COMP9311: Database Systems

Term 3 2021

Week 7 Relational Database Design Theory

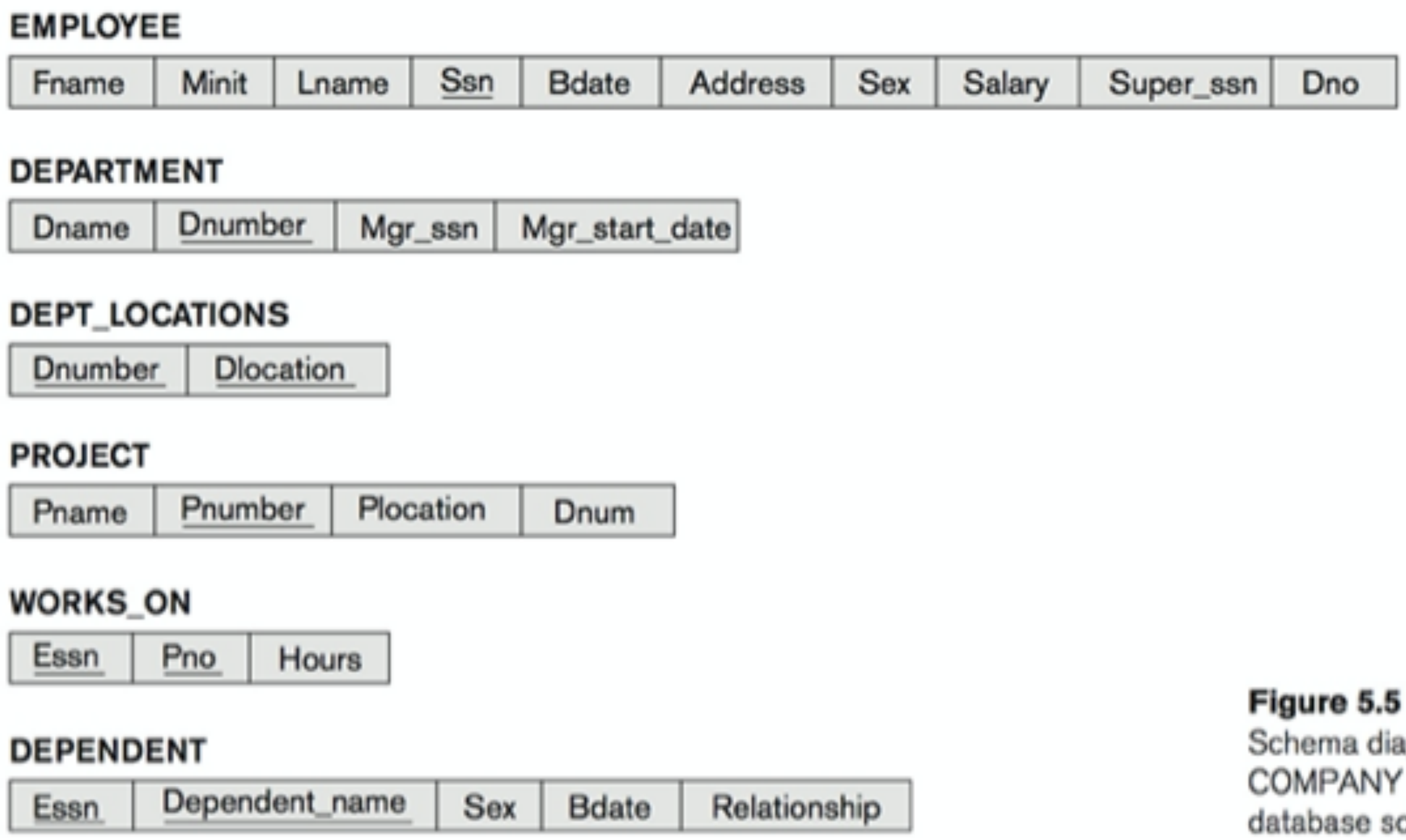
By Helen Paik, CSE UNSW

Textbook: Chapters 14 and 15

Disclaimer: the course materials are sourced from

- previous offerings of COMP9311 and COMP3311
- Prof. Werner Nutt on Introduction to Database Systems (http://www.inf.unibz.it/~nutt/Teaching/IDBs1011/)

Designing Relational Schemas



EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

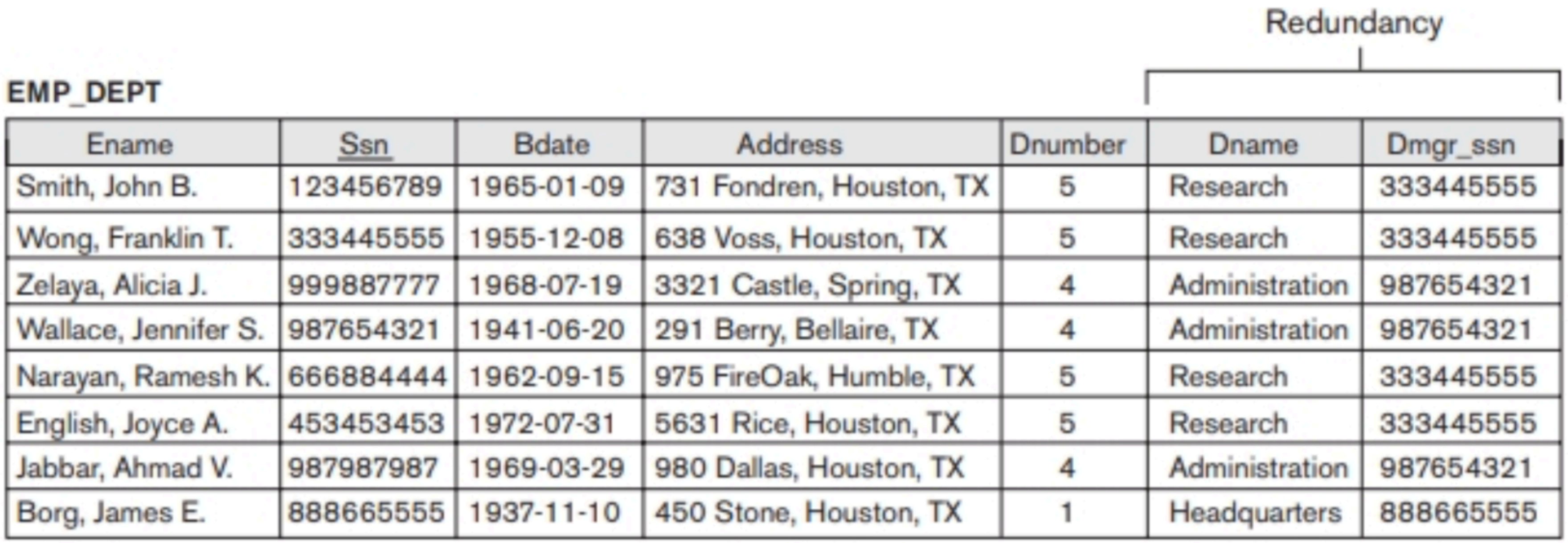
DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Figure 5.5
Schema diagram for the COMPANY relational database schema.

