COMP 3311
DATABASE MANAGEMENT SYSTEMS

LECTURE 5
RELATIONAL MODEL AND RELATIONAL DATABASE DESIGN

FUNCTIONAL DEPENDENCY (FD): DEFINITION

Let R be a relation schema, X, Y be sets of attributes in R and *f* be a time-varying function from X to Y. Then

$$f\text{: } X \rightarrow Y$$

is a *functional dependency (FD)* if *at every point in time*, for a given value of *x* in X there will be at most one value of *y* in Y.

Example: PGStudent(studentId, name, supervisor, specialization)

$$f\text{: supervisor} \rightarrow \text{specialization}$$

- If two student records have the same supervisor (e.g., Papadias), then they must have the same specialization (e.g., Databases).
- On the other hand, if two students have different supervisors, then they may have the same or different specializations.

PGStudent

studentId	name	supervisor	specialization
23455789	Bruno Ho	Yang	Artificial Intelligence
23556789	Jenny Jones	Papadias	Database
25678989	Kathy Ko	Kim	Software Technology
26789012	Susan Sze	Papadias	Database
26184624	Terry Tam	Song	Artificial Intelligence
26186666	Carol Chen	Tai	Graphics

FUNCTIONAL DEPENDENCY (FD): DEFINITION

- In general, X (and Y) can be sets of attributes.
  E.g., if $X \equiv X_1 \cup X_2 \cup \ldots \cup X_n$, then we can write $X_1X_2...X_n \rightarrow Y$

- We normally omit the "*f*:" and simply write the FD as $X \rightarrow Y$.
  - X is called the determinant set or left hand side (LHS) of the FD.
  - Y is called the dependent set or right hand side (RHS) of the FD.

  ☞ **We say that X determines Y or Y depends on X.**

- A functional dependency $X \rightarrow Y$ is trivial if Y is a subset of X.

  ☞ **Y appears on both the LHS and the RHS of the FD.**

  ➢ Trivial FDs hold for all relation instances.

- A functional dependency $X \rightarrow Y$ is non-trivial if $Y \cap X = \emptyset$.

  ☞ **Y does not appear on both the LHS and the RHS of the FD.**

  ➢ Non-trivial FDs are given as constraints when designing a database.

  ➢ Non-trivial FDs constrain the set of legal relation instances.

FUNCTIONAL DEPENDENCIES AND KEYS

- A FD is a generalization of the concept of a *key*.

  For the relation

  PGStudent (studentId, name, supervisor, specialization)

  we can write:

  studentId → name, supervisor, specialization

  because the key, studentId, determines the value of all the attributes (i.e., the entire tuple).

- If two tuples in the PGStudent relation have the same studentId, then they must have the same values on all attributes.

  ☞ **They must be the same tuple!**
  (Since the relational model does not allow duplicate tuples.)

INFERENCE RULES FOR FDS

- Given a set of functional dependencies F, there are certain other functional dependencies that are *logically implied* by F.
- We can find all functional dependencies implied by F by applying the following inference rules for FDs.

IR1: Reflexivity	If $Y \subseteq X$, then $X \rightarrow Y$ (*trivial FD*)	**Armstrong's Axioms** (basic rules)
IR2: Augmentation	$X \rightarrow Y \vDash XZ \rightarrow YZ$	
IR3: Transitivity	$X \rightarrow Y,\ Y \rightarrow Z \vDash X \rightarrow Z$	
IR4: Union	$X \rightarrow Y,\ X \rightarrow Z \vDash X \rightarrow YZ$	**Additional rules** (derivable from IR1, IR2 and IR3)
IR5: Decomposition	$X \rightarrow YZ \vDash X \rightarrow Y$ and $X \rightarrow Z$	
IR6: Psuedotransitivity	$X \rightarrow Y,\ WY \rightarrow Z \vDash WX \rightarrow Z$	

EXAMPLES OF ARMSTRONG'S AXIOMS

IR1: Reflexivity If $Y \subseteq X$, then $X \rightarrow Y$ (*trivial FD*)

name → name	since name ⊆ name
name, supervisor → name	since name ⊆ {name, supervisor}
name, supervisor → supervisor	since supervisor ⊆ {name, supervisor}

