What makes good database design?

☒ Consider this table schema for information about wholesale book sales by publishers to bookstores (it sits on its side):

	attributes	a sample row
	PublisherId	1481MK
B	**PublisherName**	Morgan Kaufmann
O	**PublisherAddress**	San Francisco, CA
O	**StoreId**	ONT392
K	**StoreName**	UofO Bookstore
	StoreAddress	Ottawa, Ontario
S	**BookId**	ISBN 1-55860-219-4
A	**BookAuthors**	Patrick O’Neil
L	**BookTitle**	Database Principles...
E	**BookListPrice**	68.00
S	**SaleQuantity**	90
	SaleDate	1996/07/15

This schema causes numerous problems:

• data are unnecessarily repeated many times,

• update, delete and insert anomalies may occur.

Repetition

Complete data about the publisher, the store and the book must be entered for every new sale. This is not only wasteful, but also likely to cause inconsistencies due to small errors in data.

Anomalies

☒ Consider how the following events in the wholesale book sales business would be reflected in this table.

• A publisher changes its address.

• A store changes its name.

• A new publisher appears on the market.

• A publisher goes out of business.

• The list price of a book changes.

There will be data changes in many rows (inconsistencies may and will happen). Missing information will cause the table to contain many null values in essential places (for example, publisher unknown).

Here is a better (though not ideal) design, which avoids most of the problems we have discussed.

PUBLISHERS

<u>PublisherId</u>	1481MK
PublisherName	Morgan Kaufmann
PublisherAddress	San Francisco, CA

STORES

<u>StoreId</u>	ONT392
StoreName	UofO Bookstore
StoreAddress	Ottawa, Ontario

BOOKS

<u>BookId</u>	ISBN 1-55860-219-4
BookAuthors	Patrick O’Neil
BookTitle	Database Principles...
BookListPrice	68.00

SALES

<u>PublisherId</u>	1481MK
<u>StoreId</u>	ONT392
<u>BookId</u>	ISBN 1-55860-219-4
SaleQuantity	90
SaleDate	1996/07/15

Functional dependencies

Let **R**($A_1, ..., A_N$) be the schema of a relation.

We have a functional dependency on **R** if one subset of $A_1, ..., A_N$ determines another subset of $A_1, ..., A_N$ in every instance of the table.

This is actually a constraint that ensures uniqueness of certain data in the table. (This constraint should be enforced by the DBMS.)

The functional dependency

$$A_{i_1} \ldots A_{i_k} \rightarrow A_{j_1} \ldots A_{j_p}$$

holds on table **R** if its instance never contains two different tuples t, u such that

$$t[\, A_{i_1}, ..., A_{i_k} \,] = u[\, A_{i_1}, ..., A_{i_k} \,]$$

and $t[\, A_{j_1}, ..., A_{j_p} \,] \neq u[\, A_{j_1}, ..., A_{j_p} \,]$

In other words, the values of $A_{j_1}, ..., A_{j_p}$ are a *function* of the values of $A_{i_1}, ..., A_{i_k}$

☒ Here are a few examples of FDs for the table of publishers, stores and books.

PublisherId → PublisherName, PublisherAddress

(a publisher can't have two names or addresses)

StoreId → StoreName, StoreAddress

(the same for the store)

BookId → BookAuthors, BookTitle, BookListPrice

(a book has one title, authors and current price)

PublisherId, StoreId, BookId → SaleQuantity, SaleDate

(you can't have two sales of the same book to the same store in one day)

Note that each FD here has to its left a primary key of one of the smaller tables. This is not accidental!

In fact, the definition says that any superkey **K** determines the values of all attributes. That is:

$$\mathbf{K} \rightarrow A_1 \ldots A_N$$

and, for all j, $\mathbf{K} \rightarrow A_j$.

FDs, obviously, allow us to express constraints other than trivial mapping from superkeys to all attributes.

