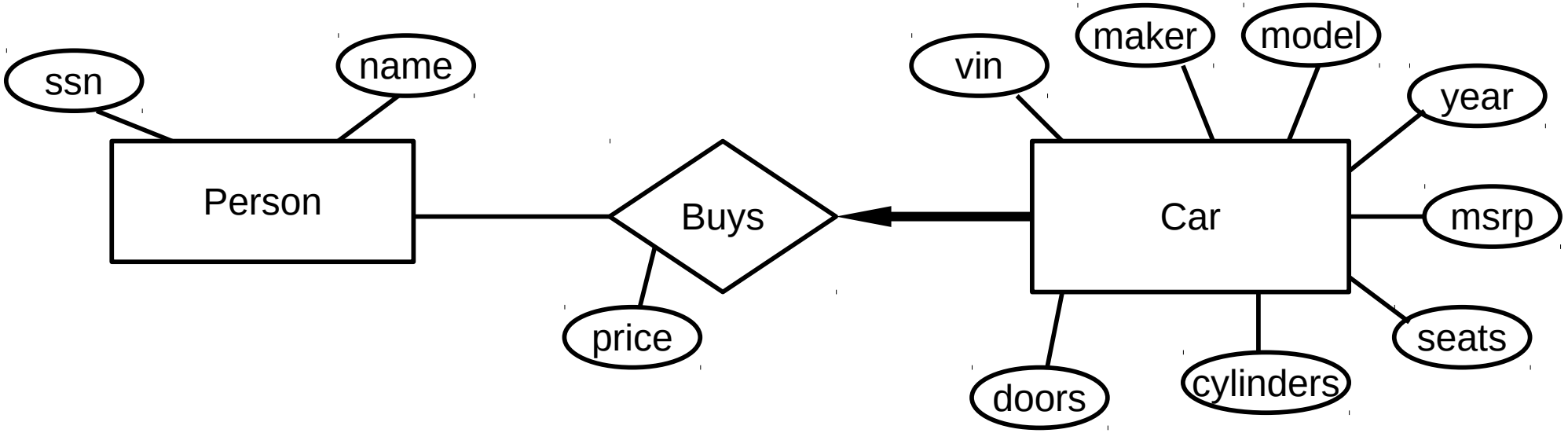
NORMAL FORMS AND REDUNDANCY

- There are several instances of redundancy that can be avoided
- Redundancy might cause several problems (anomalies) in a Database
- If the tables satisfy certain conditions (normal forms) some types of redundancy can be avoided

Suppose that we have the following E-R diagram



After creating tables and using Merge Rule we obtain the following schema:

`Person(ssn,name)`
`Car(vin,maker,model,year,msrp,seats,cylinders,doors,buyer,price)`

Where `buyer` references `Person(ssn)`

Let us look only at the following table:

`Car(vin,maker,model,year,msrp,seats,cylinders,doors,buyer,price)`

- **Deletion Anomaly:**
  Suppose that all the buyers of 2012 Civics are erased from the database. Then we will have no way of knowing how many doors a 2012 Civic has!

- **Insertion Anomaly:**
  How can I insert a car that does not have a buyer?

- **Update Anomaly:**
  If I have to change the MSRP of the 2017 Grand Caravan, I would have to change it in all the tuples of the table.

Let us try to find what might be partially responsible for causing these problems.

We notice that even though `vin` is the primary key, there are some "dependencies" in the table.
For example:
- If we know the `model`, we can get the `maker` (models are trademarks).
- If we know the `model` and the `year`, we can get: `msrp, seats, cylinders,` and `doors`

Think of a way in which we could use these "dependencies" to split the `Car` table into subtables.

Relations, sets and subsets

A relation R can be seen as a set of tuples.

We can use letters to denote subsets of tuples:

$$X \subseteq R, Y \subseteq R$$

We use the notation XY to denote the union of two sets of tuples:

$$XY = X \cup Y$$

For example, if $R(A, B, C, D, E, F)$, $X = \{A, B, C\}$, and $Y = \{D, F\}$ then

$$XY = \{A, B, C, D, F\}$$

Notice that this notation allows us to write $X = ABC$, and $Y = DF$, instead of using set notation.