IR2: Augmentation $X \rightarrow Y \vDash XZ \rightarrow YZ$

studentId → name	(given)
studentId, supervisor → name, supervisor	(inferred)

IR3: Transitivity $X \rightarrow Y,\ Y \rightarrow Z \vDash X \rightarrow Z$

studentId → supervisor	(given)
supervisor → specialization	(given)
studentId → specialization	(inferred)

INFERENCE RULES FOR FDS AND CLOSURE

Inference rules IR1, IR2 and IR3 are sound and complete.

sound: Given a set of FDs, F, specified on a relation schema R, any FD that we can infer from F by using **IR1** to **IR3** *will hold* in every relation instance *r* of R that satisfies F (i.e., it is a *true* FD).

complete: Using **IR1**, **IR2** and **IR3** repeatedly to infer FDs will infer *all* the FDs that can be inferred from F. (i.e., there are *no other FDs* that are true).

The set of all functional dependencies *logically implied* by F is called the closure of F denoted as F^+.

CLOSURE OF ATTRIBUTE SETS

- The closure of X under F (denoted by X^+) is the set of attributes that are functionally determined by X under F.

$$X \rightarrow Y \text{ is in } F^+ \Leftrightarrow Y \subseteq X^+$$

X is a set of attributes

Given studentId (e.g., studentId is X).

If studentId $\rightarrow$ name
then name is part of studentId$^+$
(i.e., studentId$^+$= {studentId, name, …})

Why is studentId in X^+?

If studentId $\rightarrow$ supervisor
then supervisor is part of studentId$^+$
(i.e., studentId$^+$= {studentId, name, supervisor, …})

COMPUTING ATTRIBUTE CLOSURE: ALGORITHM

Input:

R a relation schema
F a set of functional dependencies
$X \subset R$ (the set of attributes for which we want to compute the closure)

Output:

X^+ the closure of X w.r.t. F

$X^{(0)} := X$
Repeat
$X^{(i+1)} := X^{(i)} \cup Z$, where Z is the set of attributes such that there exists
$Y \rightarrow Z$ in F, and $Y \subset X^{(i)}$
Until $X^{(i+1)} := X^{(i)}$
Return $X^{(i+1)}$

☞ **For every attribute Y in X^i, if Y is the LHS of an FD, then add the RHS attributes Z to the closure; repeat until there are no more attributes to add.**

USES OF ATTRIBUTE CLOSURE

Testing for Superkey

To test if X is a superkey, compute X^{+}, and check if X^{+} contains all attributes of R. If X is minimal, then it is a candidate key.

☞ An attribute that is part of *any* candidate key is called a prime attribute; otherwise it is a nonprime attribute

Testing Functional Dependencies

To determine whether a functional dependency $X \rightarrow Y$ holds (i.e., if $X \rightarrow Y$ is in F^{+}), compute X^{+} and check if $Y \subseteq X^{+}$.

Computing the Closure of *F*

For each subset $X \subseteq R$, compute X^{+} and, for each $Y \subseteq X^{+}$, output a functional dependency $X \rightarrow Y$.

CLOSURE OF A SET OF ATTRIBUTES: EXAMPLE

R(A, B, C, D, E, G)

F = { AB→C, C→A, BC→D, ACD→B, D→EG, BE→C, CG→BD, CE→AG }

☞ Compute the closure of X = {B, D} w.r.t F

$X^{(0)}$ = {B, D}

$X^{(1)}$ = {B, D, E, G} apply D→EG add E, G to X

$X^{(2)}$ = {B, C, D, E, G} apply BE→C add C to X

$X^{(3)}$ = {A, B, C, D, E, G} apply C→A add A to X

$X^{(4)} = X^{(3)}$ no more FDs can be applied

$\{B, D\}^+$ = {A, B, C, D, E, G} **Is BD a candidate key?**

EXAMPLE RELATION SCHEMA & DATABASE

Car

make	model	engineSize	fee	origin	tax
Nissan	Sunny	1	4,000	Japan	90
Fiat	Mirafiori	1	4,000	Italy	85
Honda	Accord	1	4,000	Japan	90
Toyota	Camry	4	7,000	Canada	50
Ford	Mustang	4	7,000	Canada	50
Ford	Mustang	2	5,000	U.S.A.	75
BMW	7.35i	3	6,000	Germany	95
Toyota	Camry	1	4,000	Japan	90