☒ For example, here is the schema and a sample tuple for the table of classes:

CLASS

Course	Section	Day	Time	Hours	Roomnum
ALG	1	THU	13:00	1	LR#1

The primary key has four attributes, but we can also state this dependency:

Course → Hours

(the teaching unit duration is the same for all sections of the course, regardless of their time and location)

An FD is defined for a table at the database design stage, after the designer has considered the semantics of all attributes. The choice of dependencies should reflect the underlying world model. Here is an example of what would not be a good FD (why?):

Course, Section, Day → Time

We are interested not only in the specified FDs but also in all other FDs that logically follow from them. We can find out what they are by computing the closure $\mathbf{F}^+$ of an FD set **F** under certain rules of logical reasoning.

Note that $\mathbf{F} \subseteq \mathbf{F}^+$.

We can also use FDs to choose superkeys.

Armstrong's axioms are three inference rules that are sound and complete. They

- guarantee that we only derive from **F** FDs that are in the closure $\mathbf{F}^+$,
- allow us to derive from **F** any FD in $\mathbf{F}^+$.

In the rules, the letters α, β, γ, δ will denote lists of attributes for the schema $\mathbf{R}(A_1, ..., A_N)$:

If $\alpha = A_{i_1} ... A_{i_k}$ and $\beta = A_{j_1} ... A_{j_p}$
then $\alpha\, \beta = A_{i_1} ... A_{i_k}\, A_{j_1} ... A_{j_p}$

The reflexivity rule

A set of attributes functionally determines its subsets. That is, for any α, β:

$$\boxed{\alpha\ \beta \rightarrow \alpha}$$

☒ One example should be enough:

BookId, BookAuthors, BookTitle $\rightarrow$ BookAuthors, BookTitle

The augmentation rule

This rule deals with function-preserving extensions of a FD.

If the following holds:

$$\alpha \rightarrow \gamma$$

then for any additional attributes β:

$$\alpha\ \beta \rightarrow \gamma\ \beta$$

☒ For example:

Course, Section $\rightarrow$ Section, Hours

(because we have Course $\rightarrow$ Hours)

The transitivity rule

This rule deals with composition of functions:

$$\boxed{\begin{array}{ll} \text{if} & \alpha \rightarrow \beta \quad \text{and} \quad \beta \rightarrow \gamma \\ \text{then} & \alpha \rightarrow \gamma \end{array}}$$

Armstrong's axioms are the basic inference rules. Other useful rules can be derived.

Pseudo-transitivity rule

$$\boxed{\begin{array}{ll} \text{if} & \alpha \rightarrow \beta \quad \text{and} \quad \beta\,\delta \rightarrow \gamma \\ \text{then} & \alpha\,\delta \rightarrow \gamma \end{array}}$$

Union rule (combining right-hand sides):

$$\boxed{\begin{array}{ll} \text{if} & \alpha \rightarrow \beta \quad \text{and} \quad \alpha \rightarrow \gamma \\ \text{then} & \alpha \rightarrow \beta\,\gamma \end{array}}$$

Decomposition rule (narrowing a function):

$$\boxed{\begin{array}{ll} \text{if} & \alpha \rightarrow \beta \quad \text{and} \quad \gamma \subseteq \beta \\ \text{then} & \alpha \rightarrow \gamma \end{array}}$$

These inference rules allow us to reason about FDs formally without ever looking at any tuples, and without worrying about the semantics of the functions.

We may want to find (for a given set of FDs) a minimal set of dependencies that captures the same constraints as the original set. This helps enforce the constraints efficiently.

Decomposition

We have seen how one large table (BOOK SALES) can be successfully replaced by a few smaller and better organized tables. This replacement operation is called decomposition.

We distribute the attributes in a poorly designed schema into two or more smaller schemas. Some attributes should be repeated to make it possible to compute joins, which would recreate the information originally stored in the table. This may be a problem if decomposition is not correct.

☒ Consider, for example, this schema:

ORDERS(OrdId, Name, Suppl, Item, Quant)

It can be "cut" into two tables:
ON(OrdId, Name) and NSIQ(Name, Suppl, Item, Quant)

If we need to associate, let's say, attributes OrdId and Quant, we must perform a join on the attribute Name.

