
Borg, James E.	Houston

EMP_PROJ1

Ssn	Pnumber	Hours	Pname	Plocation
123456789	1	32.5	ProductX	Bellaire
123456789	2	7.5	ProductY	Sugarland
666884444	3	40.0	ProductZ	Houston
453453453	1	20.0	ProductX	Bellaire
453453453	2	20.0	ProductY	Sugarland
333445555	2	10.0	ProductY	Sugarland
333445555	3	10.0	ProductZ	Houston
333445555	10	10.0	Computerization	Stafford
333445555	20	10.0	Reorganization	Houston
999887777	30	30.0	Newbenefits	Stafford
999887777	10	10.0	Computerization	Stafford
987987987	10	35.0	Computerization	Stafford
987987987	30	5.0	Newbenefits	Stafford
987654321	30	20.0	Newbenefits	Stafford
987654321	20	15.0	Reorganization	Houston
888665555	20	NULL	Reorganization	Houston

EMP_PROJ1 Natural Join EMP_LOCS



	Ssn	Pnumber	Hours	Pname	Plocation	Ename
	123456789	1	32.5	ProductX	Bellaire	Smith, John B.
*	123456789	1	32.5	ProductX	Bellaire	English, Joyce A.
	123456789	2	7.5	ProductY	Sugarland	Smith, John B.
*	123456789	2	7.5	ProductY	Sugarland	English, Joyce A.
*	123456789	2	7.5	ProductY	Sugarland	Wong, Franklin T.
	666884444	3	40.0	ProductZ	Houston	Narayan, Ramesh K.
*	666884444	3	40.0	ProductZ	Houston	Wong, Franklin T.
*	453453453	1	20.0	ProductX	Bellaire	Smith, John B.
	453453453	1	20.0	ProductX	Bellaire	English, Joyce A.
*	453453453	2	20.0	ProductY	Sugarland	Smith, John B.
	453453453	2	20.0	ProductY	Sugarland	English, Joyce A.
*	453453453	2	20.0	ProductY	Sugarland	Wong, Franklin T.
*	333445555	2	10.0	ProductY	Sugarland	Smith, John B.
*	333445555	2	10.0	ProductY	Sugarland	English, Joyce A.
	333445555	2	10.0	ProductY	Sugarland	Wong, Franklin T.
*	333445555	3	10.0	ProductZ	Houston	Narayan, Ramesh K.
	333445555	3	10.0	ProductZ	Houston	Wong, Franklin T.
	333445555	10	10.0	Computerization	Stafford	Wong, Franklin T.
*	333445555	20	10.0	Reorganization	Houston	Narayan, Ramesh K.
	333445555	20	10.0	Reorganization	Houston	Wong, Franklin T.

⋮

Functional Dependency

- Functional dependencies are used in the design (or re-design) of relational database to help eliminate redundancy(data duplication), therefore reducing the possibility of update anomalies.

- Redundancy is eliminated through a process called normalization.

- If a database schema is properly normalized, the following should hold true for all tables:

    - ★ *all columns should be functionally dependent on the table's primary key.*

- Functional dependencies allow you to verify that this is true and if it is not, determine the steps to take to normalize your tables so that it will be true (*without losing any data or losing any connections between data that is related*).

Functional Dependency

- A set of attributes Y is functionally dependent on a set of attributes X ($X \rightarrow Y$) when Y can be uniquely determined by X. This can be read as "X functionally determines Y", or as "Y is functionally determined by X".

- Note that the converse, $Y \rightarrow X$, is not necessarily true.

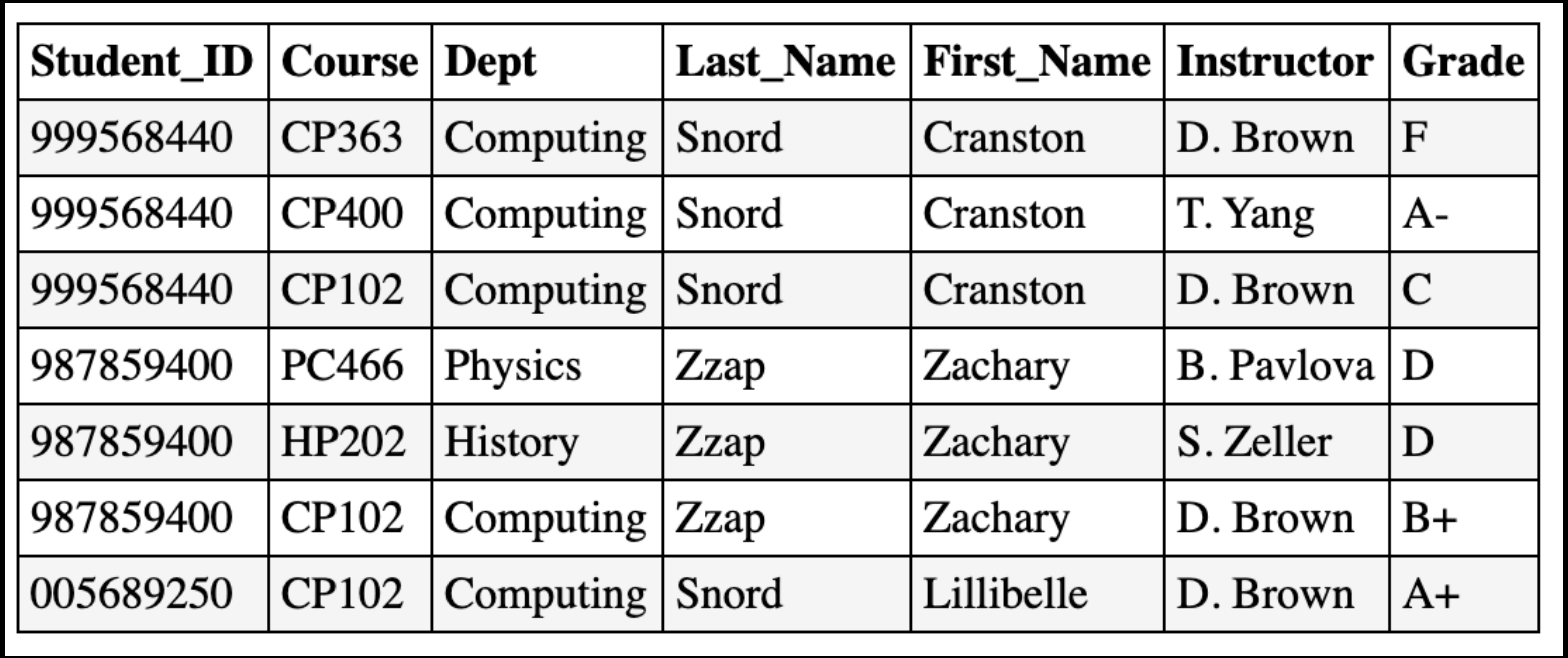
- For example, although **{Student_ID,Course}→Grade**, (i.e. a student in a given course may have only one grade, for example an 'A'), other students may also have an 'A' in the same course. It simply means that a particular student in a particular course has one and only one grade.