Functional Dependencies

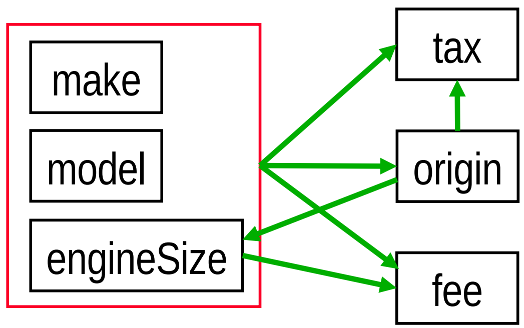
make, model, engineSize → origin
make, model, engineSize → tax
make, model, engineSize → fee

due to the primary key

origin → tax
engineSize → fee
origin → engineSize

from real-world knowledge

FD visualization



FDS AND RELATIONAL DATABASE DESIGN

- Relational database design requires that we find a "good" collection of relation schemas.

  ☞ **A bad design may lead to several problems.**

- FDs can be used to refine a relation schema reduced from an E-R schema by iteratively decomposing it (called normalization) to place it in a certain normal form.
  - The first four normal forms ⟹ use only FDs.
  - Additional normal forms ⟹ use other types of dependencies

☞ **Normal forms do not guarantee a good design!**

NORMALIZATION: GOALS

Design Guideline 1: Clear Semantics for Attributes

Design a relation schema so that it is easy to explain its meaning. Typically this means that we should not combine attributes from multiple real-world entities in a single relation schema.

- Grouping attributes into relation schemas puts an implied meaning on the attributes $\Longrightarrow$ somehow the attributes are related.
- The easier it is to explain the meaning of a relation schema, the better the design.
- Each relation schema should have a well-defined, unambiguous meaning.
  - We are able to give a clear explanation to the question "What does this relation schema represent?"

NORMALIZATION: GOALS (CONT'd)

Design Guideline 2: Minimize Use of Null Values

As far as possible, avoid placing attributes in relation schemas whose values may be null. If nulls are unavoidable, make sure that they apply in exceptional cases only.

- If many attributes of a relation schema do not apply to all instances, we end up with many null values.
- This can lead to problems of
  - understanding the meaning of attributes.
  - specifying certain operations (e.g., aggregation operations).

NORMALIZATION: GOALS (CONT'D)

Design Guideline 3: Minimize Redundancy

Design relation schemas so that no insertion, deletion or update anomalies occur in the relation instances. If any update anomalies are present, note them clearly so that update programs will operate correctly.

– One goal of database design is to minimize storage space for data.

☞ **A relation schema has redundancy if there is a FD where the LHS is not a key.**

– More importantly, redundant data in relations can also cause operation anomalies.

- insertion (e.g., insert the license fee for cars of engine size 5)
- deletion (e.g., delete the instance for "BMW, 7.35i, …")
- update (e.g., update the license fee for engine size 1 cars)

NORMALIZATION: GOALS (CONT'd)

Design Guideline 4: Lossless Decomposition

The normalized relation schemas should contain the same information as the original schema. Otherwise decomposition results in information loss.

– A decomposition is lossless (aka lossless join) if the initial relation instance can be recovered from the schema fragments.

  - The join of all the fragments results in exactly the initial relation instance.

– In general a decomposition of R into R_1 and R_2 is lossless *if and only if* at least one of the following functional dependencies is in F^+:

$$R_1 \cap R_2 \rightarrow R_1 \qquad\qquad R_1 \cap R_2 \rightarrow R_2$$

☞ **The common attributes of R_1 and R_2 must be a superkey for R_1 *or* R_2.**

LOSSY DECOMPOSITION EXAMPLE

Decompose R(A, B, C) into R_1(A, B) and R_2(B, C).

R

A	B	C
a	1	*m*
a	2	*n*
b	1	*p*

decompose →

R_1

A	B
a	1
a	2
b	1

R_2

B	C
1	*m*
2	*n*
1	*p*

R_1 JOIN R_2

A	B	C
a	1	*m*
a	1	*p*
a	2	*n*
b	1	*m*
b	1	*p*

The *decomposition is lossy* since the join produces two extra tuples. Thus, the decomposition "loses" some information! Note that the common attribute B is not a superkey of either R_1 or R_2.