Functional Dependencies (1)

Definition: We say that two tuples, s and t, agree on a set of attributes X if s and t have the same values for all attributes in X.

EXAMPLE

Suppose that we have the following table:

	A	B	C	D
	1	8	2	4
	1	5	2	4
s	5	1	1	4
	6	2	3	8
t	5	4	1	4
	1	3	2	4
	6	1	3	8

Suppose that s represents tuple number 3 and t represents tuple 5.

Based on the defintion we can say that s and t agree on A since the values of attribute A in tuples 3 and 5 are equal. i.e. $t(A) = s(A) = (5)$

Functional Dependencies (2)

A property that we might find useful is:

Property (Attribute Union):
$s(XY) = t(XY) \iff s(X) = t(X) \wedge s(Y) = t(Y)$

Definition of Functional Dependency
We say that a set of attributes X functionally determines a set of attributes Y (or Y functionally depends on X), denoted by $X \to Y \iff$ whenever s and t agree on X then s and t agree on Y.

In the case of the example:

A	B	C	D
1	8	2	4
1	5	2	4
5	1	1	4
6	2	3	8
5	4	1	4
1	3	2	4
6	1	3	8

Notice that we can see several functional dependencies: $A \to C$, $A \to D$, $C \to D$. Can you find more? Does D determine C?

Back to our Car example

`Car(vin,maker,model,year,msrp,seats,cylinders,doors,buyer,price)`

- If we know the `model`, we can get the `maker` (models are trademarks).
- If we know the `model` and the `year`, we can get: `msrp, seats, cylinders,` and `doors`

We can use the functional dependency notation to write:

$$
\begin{aligned}
model &\rightarrow maker \\
model\ year &\rightarrow msrp\ seats\ cylinders\ doors \\
vin &\rightarrow model\ year\ buyer\ price
\end{aligned}
$$

Notice that vin is the primary key. Does that show by looking at the Functional Dependencies?

Basic Properties of Functional Dependencies

1. Armstrong Axioms

(a) *Reflexivity*:
If $Y \subseteq X$ then $X \to Y$.
Proof:
Assume that $Y \subseteq X$ and s and t agree on X
$\Rightarrow$ s and t agree on Y since $Y \subseteq X$
$\Rightarrow X \to Y$ $\square$

(b) *Augmentation*:
$X \to Y \Rightarrow \forall Z \subseteq R \;\; XZ \to YZ$
Proof:
Assume $X \to Y \;\wedge\; s(XZ) = t(XZ)$
$\Rightarrow s(X) = t(X) \;\wedge\; s(Z) = t(Z)$
$\Rightarrow$ (since $X \to Y$) $s(Y) = t(Y) \;\wedge\; s(Z) = t(Z)$
$\Rightarrow s(YZ) = t(YZ)$
$\Rightarrow XZ \to YZ$ $\square$

(c) *Transitivity*:
$X \to Y \;\wedge Y \to Z \;\Rightarrow\; X \to Z$
Proof:
Assume that $X \to Y \;\wedge Y \to Z$ and that s and t agree on X
$\Rightarrow$ (since $X \to Y$) s and t agree on Y
$\Rightarrow$ (since $Y \to Z$) s and t agree on Z
$\Rightarrow X \to Z$ $\square$

Basic properties of Functional Dependencies (2)

2. **Key:**
   K is a candidate key of $R \Leftrightarrow K$ is a minimal subset of R such that $K \rightarrow R$

3. **Superkey:**
   Y is a superkey of $R \Leftrightarrow K \subseteq Y$ and K is a candidate key of R

4. **Union/decomposition**:
   $X \rightarrow Y \ \wedge \ X \rightarrow Z \ \Leftrightarrow X \rightarrow YZ$
   Proof ($\Rightarrow$):
   Assume that $X \rightarrow Y \ \wedge \ X \rightarrow Z$ and that s and t agree on X
   $\Rightarrow s(Y) = t(Y) \ \wedge s(Z) = t(Z)$
   $\Rightarrow s(YZ) = t(YZ)$
   $\Rightarrow X \rightarrow YZ \ \square$
   Proof ($\Leftarrow$):
   Assume that $X \rightarrow YZ$ and that s and t agree on X
   $\Rightarrow s(YZ) = t(YZ)$
   $\Rightarrow s(Y) = t(Y) \ \wedge \ s(Z) = t(Z)$
   $\Rightarrow X \rightarrow Y \ \wedge \ X \rightarrow Z \ \square$