- clearly **Grade→{Student_ID,Course}** is not true.

Functional Dependency

- A functional dependency is a type of constraint on attributes that arises out of the meaning of those attributes.
- In other words, in any given tuple the value of one set of attributes depends on the value of another set of attributes.
- These depend on the *semantics*, or rules, underlying the relation in question.
- The previous example, **{Student_ID,Course}→Grade**, could turn out to be false if a student is allowed to take the same course in a different term.
- Then you could infer that the functional dependency should be **{Student_ID,Course,Term}→Grade** instead.

Functional Dependency

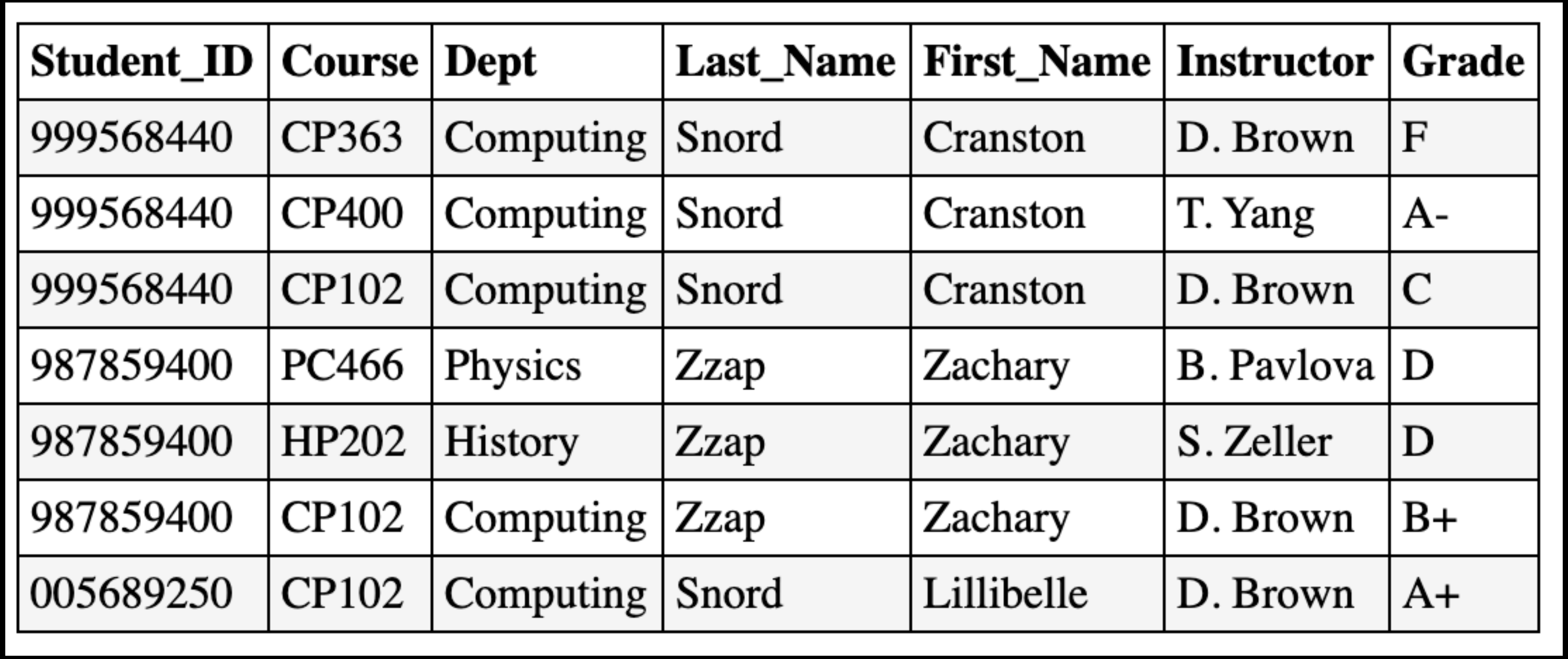


Student_ID	Course	Dept	Last_Name	First_Name	Instructor	Grade
999568440	CP363	Computing	Snord	Cranston	D. Brown	F
999568440	CP400	Computing	Snord	Cranston	T. Yang	A-
999568440	CP102	Computing	Snord	Cranston	D. Brown	C
987859400	PC466	Physics	Zzap	Zachary	B. Pavlova	D
987859400	HP202	History	Zzap	Zachary	S. Zeller	D
987859400	CP102	Computing	Zzap	Zachary	D. Brown	B+
005689250	CP102	Computing	Snord	Lillibelle	D. Brown	A+

- **{Last_Name,Course}→Grade**

★ is false, as there are multiple occurrences of a combination of the same last name and course name.

Functional Dependency



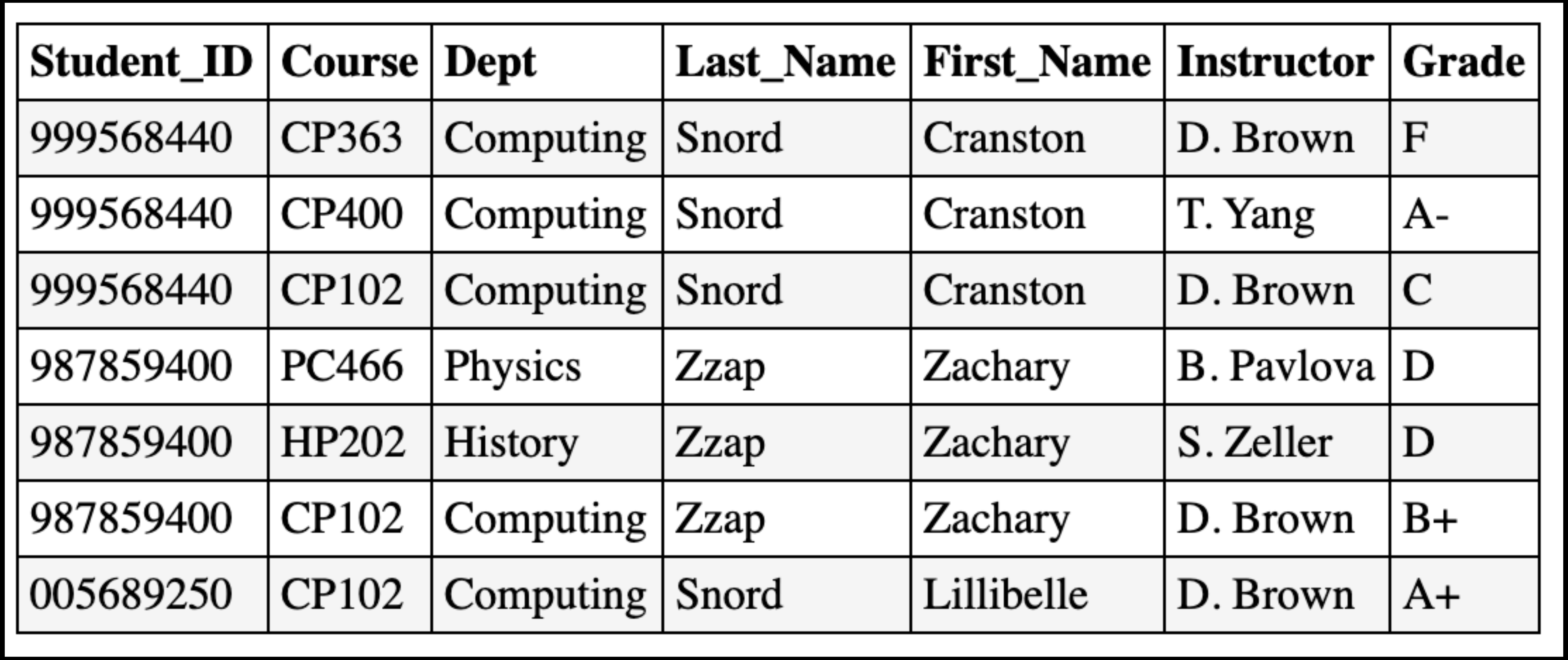
Student_ID	Course	Dept	Last_Name	First_Name	Instructor	Grade
999568440	CP363	Computing	Snord	Cranston	D. Brown	F
999568440	CP400	Computing	Snord	Cranston	T. Yang	A-
999568440	CP102	Computing	Snord	Cranston	D. Brown	C
987859400	PC466	Physics	Zzap	Zachary	B. Pavlova	D
987859400	HP202	History	Zzap	Zachary	S. Zeller	D
987859400	CP102	Computing	Zzap	Zachary	D. Brown	B+
005689250	CP102	Computing	Snord	Lillibelle	D. Brown	A+