NORMALIZATION: GOALS (CONT'd)

Design Guideline 5: Preserve Functional Dependencies

As far as possible, functional dependencies should be preserved within each relation schema; otherwise, checking updates for violation of functional dependencies may require computing joins, which is expensive.

- Functional dependencies represent real-world constraints.
- If a functional dependency does not appear in any relation schema (i.e. it is "lost"), the constraint may be much more difficult to enforce.
- The decomposition of a relation schema R with FDs F is a set of schema fragments R_i with FDs F_i.
  - F_i is the subset of dependencies in F^+ (the closure of F) that involves only attributes in R_i.
- The decomposition is dependency preserving *if and only if* $(\cup F_i)^+ = F^+$.
  - Every FD in F is present in some fragment R_i.

NON-DEPENDENCY PRESERVING DECOMPOSITION EXAMPLE

R(A, B, C) F = {A→B, B→C, A→C} **Key:** A

For the FD B→C, the LHS is not the key. Consequently, there can be considerable redundancy in R.

Solution: Break R into relations R_1(A, B), R_2(A, C) (normalization).

R

A	B	C
1	2	3
2	2	3
3	2	3
4	2	4

X Invalid! Violates B→C.

decompose

R_1

A	B
1	2
2	2
3	2
4	2

✓ Valid! No FD violation.

R_2

A	C
1	3
2	3
3	3
4	4

✓ Valid! No FD violation.

The decomposition is lossless since the common attribute A is a key for R_1 (and R_2).

The decomposition is not dependency preserving because F_1 = {A→B}, F_2 = {A→C} and $(F_1 \cup F_2)^+ \neq F^+$. **The FD B→C is lost.**

In practice, each "lost" FD is implemented as an assertion (a type of constraint), which is checked when there are updates. Thus, to find violations on B→C, R_1 and R_2 have to be joined, which can be very expensive.

DEPENDENCY PRESERVING DECOMPOSITION EXAMPLE

R(A, B, C) F = {A→B, B→C, A→C}   **Key:** A

Break R into relations R_1(A, B), R_2(B, C).

R

A	B	C
1	2	3
2	2	3
3	2	3
4	2	4

X Invalid! Violates B→C.

decompose

✓ Valid! No FD violation.

R_1

A	B
1	2
2	2
3	2
4	2

R_2

B	C
2	3
2	4

X Invalid! Violates B→C.

The decomposition is lossless since the common attribute B is a key for R_2.

The decomposition is dependency preserving because F_1 = {A→B}, F_2 = {B→C} and $(F_1 \cup F_2)^+ = F^+$.

Violations of the FDs can be found by inspecting the individual tables, without performing a join.

☞ **How a relation is decomposed, may determine whether functional dependencies are preserved.**

FIRST NORMAL FORM (1NF)

A relation schema is in *First Normal Form (1NF)* if all attributes are atomic (single-valued).

☞ **There are no multi-valued or composite attributes.**

- Relation schemas are always in 1NF according to the definition of the relational model and according to our strategy for reducing an E-R schema to relation schemas.

SECOND NORMAL FORM (2NF)

A relation schema is in *Second Normal Form (2NF)* if all non-prime attributes are <u>fully</u> functionally dependent on every candidate key.

- R is a relation schema, with the set F of FDs.

- R is in 2NF *if and only if*

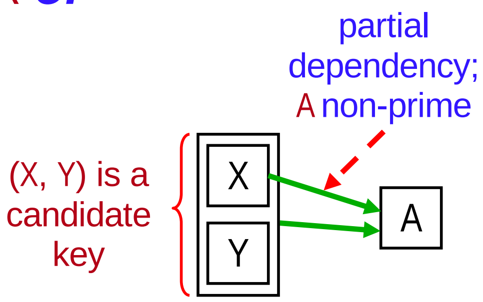
  For each FD: X→A in F^+:

  A ∈ X *(the FD is trivial)* ***or***

  X is not a proper subset of a candidate key for R ***or***

  A is a prime attribute for R.

☞ **A subset of a candidate key cannot determine a non-prime attribute.**



SECOND NORMAL FORM (2NF) EXAMPLE

– make, model, engineSize is a candidate key (it is not a proper subset).

– engineSize is a proper subset of a candidate key.