Relational Design and Redundancy

Consider the following relation defining Employee and Department


Redundancy

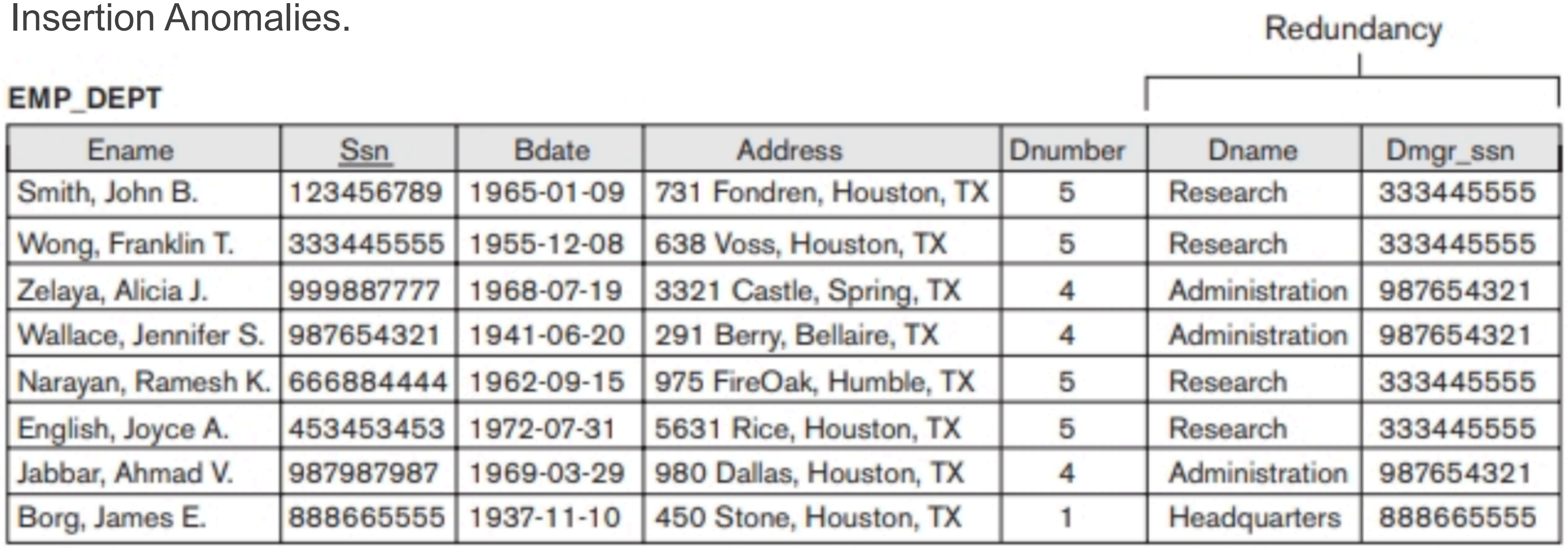
EMP_DEPT

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

We need to be careful updating this data, otherwise we may introduce inconsistencies.

Relational Design and Redundancy

Insertion Anomalies.



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

("Sue Smith", "98790098", "1968-09-10", "789 Captain, Houston, TX", 4, "Administration", "1155667788")

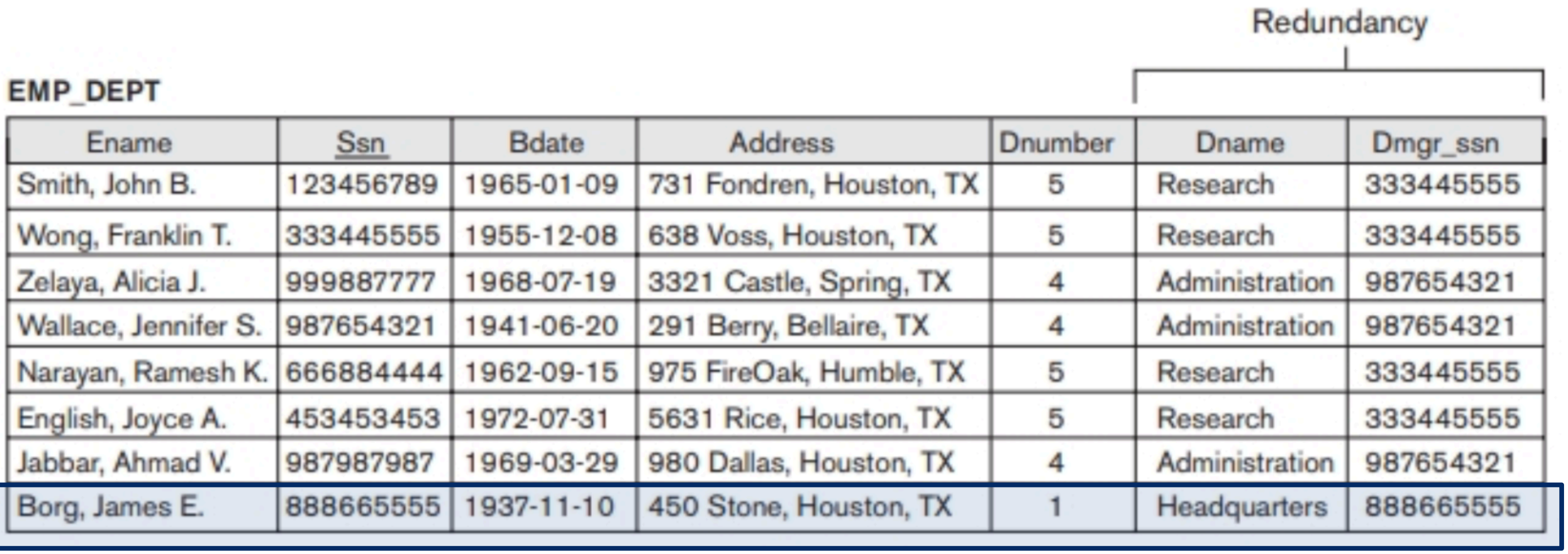
(NULL, NULL, NULL, NULL, 6, "Marketing", "123456789")

when we insert a new record:

- we need to check that department data is consistent with existing tuples
- we must include both employee and department details (or NULLs for not-known?)