- **Student_ID→{Last_Name,First_Name}**

★ is true, and there is no reason to doubt that the semantics behind this matches our observation.

★ It follows from this that both **Student_ID→Last_Name** and **Student_ID→First_Name** are true by the *decomposition* rule.

Functional Dependency

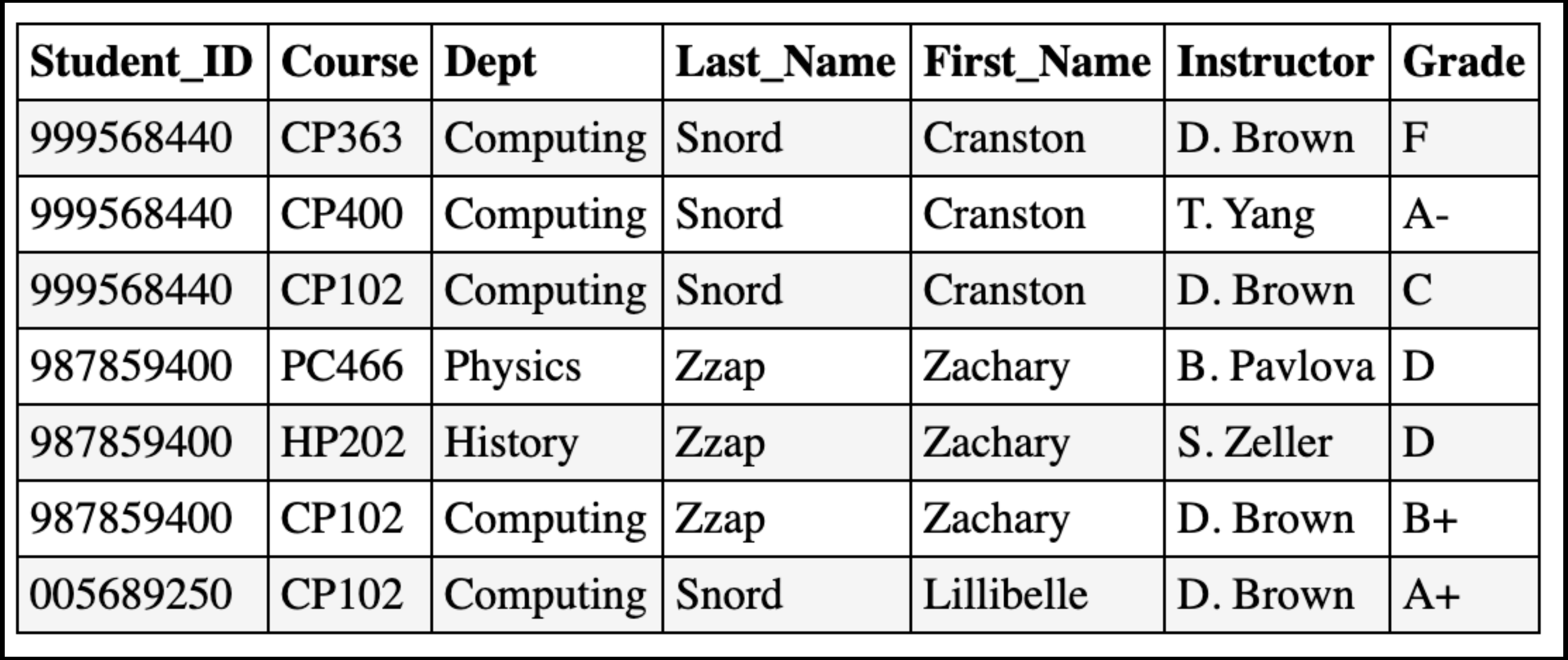


Student_ID	Course	Dept	Last_Name	First_Name	Instructor	Grade
999568440	CP363	Computing	Snord	Cranston	D. Brown	F
999568440	CP400	Computing	Snord	Cranston	T. Yang	A-
999568440	CP102	Computing	Snord	Cranston	D. Brown	C
987859400	PC466	Physics	Zzap	Zachary	B. Pavlova	D
987859400	HP202	History	Zzap	Zachary	S. Zeller	D
987859400	CP102	Computing	Zzap	Zachary	D. Brown	B+
005689250	CP102	Computing	Snord	Lillibelle	D. Brown	A+

- **Instructor→Course:**

★ is false, as clearly one instructor is involved in many courses.