– fee is a non-prime attribute.

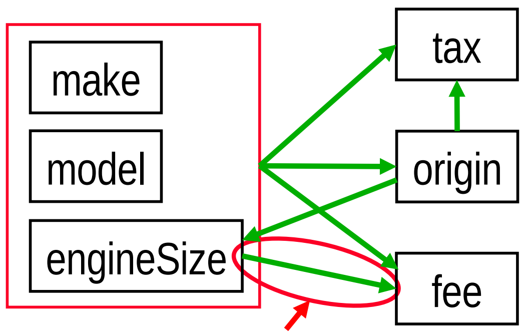
– Hence, the relation schema is not in 2NF due to the FD engineSize→fee.

Note redundancy.

Car

make	model	engineSize	fee	origin	tax
Nissan	Sunny	1	4,000	Japan	90
Fiat	Mirafiori	1	4,000	Italy	85
Honda	Accord	1	4,000	Japan	90
Toyota	Camry	4	7,000	Canada	50
Ford	Mustang	4	7,000	Canada	50
Ford	Mustang	2	5,000	U.S.A.	75
BMW	7.35i	3	6,000	Germany	95
Toyota	Camry	1	4,000	Japan	90

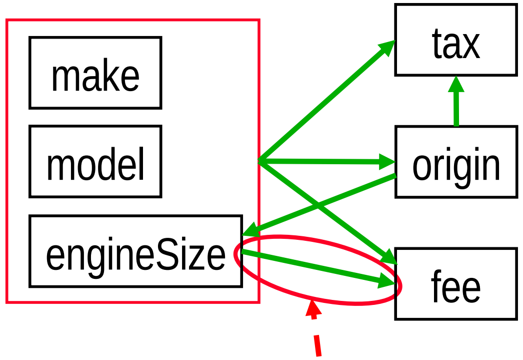
FDs in schema



partial dependency; fee is non-prime

SECOND NORMAL FORM (2NF) EXAMPLE (cont'd)

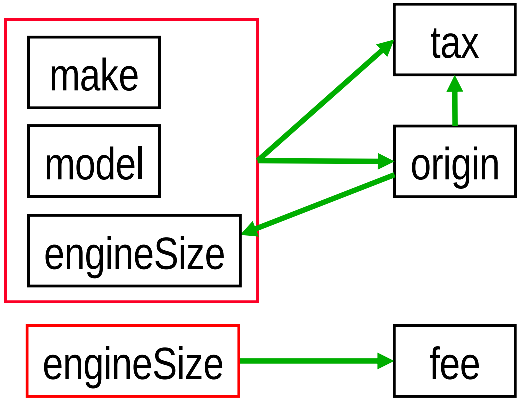
FDs in original schema



partial dependency; fee is non-prime

decompose the schema

FDs in 2NF schemas



decompose the tables

Car

make	model	engineSize	origin	tax
Nissan	Sunny	1	Japan	90
Fiat	Mirafiori	1	Italy	85
Honda	Accord	1	Japan	90
Toyota	Camry	4	Canada	50
Ford	Mustang	4	Canada	50
Ford	Mustang	2	U.S.A.	75
BMW	7.35i	3	Germany	95
Toyota	Camry	1	Japan	90

Licensing

engineSize	fee
1	4000
2	5000
3	6000
4	7000

SECOND NORMAL FORM (2NF) EXAMPLE (cont'd)

Consider the previous update anomalies:

Insert: A new engine size, fee (e.g., 5, 8000). ?

Delete: The instance of BMW 7.35i. ?

Update: The licensing fee for engine size 1. ?

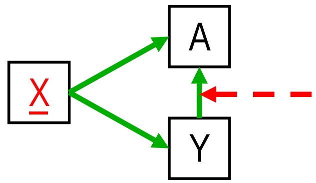
THIRD NORMAL FORM (3NF)

A relation schema is in *Third Normal Form (3NF)* if it is in 2NF and every non-prime attribute is non-transitively dependent on every candidate key.

- R is a relation schema, with set F of FDs.

- R is in 3NF *if and only if*

  For each FD: X→A in F^+:
  - A ∈ X (*trivial FD*) ***or***
  - X is a superkey for R ***or***
  - A is a prime attribute for R.



transitive dependency;
Y is not a superkey;
A is non-prime

☞ **For every FD that does not contain extraneous attributes:**

– the LHS is a candidate key, ***or***

– the RHS is a prime attribute (i.e., it is part of a candidate key).