Relational Design and Redundancy

Deletion anomaly (say delete employee James Borg)



Redundancy

EMP_DEPT

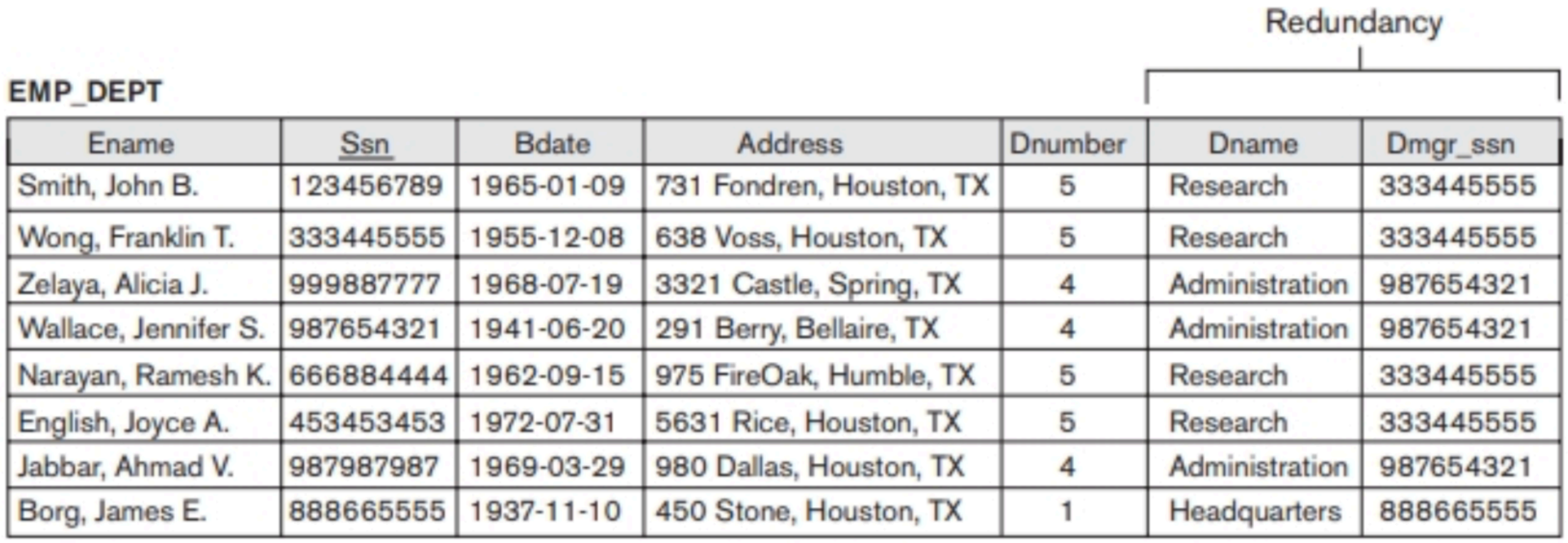
Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
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

What is the department number of Headquarters? Who is the manager?

if we remove information about the last employee at the department, all of the department information disappears

Relational Design and Redundancy

Update anomaly example (update the manager of Department 5)



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

if a branch changes address, we need to update all tuples referring to that branch

Relational (Database) Design Theory

Usually there are many different options in designing a database schema for an application … Which one to choose? How do we know which one is better than the other?

Previously we studied ER design and ER-to-relational mapping. We claimed that this allows us to produce "good" schemas. However, the mapping can also produce different relations, more over, some designers choose to go straight into relations.

Can we make a stronger/formal statement on what makes a schema good through some analysis?

The study of relational design theory

- **examines some foundational notions of "schema goodness"**
- **provides methods to transform schemas to make them better**

Relational (Database) Design Theory

Functional Dependency

Functional dependencies

- A functional dependency is a kind of **constraint** between two sets of attributes from the database.
- have implications for "good" relational schema design

What we study here:

- basic theory and definition of *functional dependencies*
- methodology for improving schema designs (***normalisation***)

The aim of studying this:

- improve understanding of relationships among data
- gain enough formalism to assist practical database design

Relational Design Theory

Our earlier approach used a top-down design procedure:

- structure data at conceptual level (ER design)
- then map a collection of tables

It appears that ER-design-then-relational-mapping

- leads to a collection of well-structured tables, so why do we need a dependency theory and normalisation procedure to deal with redundancy?

Some reasons ...

- ER design does not guarantee minimal redundancy … *dependency theory allows us to check designs for residual problems*
- *Tables can be created independently of any conceptual designs. You still may need to analyse "goodness" of schema of given tables*

Relational Design Theory

A *good* relational database design:

- must capture *all* of the necessary attributes/associations
- should do this with a *minimal* amount of stored information

Minimal stored information ⇒ no redundant data.

In database design, *redundancy* is generally a "bad thing":

- causes problems maintaining consistency after updates
- require extra storage

However, it can sometimes lead to performance improvements

e.g. may be able to avoid a join to collect bits of data together

Relational Design Theory

To avoid redundancy and update anomaly problems:

- *decompose* the relation U into several smaller relations R_i
- where each R_i has minimal overlap with other R_j
- Typically, each R_i contains information about one entity (e.g. employee, department, ...)

Design – revisited.