Functional Dependency

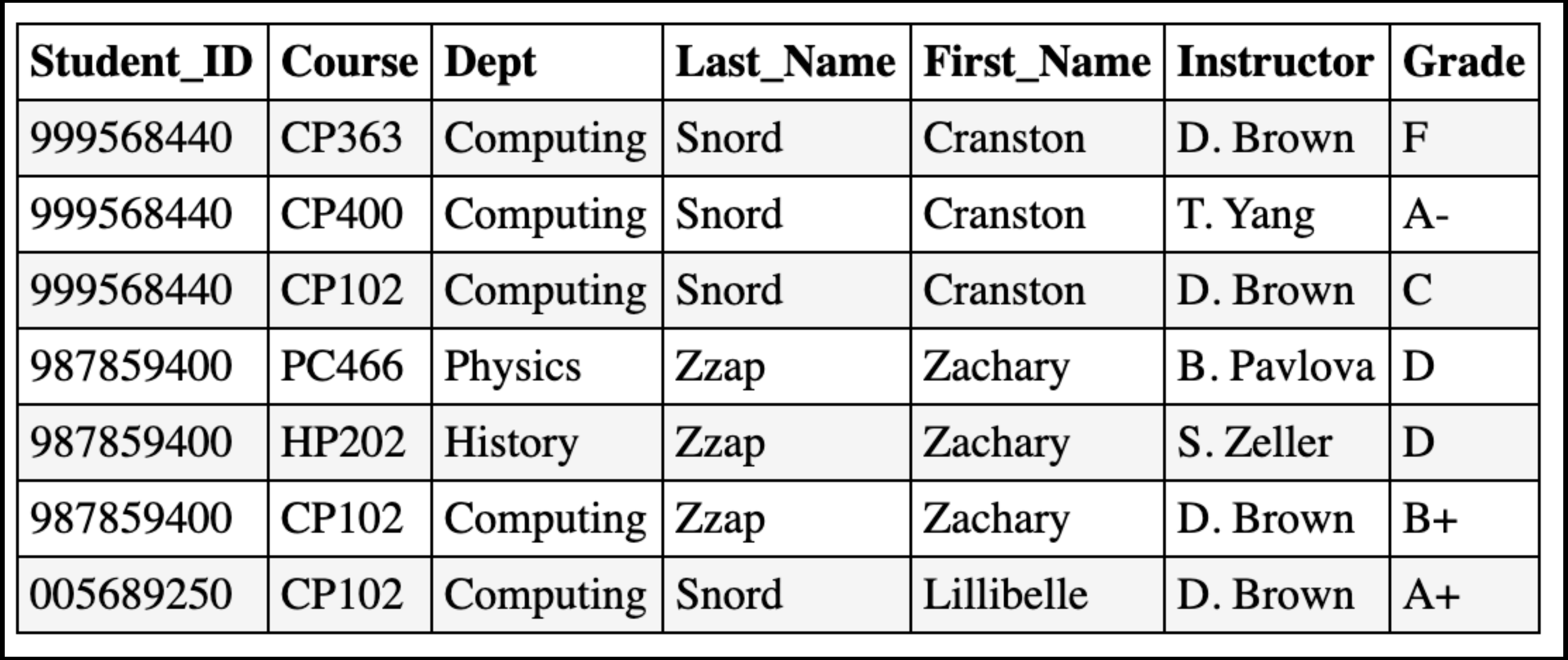


Student_ID	Course	Dept	Last_Name	First_Name	Instructor	Grade
999568440	CP363	Computing	Snord	Cranston	D. Brown	F
999568440	CP400	Computing	Snord	Cranston	T. Yang	A-
999568440	CP102	Computing	Snord	Cranston	D. Brown	C
987859400	PC466	Physics	Zzap	Zachary	B. Pavlova	D
987859400	HP202	History	Zzap	Zachary	S. Zeller	D
987859400	CP102	Computing	Zzap	Zachary	D. Brown	B+
005689250	CP102	Computing	Snord	Lillibelle	D. Brown	A+

- **Course→Instructor:**

★ may be true, as each value of **Course** determines only one value of **Instructor**. However, if the semantics of the situation claimed that a course may have multiple instructors then this FD would be false, despite the fact that the data we can see supports it. Without knowing the semantics we cannot be *sure* of an FD.

Functional Dependency

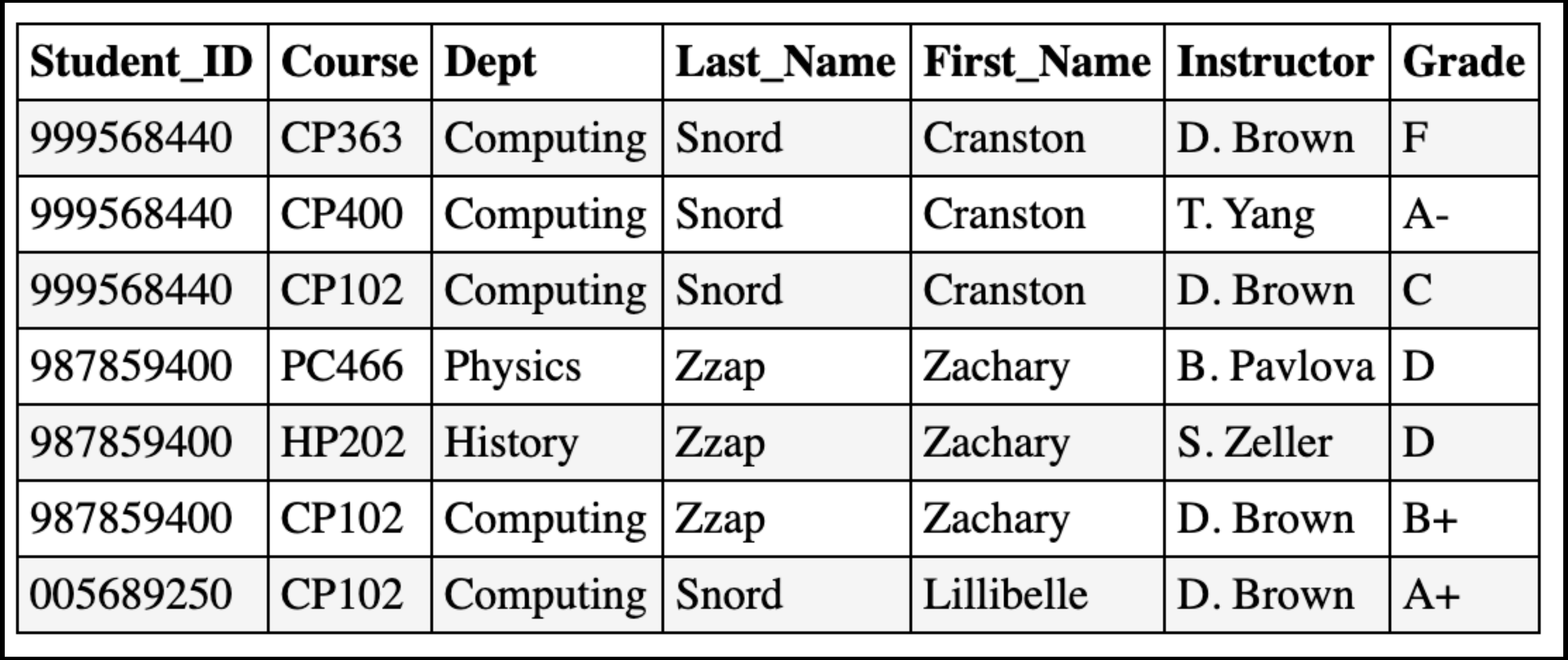


Student_ID	Course	Dept	Last_Name	First_Name	Instructor	Grade
999568440	CP363	Computing	Snord	Cranston	D. Brown	F
999568440	CP400	Computing	Snord	Cranston	T. Yang	A-
999568440	CP102	Computing	Snord	Cranston	D. Brown	C
987859400	PC466	Physics	Zzap	Zachary	B. Pavlova	D
987859400	HP202	History	Zzap	Zachary	S. Zeller	D
987859400	CP102	Computing	Zzap	Zachary	D. Brown	B+
005689250	CP102	Computing	Snord	Lillibelle	D. Brown	A+

- **Instructor→Dept:**

★ may be true, if the semantics are that an instructor may belong to only one department.