THIRD NORMAL FORM (3NF) EXAMPLE

– For the FD origin→tax, origin is not a superkey.

– tax is not a prime attribute.

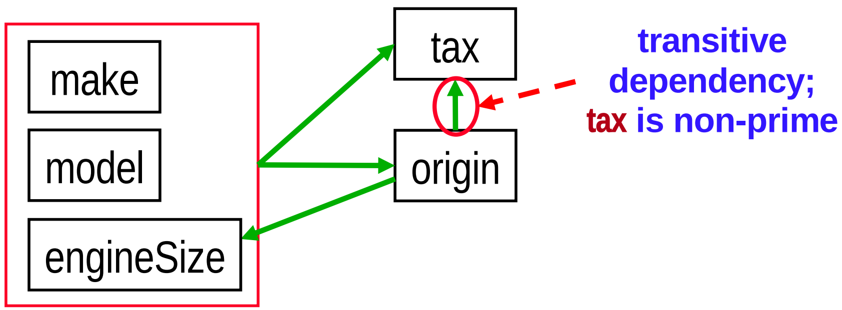
– Hence, the relation schema is not in 3NF due to the FD origin→tax.

Note redundancy.

Car

make	model	engineSize	origin	tax
Nissan	Sunny	1	Japan	90
Fiat	Mirafiori	1	Italy	85
Honda	Accord	1	Japan	90
Toyota	Camry	4	Canada	50
Ford	Mustang	4	Canada	50
Ford	Mustang	2	U.S.A.	75
BMW	7.35i	3	Germany	95
Toyota	Camry	1	Japan	90

FDs in schema



THIRD NORMAL FORM (3NF) EXAMPLE (cont'd)

Consider the following database operations on the Car 2NF relation schema:

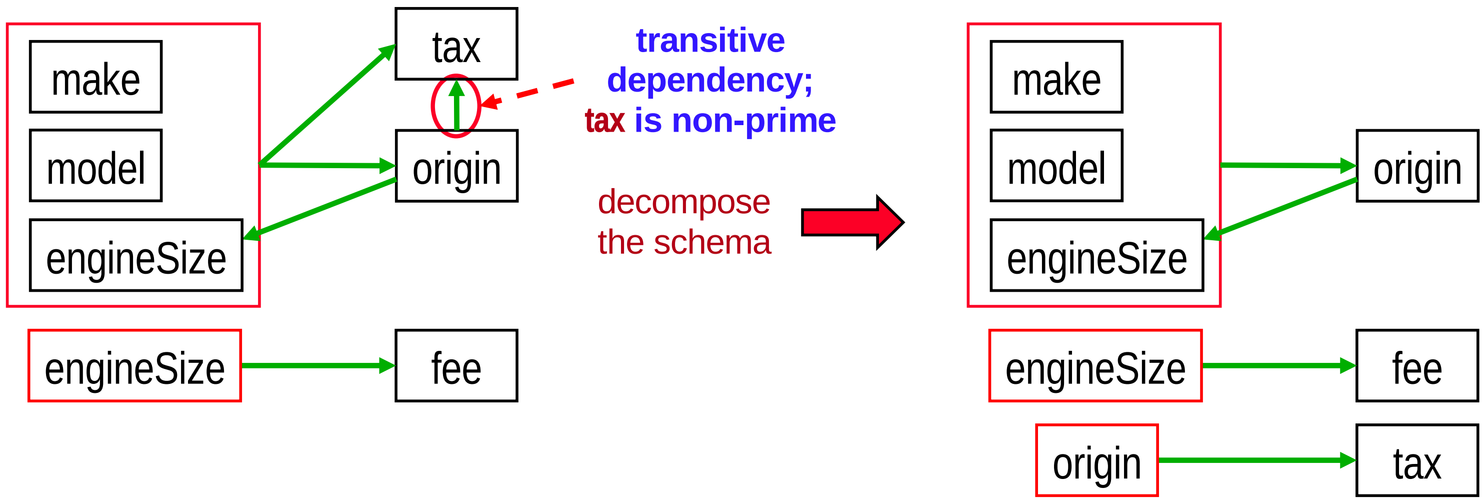
Insert: What if we want to insert tax information for a country that does not yet import any cars to Hong Kong (e.g., Australia, 40)?