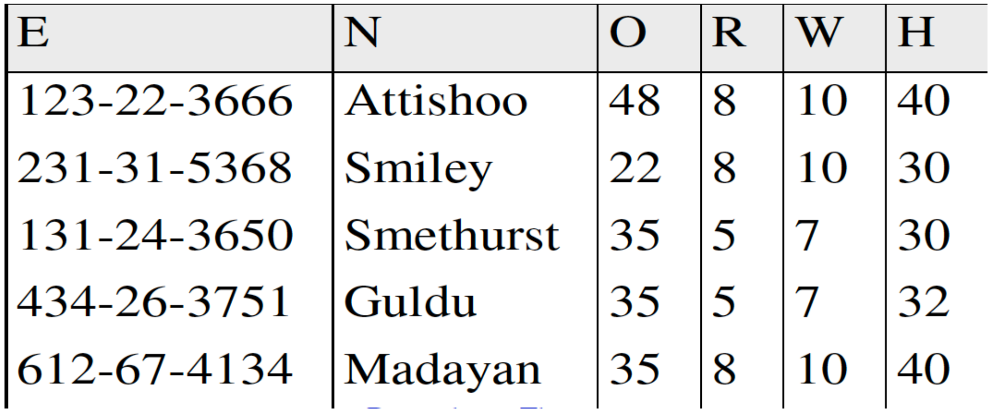
Redundancy is at the root of several problems associated with relational schemas:

- redundant storage, insert/delete/update anomalies

Consider relation obtained from Hourly_Emps:

Hourly_Emps (**E**id, **N**ame, **O**ffice, **R**ating, hrly_**W**ages, **H**rs_worked)

Let's say the hourly_**W**age is determined by rating.



E	N	O	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

- Update – can we update W for just the 1st tuple?
- Can we record the hourly wage for a rating that no employee has currently?
- If we delete an employee with rating 5, we also lose the information about its hourly rate …

i.e., we see the anomaly problems

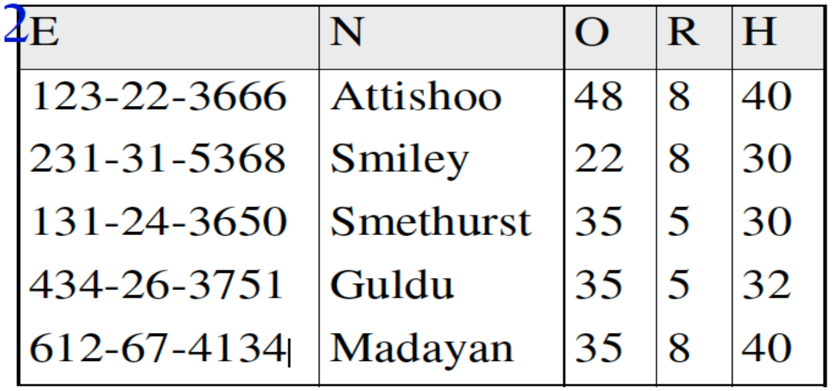
Decomposition – design refinement

Main refinement technique:

decomposition (replacing ENORWH with ENORH and RW)

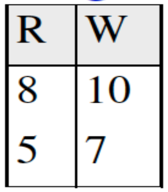
Analysing functional dependencies can help us identify schemas with problems and to suggest design refinements (i.e., better grouping of attributes).

Hourly_Emps2



E	N	O	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

Wages



R	W
8	10
5	7

(Functional) Dependency Theory

So a relational design theory, functional dependency theory in particular helps us:

Decide whether a given relation R is in “good” form.

In the case that a relation R is not in “good” form, decompose it into a set of relations {R1, R2, ..., Rn} such that:

- each relation is in good form
- the decomposition is a lossless-join decomposition

(i.e., R1 join R2 join … Rn = R)

Functional Dependency

A relation instance *r(R)* satisfies a dependency $X \rightarrow Y$ if

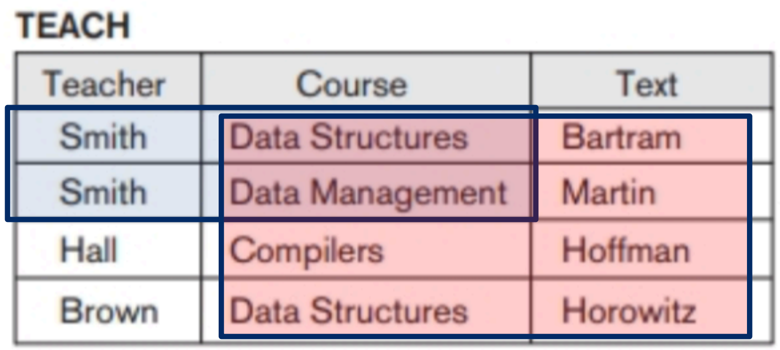
- for any $t, u \in r$, $t[X] = u[X]$ *implies that* $t[Y] = u[Y]$

In other words, if two tuples in *R* agree in their values for the set of attributes *X*, then they must also agree in their values for the set of attributes *Y*.

e.g., t[BeerName] = ‘Lager’, u[BeerName] = ‘Lager’ implies t[Manf]=’Carlton’, u[Manf]=’Carlon’

If in a relation instance *r(R)* a dependency $X \rightarrow Y$ holds

- This means that the values of the Y component of a tuple in r are determined by, the values of the X component; alternatively, we say the X component of a tuple functionally determines the Y component.


TEACH

Teacher	Course	Text
Smith	Data Structures	Bartram
Smith	Data Management	Martin
Hall	Compilers	Hoffman
Brown	Data Structures	Horowitz

Figure 14.7
A relation state of TEACH with a *possible* functional dependency TEXT → COURSE. However, TEACHER → COURSE, TEXT → TEACHER and COURSE → TEXT are ruled out.

Functional Dependency

If in a relation instance *r(R)* a dependency *X → Y* holds

- This means that the values of the Y component of a tuple in r are determined by, the values of the X component; alternatively, the values of the X component of a tuple functionally determine the values of the Y component.
- We read *X → Y (X functionally determines Y, or X determines Y)*

NOTE:

- If X → Y in R, this does not say whether or not Y → X in R.
  - Say, from the definition of FD, we can imagine emp_id → emp_phone, but we cannot say for sure that emp_phone → emp_id … (e.g., shared office?)
- The abbreviation for functional dependency is FD or f.d.
- X, Y can be a set of attributes (not just a single)   {A,B} → {C,D}
- The set of attributes X is called the left-hand side of the FD, and Y is called the right-hand side

Example

Table Drinkers(**n**ame, **a**ddr, **b**eersLiked, **m**anf, **f**avBeer) $\rightarrow$ NABMF for short

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	Voyager	WickedAle	Pete's	WickedAle
Spock	Enterprise	Bud	A.B.	Bud

Reasonable FD's to assert:

- **name** → **addr** (or N→ A)
- name → favBeer (or N → F)
- beersLiked → manf (or B → M)
- favBeer → manf (??)