Functional Dependency



Student_ID	Course	Dept	Last_Name	First_Name	Instructor	Grade
999568440	CP363	Computing	Snord	Cranston	D. Brown	F
999568440	CP400	Computing	Snord	Cranston	T. Yang	A-
999568440	CP102	Computing	Snord	Cranston	D. Brown	C
987859400	PC466	Physics	Zzap	Zachary	B. Pavlova	D
987859400	HP202	History	Zzap	Zachary	S. Zeller	D
987859400	CP102	Computing	Zzap	Zachary	D. Brown	B+
005689250	CP102	Computing	Snord	Lillibelle	D. Brown	A+

- **Course→Dept**

⋆ is true, as a course is offered by only one department.

⋆ It could also be claimed that if both **Course→Instructor** and **Instructor→Dept** are true then **Course→Dept** *must* be true by the *transitive* properties of functional dependencies.

Functional Dependency

- Problematic issue:
  - ★ Representing set of ALL FD’s for a relation R.
- Solution
  - ★ Find a basic set of FD’s
  - ★ Use axioms for inferring
  - ★ Represent the set of all FD’s as the set of FD’s that can be inferred from a basic set of FDs

Inference Rules for Functional Dependencies

IR1: REFLEXIVITY rule.

If X is a set of attributes and $Y \subseteq X$, then $X \rightarrow Y$ holds.

IR2: AUGMENTATION rule.

If $X \rightarrow Y$ holds and Z is a set of attributes, then $XZ \rightarrow YZ$ holds.

IR3: TRANSITIVITY rule.

If $X \rightarrow Y$ holds and $Y \rightarrow Z$ holds, then $X \rightarrow Z$ holds.

IR4: UNION, or ADDITIVE rule.

If $X \rightarrow Y$ holds and $Y \rightarrow Z$ holds, then $X \rightarrow YZ$ holds.

IR5: DECOMPOSITION or PROJECTIVE rule.

If $X \rightarrow YZ$ holds, then $X \rightarrow Y$ AND $X \rightarrow Z$ holds.

IR6: PSEUDO TRANSITIVITY rule.

If $X \rightarrow Y$ holds, $YZ \rightarrow W$ then $XZ \rightarrow W$ holds.

Inference Rules for Functional Dependencies

- Inference rules IR1 through IR3 are sound and complete.
- By **sound**, we mean that given a set of functional dependencies *F* specified on a relation schema *R*, any dependency that we can infer from *F* by using IR1 through IR3 holds in every relation state *r* of *R* that *satisfies the dependencies* in *F*.
- By **complete**, we mean that using IR1 through IR3 repeatedly to infer dependencies until no more dependencies can be inferred results in the complete set of *all possible dependencies* that can be inferred from *F*.
- In other words, the set of dependencies *F*+, which we called the **closure** of *F*, can be determined from *F* by using only inference rules IR1 through IR3.
- Inference rules IR1 through IR3 are known as Armstrong's Inference rules.

Deriving Functional Dependency

- Derive **AB → CD** from the following FDs

AB → E
BE → I
E → C
CI → D

1. AB → E	[Given]
2. E → C	[Given]
3. AB → C	**[TRANSITIVITY 1,2]**
4. AB → B	**[REFLEXIVITY]**
5. AB → BE	**[UNION 1,4]**
6. BE → I	[Given]
7. AB → I	**[TRANSITIVITY 5,6]**
8. AB → CI	**[UNION 3,7]**
9. CI → D	[Given]
10. AB → D	**[TRANSITIVITY 8,9]**
11. AB → CD	**[AB → C, AB → D UNION]**

Closure set of FD's

- Formally, the set of all dependencies that include F as well as all dependencies that can be inferred from F is called the closure of F; it is denoted by F^+.

- Functional Dependency set or FD set of a relation is the set of all FDs present.

```
1. $F+=F$
2. repeat
    2.1. for each functional dependency $f$ in $F+$
        2.1.1. apply reflexivity and augmentation rules on $f$
        2.1.2. add the resulting functional dependencies to $F+$
    2.2. for each pair of functional dependencies $f_1$ and $f_2$ in $F+$
        2.2.1. if $f_1$ and $f_2$ can be combined using transitivity
                add the resulting functional dependency to $F+$
3. until $F+$ does not change any further
```

Initialize F^+ to the given FDs

Derive the FDs using Axioms

Repeat Step 2 until all the FDs are added to closure(F^+)

Closure of FD Example

- Consider a relation R (A,B,C,H) with the following functional dependencies:

  A -> B
  A -> C
  B -> H

Closure of FD for the given relation is

F^+ = { A -> B,
A -> C,
B -> H,
A ->H [TRANSITIVITY]
A -> BC [UNION]
A -> BH [UNION]
}

Equivalence of Sets of Functional Dependencies

- *Definition 1*: A set of functional dependencies F is said to ***cover*** another set of functional dependencies E if every FD in E is also in F^+; that is, if every dependency in E can be inferred from F; alternatively, we can say that E is covered by F.

- *Definition 2*: Two sets of functional dependencies E and F are ***equivalent*** if $E^+ = F^+$.

- Hence equivalence means that every FD in E can be inferred from F, and every FD in F can be inferred from E; that is, *E is equivalent to F if both the conditions E covers F and F covers E hold*

Closure of an Attribute

- To find the candidate key of the given relation.
- To find the super key of the given relation.

Closure of an Attribute

- If A is an attribute then the closure of A is represented as **A^+**
- Represents all the possible attributes that can be derived from A in terms of given Functional Dependencies.
- X -is a set of attributes

Add the attributes which are present on Left Hand Side in the original functional dependency.

```
1. $X^+ = X$;
2. repeat
    2.1. old$X^+$ := $X^+$
    2.2. for each FD $Y \rightarrow Z$ in S do
        2.2.1. if Y is subset of $X^+$ then $X^+ := X^+ \cup Z$;
until ($X^+$ =old $X^+$) /* If $X^+$ did not change then leave loop*/
```

add the attributes present on the Right Hand Side of the functional dependency

process until all the possible attributes which can be derived are added in the closure.

Closure of an Attribute

Step-1 : Add the attributes which are present on *Left Hand Side* in the original functional dependency.

Step-2 : Now, add the attributes present on the *Right Hand Side* of the functional dependency.

Step-3 : With the help of attributes present on Right Hand Side, check the other attributes that can be derived from the other given functional dependencies.