Here's the problem: the join ON ⋈ NSIQ contains more tuples than we had in ORDERS, and the added tuples are meaningless. Suppose we start with 6 tuples:

OrdId	Name	Suppl	Item	Quant
1	Brooks, B.	Forest	Granola	5
2	Brooks, B.	Forest	Flour	10
3	Robins, R.	GoodFood	Granola	3
4	Harts, W.	GoodFood	Milk	5
5	Robins, R.	SunFarm	Carrots	2
6	Robins, R.	SunFarm	Lettuce	8

By cutting this table on Name, we get two narrower 6-tuple tables. The first:

ON

OrdId	Name
1	Brooks, B.
2	Brooks, B.
3	Robins, R.
4	Harts, W.
5	Robins, R.
6	Robins, R.

And the second table:

Name	Suppl	Item	Quant
Brooks, B.	Forest	Granola	5
Brooks, B.	Forest	Flour	10
Robins, R.	GoodFood	Granola	3
Harts, W.	GoodFood	Milk	5
Robins, R.	SunFarm	Carrots	2
Robins, R.	SunFarm	Lettuce	8

NSIQ

The join ON $\bowtie$ NSIQ contains 14 tuples.

Here are some spurious tuples in the join:

1	Brooks, B.	Forest	Granola	10
2	Brooks, B.	Forest	Flour	5
3	Robins, R.	SunFarm	Carrots	2
3	Robins, R.	SunFarm	Lettuce	8

Less discriminating data (too much to choose from) mean loss of information. This happens because the “cut” was on an insufficiently discriminating attribute—essentially, not on a key or on part of a key.

A decomposition from which we can revert to the original table by performing a join is called a lossless-join decomposition.

A *schema* $\mathbf{R}(A_1, ..., A_N)$ is decomposed into:

$$\mathbf{R}_1(\alpha), ..., \mathbf{R}_k(\beta)$$

if the new schemas contain all attributes:

$$\alpha \cup ... \cup \beta = \{A_1, ..., A_N\}.$$

Normally, we expect the smaller schemas to overlap. In particular, if we decompose into two schemas:

$$\alpha = \alpha1 \cup \alpha2,$$

we expect that

$$\alpha1 \cap \alpha2 \neq \emptyset.$$

A instance R of the schema $\mathbf{R}(A_1, ..., A_N)$ is decomposed simply by doing projections:

$R_1 = R [\alpha]$

......

$R_k = R [\beta]$

It is always true that $R \subseteq R_1 \bowtie \ldots \bowtie R_k$

Loss of information occurs when

$$R \subset R_1 \bowtie \ldots \bowtie R_k$$

☒ Example:

ON(OrdId, Name)

NSIQ(Name, Suppl, Item, Quant)

ORDERS(OrdId, Name, Suppl, Item, Quant)

ON ⋈ NSIQ has more tuples than ORDERS.

Suppose we have constraints, such as FDs, on the original schema and on the schemas in the decomposition. We can show good properties of the decomposition, and even have an algorithm to find a good decomposition.

Let **R**(A_1, ..., A_N) be a schema, and let **F** be a set of FDs for **R** with the closure $\mathbf{F}^+$.

$\mathbf{R}_1(\alpha)$, $\mathbf{R}_2(\beta)$ is a lossless-join decomposition if the dependency set $\mathbf{F}^+$ contains one or both of the following functional dependencies:

$$\alpha \cap \beta \rightarrow \alpha \qquad \alpha \cap \beta \rightarrow \beta$$

☒ Example (ORDERS again):

α = OrdId Name

β = Name Suppl Item Quant

We would need either of these dependencies:

Name $\longrightarrow$ OrdId Name

or, after simplification, Name $\longrightarrow$ OrdId

Name $\longrightarrow$ Name Suppl Item Quant

or, after simplification, Name $\longrightarrow$ Suppl Item Quant

These FDs both do not hold: the decomposition is lossy.

☒ On the other hand, if we choose

α = OrdId Name and β = OrdId Suppl Item Quant

we want one of these FDs to hold:

OrdId $\longrightarrow$ Name and OrdId $\longrightarrow$ Suppl Item Quant

In fact, both FDs are semantically justified.