NOTE: generally you cannot just look at the tuples to decide on FD. FD should hold for all possible instances of the table, not just for the given tuples/instances …
Normally, you'd rely on your domain knowledge to decide …

FD's With Multiple Attributes

FD's left and right can have multiple attributes …

The attribute on the right can be combined (i.e., short handed)

Example: name → addr and name → favBeer

become

name → **a**ddr **f**avBeer   (or N→ AF)

Multiple attributes on the left may have more crucial role when together … (i.e., the "combination" determines the right hand)

Example: **b**ar **b**eer → **p**rice   (or AB → P)

Functional Dependency

T1

A	B
X1	Y1
X2	Y2
X3	Y1
X4	Y1
X5	Y2
X6	Y2

T2

A	B
X1	Y1
X2	Y4
X1	Y1
X3	Y2
X2	Y4
X4	Y3

T3

A	B
X1	Y1
X2	Y4
X1	Y1
X3	Y2
X2	Y4
X4	Y4

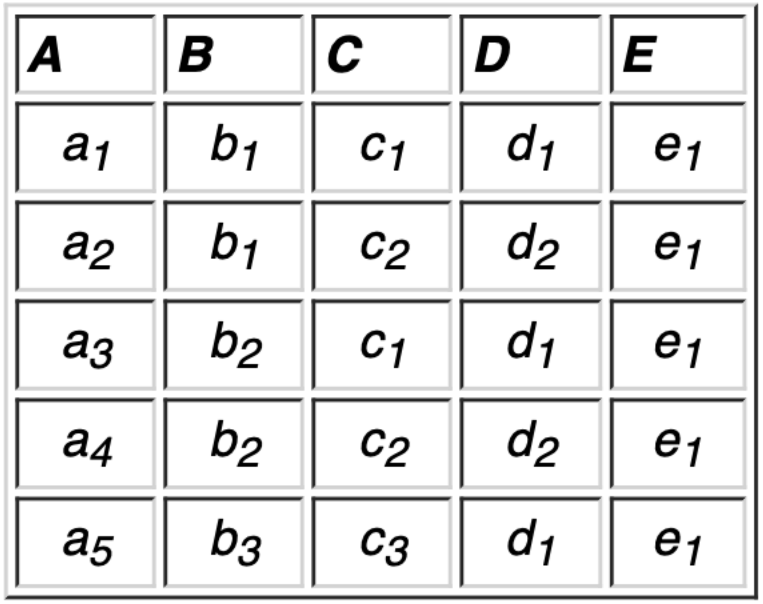
T1 , A $\rightarrow$ B holds … attempt to check if r1(A) = r2(A) then r1(B) = r2(B), however, there is no pair of rows in T1 that have equal vale for A, so the condition is trivially satisfied. In fact, take note of this case … If A is unique for all tuples, A can determine the whole relation. B $\rightarrow$ A does not hold … see tuple 1 and tuple 3

T2, A $\rightarrow$ B holds, see tuple 1 and tuple 3, or tuple 2 and tuple 5, B $\rightarrow$ A holds as well

T3, A $\rightarrow$ B holds, but B $\rightarrow$ A does not, see tuple 2 and 6.

Functional Dependency

Consider the following instance *r(R)* of the relation schema *R(ABCDE)*:

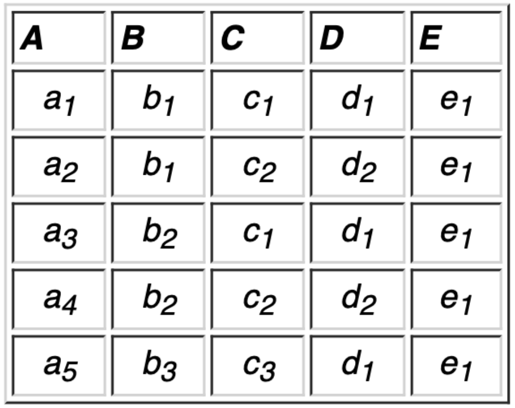


A	*B*	*C*	*D*	*E*
a_1	b_1	c_1	d_1	e_1
a_2	b_1	c_2	d_2	e_1
a_3	b_2	c_1	d_1	e_1
a_4	b_2	c_2	d_2	e_1
a_5	b_3	c_3	d_1	e_1

What kind of dependencies can we observe among the attributes in r(R)?

Note: here, we do not know the domain of schema, so we will just observe the given tuples

Functional Dependency



A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_1	c_2	d_2	e_1
a_3	b_2	c_1	d_1	e_1
a_4	b_2	c_2	d_2	e_1
a_5	b_3	c_3	d_1	e_1

Since the values of *A* are unique,

it follows from the *fd* definition that:

$$A \rightarrow B, \quad A \rightarrow C, \quad A \rightarrow D, \quad A \rightarrow E$$

The right side can be combined … This can be summarised as $A \rightarrow BCDE$

From our understanding of primary keys, *A* is a PK.

We will see later how FD helps compute potential keys in a relation effectively.

Functional Dependency

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_1	c_2	d_2	e_1
a_3	b_2	c_1	d_1	e_1
a_4	b_2	c_2	d_2	e_1
a_5	b_3	c_3	d_1	e_1

Other observations

(mainly aiming for "uniqueness" in tuples):

- combinations of *BC* are unique, therefore $BC \rightarrow ADE$
- combinations of *BD* are unique, therefore $BD \rightarrow ACE$
- if *C* values match, so do *D* values, therefore $C \rightarrow D$
- however, *D* values don't determine *C* values, so $D \nrightarrow C$

We could derive many other dependencies, e.g. $AE \rightarrow BC$, ...

In practice, choose a minimal set of *fd*s (*basis*)

- from which all other *fd*s can be derived
- which typically captures useful problem-domain information

Use of Functional Dependencies

We use functional dependencies to:

1. **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.
2. **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.

An FD is a statement about all allowable relations.