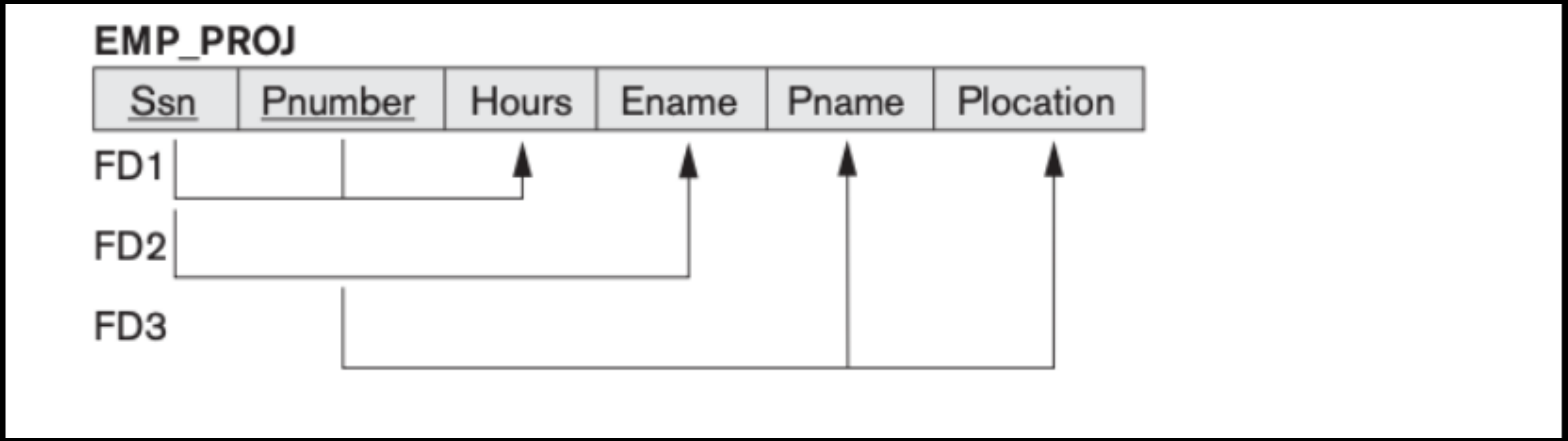
Step 4: Repeat this process until all the possible attributes which can be derived are added in the closure.

Closure of an Attribute - Eg

Consider the following functional dependencies related to the schema given below

FD1 : {SSN, Pnumber} → Hours
FD2 : SSN → Ename ,
FD3 : Pnumber → {Pname, Plocation}

Find the attribute closure?



Closure of an Attribute Example

Step-1 : Add attributes present on the LHS of the first functional dependency to the closure.

$\{SSN\}^+ = \{SSN\}$

Step-2 : Add attributes present on the RHS of the original functional dependency to the closure.

$\{SSN\}^+ = \{SSN, Ename\}$

Step-3 : Add the other possible attributes which can be derived using attributes present on the RHS of the closure.

Nothing is derived at this step

Therefore, complete closure of SSN will be :
$\{SSN\}^+ = \{SSN, Ename\}$

Closure of an Attribute Example

Step-1 : Add attributes present on the LHS of the first functional dependency to the closure.

$\{SSN, Pnumber\}^+ = \{SSN, Pnumber\}$

Step-2 : Add attributes present on the RHS of the original functional dependency to the closure.

$\{SSN, Pnumber\}^+ = \{SSN, Pnumber, Hours\}$

Step-3 : Add the other possible attributes which can be derived using attributes present on the RHS of the closure.

$\{SSN, Pnumber\}^+ = \{SSN, Pnumber, Hours, \boxed{Ename\}}$
$\{SSN, Pnumber\}^+ = \{SSN, Pnumber, Hours, Ename, \boxed{Pname, Plocation\}}$

Therefore, complete closure of SSN will be :
$\{SSN, Pnumber\}^+ = \{SSN, Pnumber, Hours, Ename, Pname, Plocation\}$

Closure of an Attribute

- Consider a relation R(A,B,C,D,E) having below mentioned functional dependencies.

FD1 : $A \rightarrow BC$

FD2 : $C \rightarrow B$

FD3 : $D \rightarrow E$

FD4 : $E \rightarrow D$

Calculate the closure of attributes of the relation?

$\{A\}^+ = \{A, B, C\}$

$\{B\}^+ = \{B\}$

$\{C\}^+ = \{B, C\}$

$\{D\}^+ = \{D, E\}$

$\{E\}^+ = \{E, D\}$

Find Super Key From Functional Dependencies

- Let R be the relational schema, and X be the set of attributes over R. If X^+ determine all the attributes of R, then X is said to be super key of R.

To Identify Super keys:

–Compute Closure for the attributes or combination of attributes on the LHS of Functional Dependency i.e. Determinants.

–If any closure includes all the attributes, then that can be declared as a key for the table.

EMP_PROJ (SSN, Pnumber, Hours, Ename, Pname, Plocation

$\{SSN, Pnumber\}^+ = \{SSN, Pnumber, Hours, Ename, Pname, Plocation\}$

Super Key

$\{SSN\}^+ = \{SSN, Ename\}$

Not a Super Key

Closure of an Attribute Calculating Candidate Key

- If there exists no subset of an attribute set whose closure contains all the attributes of the relation, then that attribute set is called as a candidate key of that relation.

- Consider the given functional dependencies-

$$\boxed{\begin{array}{c} AB \rightarrow CD \\ AF \rightarrow D \\ DE \rightarrow F \\ C \rightarrow G \\ F \rightarrow E \\ G \rightarrow A \end{array}}$$

- Which of the following options is false?

(A) $\{ CF \}^+ = \{ A , C , D , E , F , G \}$

(B) $\{ BG \}^+ = \{ A , B , C , D , G \}$

(C) $\{ AF \}^+ = \{ A , C , D , E , F , G \}$

(D) $\{ AB \}^+ = \{ A , C , D , F ,G \}$

Calculating Candidate Key -Example

- Consider the relation R(A,B,C) with given functional dependencies :

FD1 : $A \rightarrow B$

FD2 : $B \rightarrow C$

Now, calculating the closure of the attributes as :

$$\boxed{\begin{array}{l} \{A\}^+ = \{A, B, C\} \\ \{B\}^+ = \{B, C\} \\ \{C\}^+ = \{C\} \end{array}}$$

Clearly, “A” is the candidate key as, its closure contains all the attributes present in the relation “R”.

Calculating Candidate Key -Example

- Consider another relation R(A, B, C, D, E) having the Functional dependencies :

FD1 : $A \rightarrow BC$

FD2 : $C \rightarrow B$

FD3 : $D \rightarrow E$

FD4 : $E \rightarrow D$

Now, calculating the closure of the attributes as :

$\{A\}^+ = \{A, B, C\}$

$\{B\}^+ = \{B\}$

$\{C\}^+ = \{C, B\}$

$\{D\}^+ = \{E, D\}$

$\{E\}^+ = \{E, D\}$

In this case, a single attribute is unable to determine all the attribute on its own like in previous example. Here, we need to combine two or more attributes to determine the candidate keys.