Delete: What happens if we delete (BMW, 7.35i)?

Update: What if we want to update the tax rate for cars from Japan?

THIRD NORMAL FORM (3NF) EXAMPLE (cont'd)

FDs in 2NF schemas | **FDs in 3NF schemas**



decompose the tables

Car

make	model	engineSize	origin
Nissan	Sunny	1	Japan
Fiat	Mirafiori	1	Italy
Honda	Accord	1	Japan
Toyota	Camry	4	Canada
Ford	Mustang	4	Canada
Ford	Mustang	2	U.S.A.
BMW	7.35i	3	Germany
Toyota	Camry	1	Japan

ImportTax

origin	tax
Canada	50
U.S.A.	75
Italy	85
Japan	90
Germany	95

Licensing

engineSize	fee
1	4000
2	5000
3	6000
4	7000

If none of the decomposed relations contains a candidate key of the original relation, then add a relation containing one of the candidate keys.

THIRD NORMAL FORM (3NF) EXAMPLE (CONT'D)

Consider the previous update anomalies:

Insert: A new country and tax (e.g., Australia, 40). **?**

Delete: The instance of BMW 7.35i. **?**

Update: Update tax rate for Japan. **?**

BOYCE-CODD NORMAL FORM (BCNF)

A relation schema is in *Boyce-Codd Normal Form (BCNF)* if every determinant (left hand side) of its FDs is a superkey.

- R is a relation schema, with the set F of FDs.
- R is in BCNF *if and only if*

  For each FD: X→A in F^+:
  A ∈ X (*trivial FD*) ***or***
  X is a superkey for R.

☞ **For every FD that does not contain extraneous attributes, the LHS is a candidate key.**

– BCNF tables have no redundancy.

– If a table is in BCNF it is also in 3NF (and 2NF and 1NF).

BOYCE-CODD NORMAL FORM (BCNF): EXAMPLE

– For the FD origin→engineSize, origin is not a superkey.

– Hence, this relation schema is not in BCNF due to the FD origin→engineSize.

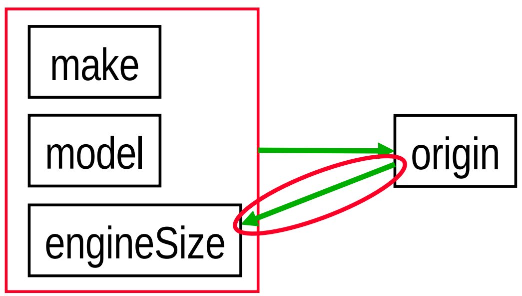
Note redundancy.

Car

<u>make</u>	<u>model</u>	<u>engineSize</u>	origin
Nissan	Sunny	1	Japan
Fiat	Mirafiori	1	Italy
Honda	Accord	1	Japan
Toyota	Camry	4	Canada
Ford	Mustang	4	Canada
Ford	Mustang	2	U.S.A.
BMW	7.35i	3	Germany
Toyota	Camry	1	Japan

Note: Need to use null values if we want to represent an engine size and origin, but do not know the make and model.

FDs in 3NF schema Car



☞ In the FD origin→engineSize, engineSize is a prime attribute.

BOYCE-CODD NORMAL FORM (BCNF): EXAMPLE
(CONT'D)

Update anomalies due to the FD origin→engineSize:

Insert: What if we want to insert a new (engineSize, origin) record (e.g., 5, Korea) for which there is yet no make and model?

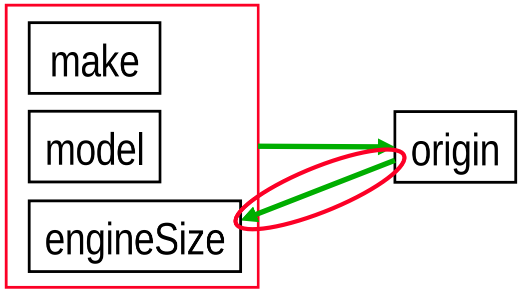
Delete: If we delete all instances of Ford, Mustang, what happens?

Update: What if we want to update the size of engine for cars from Japan?