- Must be identified based on semantics of application.
- Given some allowable instance r1 of R, we can check if it violates some FD f, but we cannot tell if f holds over R!

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 Students may, by chance, satisfy Sname → *Sid*

Inference rules on FDs

Can we generalise some ideas about functional dependency?

E.g. are there dependencies that hold for *any* relation?

Yes, but they're rather uninteresting (or trivial/obvious) ones such as:

- $X \rightarrow X$ *(e.g., emp_id → emp_id, emp_name → emp_name)*
- $Y \subseteq X$ *implies* $X \rightarrow Y$ *(e.g., (emp_id, emp_name) → emp_name)*

E.g. do some dependencies suggest the existence of others?

Yes, and this is much more interesting ... there are a number of *rules of inference* that allow us to *derive* more dependencies given some dependencies …!

Amstrong's Axioms & Inference Rules

So … given some FDs, we can usually infer additional FDs:

e.g., $\text{ssn} \rightarrow \text{did}$, $\text{did} \rightarrow \text{lot}$ implies $\text{ssn} \rightarrow \text{lot}$

We use **Amstrong's Axioms** and Inference Rules to derive FDs …

Let's say F is given FDs … then F closure (denote F^+) is all FDs that can logically imply from F … Amstrong's Axiom is both "sound" and "complete" which means, when you apply Amstrong's Axiom on F:

- You won't find any fd that is OUTSIDE of F+ (i.e., no wrong fd will be derived)
- You will find ALL fds that can be implied from F

Amstrong's Axioms & Inference Rules

Armstrong's axioms: (X, Y are a set of attributes)

- **(F1) Reflexivity** **If $X \supseteq Y$, then $X \rightarrow Y$**

a formal statement of *trivial dependencies*; useful for derivations of other fd

- ***(F2) Augmentation*** $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$

if a dependency holds, then we can add an extra component to both sides

(useful for expanding the left side)

- ***(F3) Transitivity*** **e.g.** $X \rightarrow Y, Y \rightarrow Z \Rightarrow X \rightarrow Z$

the "most powerful" inference rule; useful in multi-step derivations

Amstrong's Axioms & Inference Rules

While *Armstrong's rules* are complete, there are other useful rules exists:

- ***(F4) Additivity (or Union)*** $X \to Y, X \to Z \;\Rightarrow\; X \to YZ$

useful for constructing new right hand sides of *fd*s

- ***(F5) Projectivity (or Decomposition)*** $X \to YZ \;\Rightarrow\; X \to Y, X \to Z$

useful for reducing right hand sides of *fd*s

- ***(F6) Pseudotransitivity*** $X \to Y, YZ \to W \;\Rightarrow\; XZ \to W$

shorthand for a common transitivity derivation

Applying the Inference Rules

(F1) Reflexivity If $X \supseteq Y$, then $X \rightarrow Y$	*(F4) Union* $X \rightarrow Y, X \rightarrow Z \Rightarrow X \rightarrow YZ$
(F2) Augmentation $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$	*(F5) Decomposition* $X \rightarrow YZ \Rightarrow X \rightarrow Y, X \rightarrow Z$
(F3) Transitivity $X \rightarrow Y, Y \rightarrow Z \Rightarrow X \rightarrow Z$	*(F6) Pseudotransit* $X \rightarrow Y, YZ \rightarrow W \Rightarrow XZ \rightarrow W$

R = ABCDE, consider the given set of FDs

F = { $A \rightarrow BC$, $CD \rightarrow E$, $B \rightarrow D$, $E \rightarrow A$}

$A \rightarrow A$ (F1)	$A \rightarrow CD$ (F4)
$B \rightarrow B$ (F1)	$A \rightarrow E$ (F3)
$C \rightarrow C$ (F1)	$A \rightarrow ABCDE$ (F4)
$D \rightarrow D$ (F1)	$E \rightarrow ABCDE$ (F3)
$E \rightarrow E$ (F1)	$CD \rightarrow ABCDE$ (F3)
$A \rightarrow B$ (F5)	$BC \rightarrow CD$ (F2)
$A \rightarrow C$ (F5)	$BC \rightarrow ABCDE$ (F3)
$A \rightarrow D$ (F3)	

i.e., Using the rules, we can logically imply all FDs (F closure) from given F

Closure

Given a set *F* of *fd*s, how many new *fd*s can we derive?

For a finite set of attributes, there must be a finite set of *fd*s.

The largest collection of dependencies that can be derived from *F* is called the *closure* of *F* and is denoted F^+ *(read F closure …)*

Closures allow us to answer two questions:

- Q1: is a particular dependency $X \rightarrow Y$ derivable from *F*?
- Q2: are two sets of dependencies *F* and *G* equivalent?

Closure

For Q1, the question "is $X \rightarrow Y$ derivable from F?" ...

compute the closure F^+; check whether $X \rightarrow Y \in F^+$

For Q2, the question "are F and G equivalent?" ...

compute closures F^+ and G^+; check whether they're equal

Unfortunately, closures on even small sets of functional dependencies can be very large. Algorithms based on F^+ rapidly become infeasible.

Example (of *fd* closure):

$R = ABC, \quad F = \{ AB \rightarrow C, \quad C \rightarrow B \}$

$F^+ = \{ A \rightarrow A, \quad AB \rightarrow A, \quad AC \rightarrow A, \quad AB \rightarrow B, \quad BC \rightarrow B, \quad ABC \rightarrow B,$
$C \rightarrow C, \quad AC \rightarrow C, \quad BC \rightarrow C, \quad ABC \rightarrow C, \quad AB \rightarrow AB, \quad . \ . \ . \ . \ . \ . \ . \ ,$
$AB \rightarrow ABC, \quad AB \rightarrow ABC, \quad C \rightarrow B, \quad C \rightarrow BC, \quad AC \rightarrow B, \quad AC \rightarrow AB \}$

Closure of Attribute Sets

Take Q1, to answer if $X \rightarrow Y$ is derivable from *F? The strategy is compute* F^+; check whether $X \rightarrow Y \in F^+$ ….

Since computing F^+ is expensive, we reduce the computation task to "closure of attribute sets"

Say given R = ABC, $F = \{ AB \rightarrow C, \quad C \rightarrow B \}$

Consider a set *X* of attributes and a set *F* of *fd*s, a closure of attribute set X is *the largest set of attributes that can be determined by X using F (denoted X^+)*.