$\{A, D\}^+ = \{A, B, C, D, E\}$
$\{A, E\}^+ = \{A, B, C, D, E\}$

Hence, "AD" and "AE" are the two possible keys of the given relation "R". Any other combination other than these two would have acted as extraneous attributes.

Closure Functional Dependencies - Key Definition

- **Prime Attributes:** Attributes which are indispensable part of candidate keys. For example : “A, D, E” attributes are prime attributes.
- **Non-Prime Attributes:** Attributes other than prime attributes which does not take part in formation of candidate keys.
- **Extraneous Attributes:** Attributes which does not make any effect on removal from candidate key.

Consider the relation R(A, B, C, D) with functional dependencies :
FD1 : A → BC
FD2 : B → C
FD3 : D → C

Candidate key : AD
Prime Attributes : A, D.
Non-Prime Attributes : B, C
Extraneous Attributes : B, C

Equivalence of FDs

- Two different sets of functional dependencies for a given relation may or may not be equivalent.

- If F and G are the two sets of functional dependencies, then following 3 cases are possible-

Case-01: F covers G ($F \supseteq G$)

Case-02: G covers F ($G \supseteq F$)

Case-03: Both F and G cover each other ($F = G$)

Determining whether F covers G

Step-01:

- Take the functional dependencies of set G into consideration.
- For each functional dependency X → Y, find the closure of X using the functional dependencies of set G.

Step-02:

- Take the functional dependencies of set G into consideration.
- For each functional dependency X → Y, find the closure of X using the functional dependencies of set F.

Step-03:

- Compare the results of Step-01 and Step-02.
- If the functional dependencies of set F has determined all those attributes that were determined by the functional dependencies of set G, then it means F covers G.
- Thus, we conclude F covers G ($F \supseteq G$) otherwise not.

PRACTICE PROBLEM BASED ON EQUIVALENCE OF FUNCTIONAL DEPENDENCIES

- A relation R (A , C , D , E , H) is having two functional dependencies sets F and G

SET F - {A → C, AC → D, E → AD, E → H}

SET G - {A → CD, E → AH}

Which of the following holds true?

(A) G ⊇ F

(B) F ⊇ G

(C) F = G

(D) All of the above

Determining whether F coversG

Step-01:

- $(A)^+ = \{ A , C , D \}$ // closure of left side of $A \rightarrow CD$ using set G
- $(E)^+ = \{ A , C , D , E , H \}$ // closure of left side of $E \rightarrow AH$ using set G

Step-02:

- $(A)^+ = \{ A , C , D \}$ // closure of left side of $A \rightarrow CD$ using set F
- $(E)^+ = \{ A , C , D , E , H \}$ // closure of left side of $E \rightarrow AH$ using set F

Step-03:

Comparing the results of Step-01 and Step-02,

- Functional dependencies of set F can determine all the attributes which have been determined by the functional dependencies of set G.
- Thus, we conclude F covers G i.e. $F \supseteq G$.

Determine G covers F

Step-01:

- $(A)^+ = \{ A , C , D \}$ // closure of left side of $A \rightarrow C$ using set F
- $(AC)^+ = \{ A , C , D \}$ // closure of left side of $AC \rightarrow D$ using set F
- $(E)^+ = \{ A , C , D , E , H \}$ // closure of left side of $E \rightarrow AD$ and $E \rightarrow H$ using set F

Step-02:

- $(A)^+ = \{ A , C , D \}$ // closure of left side of $A \rightarrow C$ using set G
- $(AC)^+ = \{ A , C , D \}$ // closure of left side of $AC \rightarrow D$ using set G
- $(E)^+ = \{ A , C , D , E , H \}$ // closure of left side of $E \rightarrow AD$ and $E \rightarrow H$ using set G

Step-03:

Comparing the results of Step-01 and Step-02, we find-

- Functional dependencies of set G can determine all the attributes which have been determined by the functional dependencies of set F.
- Thus, we conclude G covers F i.e. $G \supseteq F$.

Extraneous

- If we are able to remove an attribute from a functional dependency without changing the closure of the set of functional dependencies, that attribute is called as Extraneous Attribute.

- Consider a set of functional dependencies F, and the closure of set of functional dependencies F^+.

- If we remove an attribute from any of the FDs under F and find the closure of new set of functional dependencies. Let the new closure of set of functional dependencies be G^+. If F^+ equals the newly constituted closure G^+, then the attribute which has been removed is called as Extraneous Attribute.

Extraneous

- Test efficiently if an attribute is extraneous. Let R be the relation schema, and let F be the given set of functional dependencies that hold on R. Consider an attribute A in a dependency $\alpha \rightarrow \beta$.

Extraneous in RHS

Case 1: If $A \in \beta$, to check if A is extraneous in β,

1. Compute α^+ (closure of α) using only the dependencies in the set

$$F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$$

2. Check if $\alpha \rightarrow A$ can be inferred from F' ; if $\alpha+$ includes A, then A is extraneous in β .

Extraneous in LHS

Case 2: If $A \in \alpha$,to check if A is extraneous in α,

1. let $\gamma = \alpha - \{A\}$, and compute γ^+ using the dependencies in F; if $\gamma \rightarrow \beta$ can be inferred from F.
2. Check that if γ^+ includes all attributes in β , then A is extraneous in α.

Extraneous Example

- Example 1: Given F = {A → C, AB → C }
- B is extraneous in AB → C (Here α is AB and β is C)

1. $\gamma = \alpha - \{B\}$ now γ = AB - B = A
2. Compute A⁺ = {A, C}, The closure includes all attributes in RHS, therefore B is extraneous.

- Example 2: Given F = {A → C, AB → CD}
- C is extraneous in AB → CD

1. F′ = {A → C, AB → D}
2. Compute AB^+ = {A, B, C, D}, The closure includes C(which is now removed in F′) therefore C is extraneous.

Canonical Cover

- A canonical cover F_c for F is a set of functional dependencies such that:
    - ★ F logically implies all dependencies in F_c
    - ★ F_c logically implies all dependencies in F
    - ★ Can't infer any functional dependency in F_c from other dependencies in F_c
    - ★ No functional dependency in F_c contains an extraneous attribute
    - ★ Left side of all functional dependencies in F_c are unique
        - *There are no two dependencies $\alpha_1 \rightarrow \beta_1$ and $\alpha_2 \rightarrow \beta_2$ in Fc such that $\alpha_1 = \alpha_2$*

Canonical Cover

- A canonical cover is a simplified and reduced version of the given set of functional dependencies.
- Since it is a reduced version, it is also called as **Irreducible set**.

```
1. $F_c = F$
2. Repeat
        2.1. Use the union rule to replace any dependencies in $F_c$ of the form
             $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1\beta_2$.
        2.2. Find a functional dependency $\alpha \rightarrow \beta$ in $F_c$ with an extraneous
             attribute either in $\alpha$ or in $\beta$ .
           /* Note: the test for extraneous attributes is done using $F_c$ , not $F$ */
        2.3. If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$ in $F_c$ .
until ($F_c$ does not change)
```

Canonical Cover

- Algorithm to find minimal cover for a set of FDs F
- Step 1: Let G be the set of FDs obtained from F by *decomposing* the right hand sides of each FD to a single attribute.
- Step 2: Remove all *redundant attributes* from the left hand sides of FDs in G.
- Step 3: From the resulting set of FDs, *remove all redundant FDs*.
- Output the resulting set of FDs.