Basic properties of Functional Dependencies (3)

The decomposition property implies that the right hand side (RHS) of a functional dependency can be divided, for example

$$AB \rightarrow CD$$

can be divided in two functional dependencies:

$$\begin{aligned} AB &\rightarrow C \\ AB &\rightarrow D \end{aligned}$$

IMPORTANT: The left hand side of functional dependencies cannot be divided. For example, in the case of our car example, dividing the LHS of the functional dependency

$$model \;\; year \rightarrow msrp$$

makes no sense, since *model* alone (without *year*) cannot functionally determine *msrp*.

Closure

Definition: Let F be a set of functional dependencies. The closure of a set of attributes X, denoted by X^+ is defined as:

$$X^+ = \{A \in R | X \rightarrow A\}$$

In other words, the closure of X is the set of all attributes that functionally depend on X based on a given set of functional dependencies F.

Algorithm to compute X^+

Input: A set of functional dependencies F
A set of attributes X

X^+:=X
Repeat
  For each dependency $V \rightarrow W \in F$
    If $V \subseteq X^+$ then
      X^+:=$X^+ \cup W$
    endif
  endfor
Until X^+ does not change

Example

Given $R(A, B, C, D)$ and $F = \{A \to B,\ B \to C\}$

1. Find A^+
   We start with $A^+ = \{A\}$
   For each dependency:
   $A \to B$: $A \in A^+$, therefore $A^+ := \{A, B\}$
   $B \to C$: $B \in A^+$, therefore $A^+ := \{A, B, C\}$
   If we try again for each dependency, A^+ will not change
   Therefore, $A^+ = \{A, B, C\}$

2. Find AD^+
   We start with $AD^+ = \{A, D\}$
   For each dependency:
   $A \to B$: $A \in AD^+$, therefore $AD^+ = \{A, D, B\}$
   $B \to C$: $B \in AD^+$, therefore $AD^+ = \{A, D, B, C\}$
   Therefore, $AD^+ = \{A, B, C, D\}$

3. To find **all** possible functional dependencies we must compute the closure of every possible combination of attributes: A^+, B^+, C^+, D^+, AB^+, AC^+, AD^+, BC^+, BD^+, CD^+, ABC^+, ABD^+, ACD^+, BCD^+, and $ABCD^+$.

Finding Keys

We are given a relation R and a set of functional dependencies F. We are asked to find one or all keys, i.e. we need to find minimal sets of attributes that determine all the attributes in R.

In order to make the process of finding keys faster, let us notice the following properties:

- An attribute that appears only on the left hand side of functional depencies must belong to a key.
- An attribute that does not appear on any functional dependency must belong to the key.
- An attribute that appears only on the right hand side of functional dependencies cannot be part of any key.
- An attribute that appears on the right hand side of some functional dependencies and on the left hand side of others, might or might not belong to a key.

So we divide the set of attributes in three sets:

None or Left	Both	Right
L	B	R

Notice that L, B, and R are sets of attributes. To find the keys we start by computing L^+, if $L^+ = R$, then L is the key, and there is no other key. If L is not the key, then we will try LX^+ for each $X \subseteq B$, in order of cardinality, until we find all of the keys.

Prime and non-prime Attributes

An attribute is prime if it belongs to some candidate key.

Example 1 (keys)

Given $R(A, B, C, D)$ and $F = \{A \to B,\ B \to C\}$, find all the keys, and determine non-prime attributes.

We first divide the attributes into the sets described previously:

None or Left	Both	Right
A, D	B	C

We now compute AD^+:

AD^+
$ADBC$

Since AD determines every attribute in R then AD is the key. Notice that in this case we do not try those attributes in the "Both" column since adding anything to AD would make the set not minimal.
We can also try to remove eithr A or D from the key, and we will see that it is minimal.
Since B and C are not part of any key, then B and C are non-prime attributes.

Example 2 (keys)

Given $R(A, B, C, D)$ and $F = \{A \to C,\ A \to D,\ C \to A\}$, find all the keys.