We say that $(X \rightarrow Y) \in F^+$ iff $Y \subset X^+$

For computation, | X^+ | is bounded by the number of attributes.

Say the question is AC $\rightarrow$ *B, here, X = {AC}, Y = {B}, X closure = {ABC}, {B}* $\subset$ X^+*, so the answer is YES.*

Closure of Attribute Sets

From the following R and FD … let's derive some attribute set closures

R = ABCDE

FD = { A $\rightarrow$ *B, B* $\rightarrow$ *C, C* $\rightarrow$ *D, D* $\rightarrow$ *E}*

$A \rightarrow A$	$B \rightarrow B$	$C \rightarrow C$
$A \rightarrow B$	$B \rightarrow C$	$C \rightarrow D$
$A \rightarrow C$	$B \rightarrow D$	$C \rightarrow E$
$A \rightarrow D$	$B \rightarrow E$	$C \rightarrow CDE$
$A \rightarrow E$	$B \rightarrow BCDE$	
$A \rightarrow ABCDE$		*And so on …*

According to the definition,
Attribute Closure of A (A^+) = {A,B,C,D,E}
Attribute Closure of B (B^+) = {B,C,D,E}, etc.
How about AD^+ or other combinations…. Is there a way to come up with this quickly?

Attribute Set Closure Algorithm

R = ABCDE

FD = { A → *B, B* → *C, C* → *D, D* → *E}*

What is AD closure?

```
Inputs: set $F$ of $fds$
        set $X$ of attributes
Output: closure of $X$ (i.e. $X^+$)

$X^+ = X$
stillChanging = true;
while (stillChanging) {
    stillChanging = false;
    for each $W \rightarrow Z$ in $F$ {
        if ($W \subseteq X^+$) and not ($Z \subseteq X^+$) {
            $X^+ = X^+ \cup Z$
            stillChanging = true;
        }
    }
}
```

Attribute Set Closure

E.g. $R = ABCDEF$, $Z = \{ AB \rightarrow C,\ BC \rightarrow AD,\ D \rightarrow E,\ CF \rightarrow B \}$

Does $AB \rightarrow D$ follow from Z? Solve by checking $D \in AB^+$.

Computing AB^+:

1.	$AB^+ = AB$	(initially)
2.	$AB^+ = ABC$	(using $AB \rightarrow C$)
3.	$AB^+ = ABCD$	(using $BC \rightarrow AD$)
4.	$AB^+ = ABCDE$	(using $D \rightarrow E$)

Since D is in AB^+, then $AB \rightarrow D$ does follow from Z.

Attribute Set Closure

E.g. $R = ABCDEF$, $Z = \{ AB \rightarrow C,\ BC \rightarrow AD,\ D \rightarrow E,\ CF \rightarrow B \}$

Does $D \rightarrow A$ follow from Z? Solve by checking $A \in D^+$.

Computing D^+:

1. $D^+ = D$ (initially)
2. $D^+ = DE$ (using $D \rightarrow E$)

Since A is not in D^+, then $D \rightarrow A$ does not follow from Z.

Utilising Attribute Set Closure …

For Q1, the question "is $X \rightarrow Y$ derivable from F?" ...

compute the closure X^+, check whether $Y \subset X^+$

For Q2, the question "are F and G equivalent?" ... The strategy is to For Q2, the compute closures F^+ and G^+; check whether they're equal

- for each dependency in G, check whether derivable from F
- for each dependency in F, check whether derivable from G
- if true for all, then $F \Rightarrow G$ and $G \Rightarrow F$ which implies $F^+ = G^+$

Interesting use case of attribute set closure:

For Q3 the question "what are the keys of R implied by F?" ...

to answer this, we find subsets $X \subset R$ such that $X^+ = R$

(in other words, find a set of attributes whose "closure" contains all attributes of the relation, then X is a superkey (i.e., contains a key))

Attribute Closure to Computing Keys

Let's examine the keys and attribute closure again …

$R = ABCDE$, $F = \{ A \rightarrow B,\ B \rightarrow C,\ C \rightarrow D,\ D \rightarrow E \}$

What are the keys of R? … Find $X \subset R$ such that $X^+ = R$.

Attribute Closure to Computing Keys

E.g. $R = ABCDEF$, $Z = \{ AB \rightarrow C,\ BC \rightarrow AD,\ D \rightarrow E,\ CF \rightarrow B \}$

What are the keys of R?

Solve by finding $X \subset R$ such that $X^+ = R$.

From previous examples (slide 34/35), we know *AB* and *D* are not keys:

- AB+ = {ABCDE}
- D+ = {DE}
- Both are not R.

This also implies that *A* and *B* alone are not keys.

So how to find keys? Try all combinations of *ABCDEF* ...

E.g. maybe *ACF* is a key ...

Attribute Closure to Computing Keys

E.g. $R = ABCDEF$, $Z = \{ AB \rightarrow C,\ BC \rightarrow AD,\ D \rightarrow E,\ CF \rightarrow B \}$

What are the keys of *R*?

Solve by finding $X \subset R$ such that $X^+ = R$.