Canonical Cover

- **Characteristics**

★ Canonical cover is free from all the extraneous functional dependencies.

★ The closure of canonical cover is same as that of the given set of functional dependencies.

★ Canonical cover is not unique and may be more than one for a given set of functional dependencies.

- **Need**

★ Working with the set containing extraneous functional dependencies increases the computation time.

★ Therefore, the given set is reduced by eliminating the useless functional dependencies.

★ This reduces the computation time and working with the irreducible set becomes easier.

Computing Canonical Cover

- R = {A,B,C,D,E,F,G,H}
- F = {AC →G, D →EG, BC →D, CG →BD, ACD →B, CE →AG}
- Find the canonical cover of F?

$$\boxed{\begin{array}{l} AC \rightarrow G \\ D \rightarrow E \\ D \rightarrow G \\ BC \rightarrow D \\ CG \rightarrow B \\ CG \rightarrow D \\ ACD \rightarrow B \\ CE \rightarrow A \\ CE \rightarrow G \end{array}}$$

Computing Canonical Cover

- R = {A,B,C,D,E,F,G,H}
- F = {AC →G, D →EG, BC →D, CG →BD, ACD →B, CE →AG}
- Find the canonical cover of F?

AC → G
D → E ✓
D → G ✓
BC → D
CG → B
CG → D
ACD → B
CE → A
CE → G

Find the extraneous attribute in this FD:

D?
(AC)+ = ACG**B**, so we got **B**;
D is extraneous and can be safely eliminated.
Rewrite the new FD as AC B

Computing Canonical Cover

- R = {A,B,C,D,E,F,G,H}
- F = {AC →G, D →EG, BC →D, CG →BD, ACD →B, CE →AG}
- Find the canonical cover of F?

AC → G
D → E ✓
D → G ✓
BC → D
CG → B
CG → D
AC → B
CE → A
CE → G

Find the extraneous attribute in this FD:

A? C?
(A)+ = A, so can't get G; C is not extraneous;
(C)+ = C, so can't get G; A is not extraneous;
Keep this FD as is

Computing Canonical Cover

- R = {A,B,C,D,E,F,G,H}
- F = {AC →G, D →EG, BC →D, CG →BD, ACD →B, CE →AG}
- Find the canonical cover of F?

AC → G
D → E ✓
D → G ✓
BC → D
CG → B
CG → D
AC → B
CE → A
CE → G

Find the extraneous attribute in this FD:

B? C?
(B)+ = B, so can't get D; C is not extraneous;
(C)+ = C, so can't get D; A is not extraneous;

Keep this FD as is

Computing Canonical Cover

- R = {A,B,C,D,E,F,G,H}
- F = {AC →G, D →EG, BC →D, CG →BD, ACD →B, CE →AG}
- Find the canonical cover of F?

AC → G

D → E ✓

D → G ✓

BC → D

CG → B

CG → D

AC → B

CE → A

CE → G

Find the extraneous attribute in this FD:

G? C?
(C)+ = C, so can’t get B; G is not extraneous;
(G)+ = G, so can’t get B; C is not extraneous;

Keep this FD as is

Computing Canonical Cover

- R = {A,B,C,D,E,F,G,H}
- F = {AC →G, D →EG, BC →D, CG →BD, ACD →B, CE →AG}
- Find the canonical cover of F?

AC → G
D → E ✓
D → G ✓
BC → D
CG → B
CG → D
AC → B
CE → A
CE → G

Find the extraneous attribute in this FD:

G? C?
(C)+ = C, so can’t get D; G is not extraneous;
(G)+ = G, so can’t get D; C is not extraneous;

Keep this FD as is

Computing Canonical Cover

- R = {A,B,C,D,E,F,G,H}
- F = {AC →G, D →EG, BC →D, CG →BD, ACD →B, CE →AG}
- Find the canonical cover of F?

$AC \rightarrow G$
$D \rightarrow E$ ✓
$D \rightarrow G$ ✓
$BC \rightarrow D$
$CG \rightarrow B$
$CG \rightarrow D$
$AC \rightarrow B$
$CE \rightarrow A$
$CE \rightarrow G$

If we continue we will not find any extraneous attribute on LHS of any FD.

Computing Canonical Cover

- R = {A,B,C,D,E,F,G,H}
- F = {AC →G, D →EG, BC →D, CG →BD, ACD →B, CE →AG}
- Find the canonical cover of F?

✗ **AC → G**
D → E ✓
D → G ✓
BC → D
CG → B
CG → D
AC → B
CE → A
CE → G

Find the redundant FDs:

(AC)+ = ACBDE**G** ; so we got G from other FDs

Remove the entire FD from the list.

Computing Canonical Cover

- R = {A,B,C,D,E,F,G,H}
- F = {AC →G, D →EG, BC →D, CG →BD, ACD →B, CE →AG}
- Find the canonical cover of F?

✗ **AC → G**
D → E ✓
D → G ✓
BC → D
✗ **CG → B**
CG → D
AC → B
CE → A
CE → G

Find the redundant FDs:

(CG)+ = CGDEAB ; so we got B from other FDs

Remove the entire FD from the list.

Computing Canonical Cover

- R = {A,B,C,D,E,F,G,H}
- F = {AC →G, D →EG, BC →D, CG →BD, ACD →B, CE →AG}
- Find the canonical cover of F?

✗ **AC → G**

D → E ✓

D → G ✓

BC → D

✗ **CG → B**

CG → D

AC → B

CE → A

CE → G

Find the redundant FDs:

(CE)+ = CEGD ; so we could not get A from other FDs

Keep this FD in the list.

Computing Canonical Cover

- R = {A,B,C,D,E,F,G,H}
- F = {AC →G, D →EG, BC →D, CG →BD, ACD →B, CE →AG}
- Find the canonical cover of F?

✗ **AC → G**
D → E ✓
D → G ✓
BC → D
✗ **CG → B**
CG → D
AC → B
CE → A
✗ **CE → G**

Find the redundant FDs:

(CE)+ = CEABDG ; so we got G from other FDs

Remove the entire FD from the list.

Computing Canonical Cover

- R = {A,B,C,D,E,F,G,H}
- F = {AC →G, D →EG, BC →D, CG →BD, ACD →B, CE →AG}
- Find the canonical cover of F?

Apply union (if any) on the remaining FDs

D → EG ; so we got G from other FDs

The result is the canonical cover (F_c) of F

D → E
D → G
BC → D
CG → D
AC → B
CE → A

Different order of considering the extraneous attributes can result in different Fc

D → EG
BC → D
CG → D
AC → B
CE → A

Thank You