☞ **How to decompose the schema so that it is lossless?**

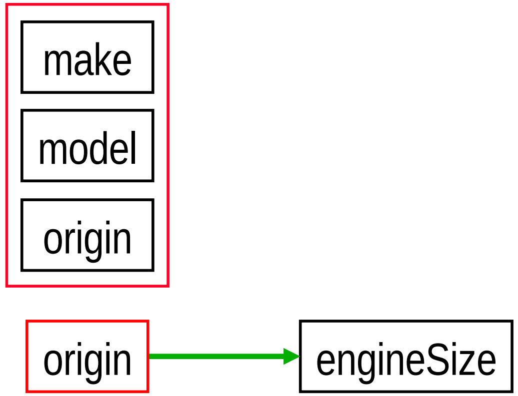
BOYCE-CODD NORMAL FORM (BCNF): EXAMPLE (CONT'D)

FDs in 3NF schema



decompose
the schema

FDs in BCNF schemas



decompose
the tables

Car

make	model	origin
Nissan	Sunny	Japan
Fiat	Mirafiori	Italy
Honda	Accord	Japan
Toyota	Camry	Canada
Ford	Mustang	Canada
Ford	Mustang	U.S.A.
BMW	7.35i	Germany
Toyota	Camry	Japan

Country

origin	engineSize
Italy	1
Canada	4
U.S.A.	2
Germany	3
Japan	1

This decomposition avoids the 3NF problems of redundancy and null values.

BOYCE-CODD NORMAL FORM (BCNF): EXAMPLE (CONT'D)

- We can generate the original relation instance by joining the two fragments, using a full outer join (explained in the next lecture).

Car

make	model	origin
Nissan	Sunny	Japan
Fiat	Mirafiori	Italy
Honda	Accord	Japan
Toyota	Camry	Canada
Ford	Mustang	Canada
Ford	Mustang	U.S.A.
BMW	7.35i	Germany
Toyota	Camry	Japan

⟗

Country

origin	engineSize
Italy	1
Canada	4
U.S.A.	2
Germany	3
Japan	1

=

Car

make	model	engineSize	origin
Nissan	Sunny	1	Japan
Fiat	Mirafiori	1	Italy
Honda	Accord	1	Japan
Toyota	Camry	4	Canada
Ford	Mustang	4	Canada
Ford	Mustang	2	U.S.A.
BMW	7.35i	3	Germany
Toyota	Camry	1	Japan

Is the decomposition dependency preserving?

No. We lose the FD make, model, engineSize→origin.

A relation may not have a dependency preserving BCNF decomposition!

Can we have a dependency preserving decomposition?

No. No matter how we break up the relation schema we lose the FD make, model, engineSize→origin since it involves all the attributes of the original 3NF Car relation.

OBSERVATIONS ABOUT BCNF

- BCNF is the best normal form.

- BCNF avoids the problems of redundancy and all anomalies.

- There is always a lossless decomposition that generates BCNF relation schemas.

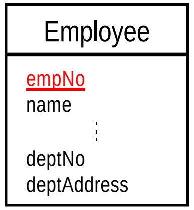
- However, all the functional dependencies may not be preserved.

NORMALIZATION AND THE E-R MODEL

- When an E-R schema is well designed, the relation schemas generated from it should not need further normalization.

- However, in an imperfect design there can be FDs from non-key attributes of an entity to some other attributes of the entity.

- Consider an Employee entity with an FD deptNo→deptAddress.



- A good design would have made department a separate entity.

RELATIONAL MODEL & RELATIONAL DATABASE DESIGN: SUMMARY

Relational Model

Relation (table) – represents entities *and* relationships

Foreign keys – represent relationships

Result of E-R Schema to Relation Schema Reduction

1. A **relation schema** with the same attributes as the original entity:
   - entities with 1:1, 1:N (on the 1-side) or N:M binary relationships.
2. A **relation schema** with the embedded primary key of the entity on the 1-side:
   - entities with a 1:1 binary relationship for one of the entities.
   - entities with a 1:N binary relationship for the entity on the N-side.
3. A **relation schema** with the primary keys of all the entities in the relationship:
   - binary N:M (or higher degree) relationships.

Goal of Normalization

A relational schema that: has clear semantics; minimizes null values; minimizes redundancy; is lossless join; preserves functional dependencies.