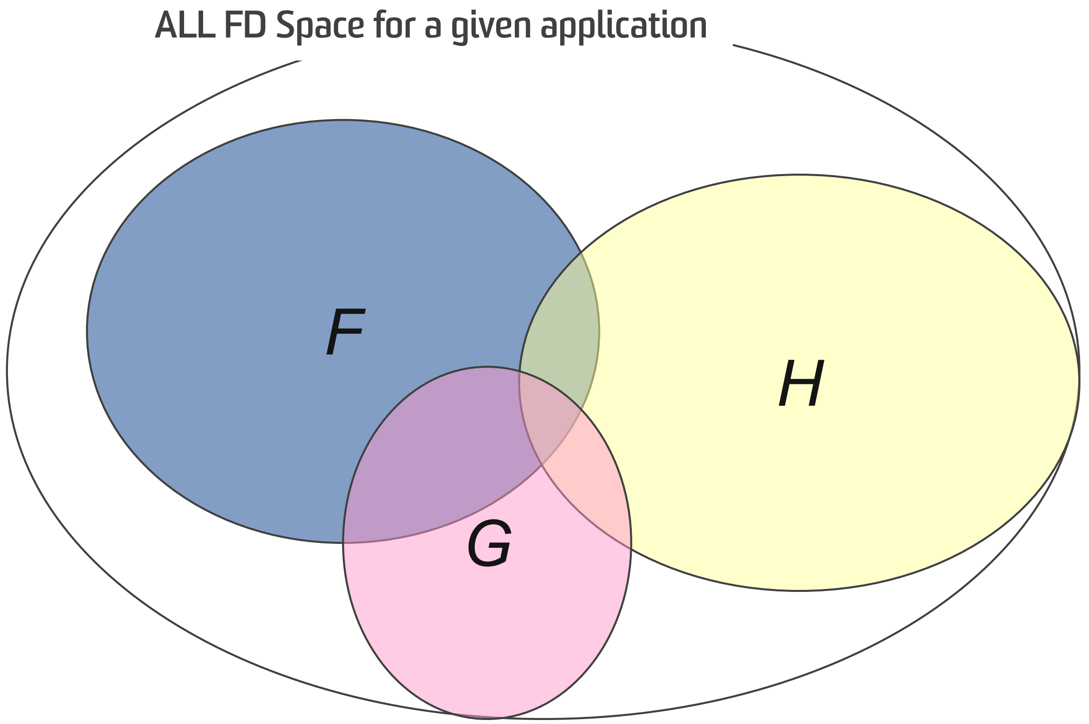
Computing ACF^+:

1.	$ACF^+ = ACF$	(initially)
2.	$ACF^+ = ABCF$	(using $CF \rightarrow B$)
3.	$ACF^+ = ABCDF$	(using $BC \rightarrow AD$)
4.	$ACF^+ = ABCDEF$	(using $D \rightarrow E$)

Since $ACF^+ = R$, *ACF* is a key (as is *ABF* … *is ACF or ABF a candidate key?)*

Minimal Covers

For a given application, we can define many different sets of *fd*s whose closure is the same (e.g. *F* and *G* where $F^+ = G^+$) …



F closure = H closure = G closure
= ALL FDs for a given application

Which one is best to "model" the application?

Minimal Covers

Which one is best to "model" the application?

- any model has to be complete (i.e. capture entire semantics)
- models should be as small as possible (we use them to check DB validity after update; less checking is better)

If we can ...

- determine a number of candidate *fd* sets, *F*, *G* and *H*
- establish that $F^+ = G^+ = H^+$
- we would then choose the smallest one for our "model"

Better still, can we *derive* the smallest complete set of *fd*s? Sets of functional dependencies may have redundant dependencies that can be inferred from the others

E.g. $\{A \rightarrow B, B \rightarrow C, A \rightarrow CD\}$ can be simplified to $\{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$

E.g. $\{A \rightarrow B, B \rightarrow C, AC \rightarrow D\}$ can be simplified to $\{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$

Intuitively, a canonical cover of F is a ***"minimal" set of functional dependencies*** equivalent to F, with no redundant dependencies or having redundant parts of dependencies

Minimal Covers

Minimal cover F_c for a set F of *fd*s:

- F_c is equivalent to F
- all *fd*s have the form $X \rightarrow A$ (where A is a single attribute)
- it is not possible to make F_c smaller
  - » either by deleting an *fd*
  - » or by deleting an attribute from an *fd*

An *fd* d is redundant if $(F\text{-}\{d\})^+ = F^+$

An attribute a is redundant if $(F\text{-}\{d\} \cup \{d'\})^+ = F^+$
(where d' is the same as d but with attribute A removed)

i.e., Remove attributes in d ∈ F that does not change F^+

Minimal Covers

Algorithm for computing minimal cover:

An Overview of the Algorithm …

```
Inputs: set $F$ of $fds$
Output: minimal cover $F_c$ of $F$

$F_c$ = $F$
Step 1:
    put $f \in F_c$ into canonical form
Step 2:
    eliminate redundant attributes from $f \in F_c$
Step 3:
    eliminate redundant $fds$ from $F_c$
```

Minimal Covers

E.g., R = ABC, F = { $A \rightarrow BC$, $B \rightarrow C$, $A \rightarrow B$, $AB \rightarrow C$ }

Step 1: put *fd*s into canonical form

```
for each $f \in F_c$ like $X \rightarrow \{A_1,..,A_n\}$
    $F_c = F_c - \{f\}$
    for each $a$ in $\{A_1,..,A_n\}$
        $F_c = F_c \cup \{X \rightarrow a\}$
    end
end
```

- canonical *fd*s: $A \rightarrow B$, $A \rightarrow C$, $B \rightarrow C$, $AB \rightarrow C$

Minimal Covers

E.g., R = ABC, F = { $A \rightarrow BC$, $B \rightarrow C$, $A \rightarrow B$, $AB \rightarrow C$ *}*

Step 2: eliminate redundant attributes

- canonical *fds*: $A \rightarrow B$, $A \rightarrow C$, $B \rightarrow C$, $AB \rightarrow C$

```
for each f ∈ $F_c$ like X → A
    for each b in X
        f' = (X-{b}) → A;   // what if I removed b from f ??
        G = $F_c$ - {f} U {f'}
        if ($G^+$ == ${F_c}^+$) $F_c$ = G
    end
end
```

- redundant attrs: $A \rightarrow B$, $A \rightarrow C$, $B \rightarrow C$, $AB \rightarrow C$

Minimal Covers

E.g., $R = ABC$, $F = \{ A \to BC,\ \ B \to C,\ \ A \to B,\ \ AB \to C \}$

Step 3: eliminate redundant functional dependencies

- redundant attrs: $A \to B,\ \ A \to C,\ \ B \to C,\ \ AB \to C$

```
for each $f \in F_c$
    $G = F_c - \{f\}$
    if $(G^+ == F_c^{\ +})$ $F_c = G$
end
```

- This gives the minimal cover $F_c = \{ A \to B,\ \ B \to C \}$.

Summary so far ..

- Functional dependency (FD) theory forms the base of relational schema design theory. In part two, we will see how they are used in deriving good schema design
- There are other forms dependencies, such as multivalued dependency (MVD), inclusion dependency, etc.
- In practical applications, FD seems sufficient to derive good schema. So we consider that as the most important dependency theory to learn and practice with.