None or Left	Both	Right
B	AC	D

$$\frac{B^+}{B} \qquad \frac{BA^+}{BACD} \qquad \frac{BC^+}{BCAD}$$

So we have two keys: AB and BC.
A, B, and C are prime attributes, and D is non-prime.

Back to anomalies

We started this entire section with the objective of eliminating anomalies. Let us see if we can use some of the tools that we have developed so far to help us with this task.

Let us go back to our "Car" example and itentify what is causing the anomalies:

Given: $Car(vin, maker, model, year, cylinders, doors, buyer, price)$ and the functional dependencies:

$$\begin{aligned} model &\rightarrow maker \\ model\ \ year &\rightarrow msrp\ \ seats\ \ cylinders\ \ doors \\ vin &\rightarrow model\ \ year\ \ buyer\ \ price \end{aligned}$$

Back to anomalies (2)

We can compute the key, and determine that the only key is vin, however, there is (at least) one functional dependency,

$$model \quad year \rightarrow msrp \quad seats \quad cylinders \quad doors$$

that creates redundancy since there might be several cars of the same $model$ and $year$ in the table, for which the attributes $msrp$, $seats$, $cylinders$, and $doors$ will be repeated!

The solution could be to split the table into two tables based on this "offending" functional dependency in the following way:
$R_1(model, year, msrp, seats, cylinders, doors)$
$R_2(model, year, vin, maker, buyer, price)$

The question now is: Are we sure that if we decompose the Car table in this way, all the data of the original table can be recovered from R_1 and R_2?

Table decomposition

A relation $R(X)$ can be decomposed into two relations $S(Y)$, $Y \subseteq X$ and $T(Z)$, $Z \subseteq X$. The instances of S and T can be computed by using the projection operator on a given instance of R.

$$\begin{aligned} S &= \Pi_Y(R) \\ T &= \Pi_Z(R) \end{aligned}$$

Example: Let $R(ABCD)$ be a relation with functional dependencies $A \to C$, $A \to D$, and $C \to D$.

Let us decompose $R(ABCD)$ into $S(BC)$ and $T(BAD)$, where the instances of the decoposed tables are: $S = \Pi_{BC}R$ and $T = \Pi_{BAD}R$:

$R=$

A	B	C	D
1	8	2	4
1	5	2	4
5	1	1	4
6	2	3	8
5	4	1	4
1	3	2	4
6	1	3	8

$S=$

B	C
8	2
5	2
1	1
2	3
4	1
3	2
1	3

$T=$

B	A	D
8	1	4
5	1	4
1	5	4
2	6	8
4	5	4
3	1	4
1	6	8

Lossless Decomposition

Notice that if we try to recover the data originally stored in R we should be able to do so using the natural join operator: $R = S \bowtie T$, but if we try to compute R in this way, we obtain tuples that were NOT in the original R from which S and T were obtained.

As an exercise compute the entire $S \bowtie T$ table, that includes the tuple: $(6, 1, 1, 8)$.

This decomposition is called "lossy" because it does not preserve the data of the original table.
However, if we use functional dependencies to decompose a table, we can decompose it in such a way that the original data can be recovered from the decomposed tables. This decomposition is said to be **lossless**.

Theorem
Given a relation $R(XYZ)$ and a set of functional dependencies F, if $X \rightarrow Y \in F^+$ then R can be losslessly decomposed into $S(XY)$ and $T(XZ)$.

Example (Lossless decomposition)

If we decompose the instance of R of our previous examples based on functional dependency: $A \rightarrow C$, as described in the previous theorem, we obtain: $S(AC)$, and $T(BAD)$. Computing the instances of S and T using the projection operator yields:

$R=$

A	B	C	D
1	8	2	4
1	5	2	4
5	1	1	4
6	2	3	8
5	4	1	4
1	3	2	4
6	1	3	8

$S=$

A	C
1	2
5	1
6	3

$T=$

B	A	D
8	1	4
5	1	4
1	5	4
2	6	8
4	5	4
3	1	4
1	6	8

As an exercise, compute $S \bowtie T$ to verify that it is identical to R, i.e. the decomposition based on a functional dependency is lossless.