Normalization

Preliminaries

Conceptual schema design gives a set of relation schemas and integrity constraints (IC) that can be regarded as a good starting point for the final database design. This initial design must be refined by taking the IC's into account more fully than is possible just with the ER model constructs, and also by considering performance criteria and typical workloads.

1 Problem

Storing the same information **redundantly**, that is, in more than one place within a database, can lead to several problems:

- **Redundant storage**: some information is stored repeatedly.
- **Update anomalies**: if one copy of a repeated data is updated, an inconsistency is created unless all copies are similarly updated.
- **Insertion anomalies**: it may not be possible to store some information unless some other information is stored as well.
- **Deletion anomalies**: it may not be possible to delete some information without losing some other information as well.

Supplier

Name	Address	Product	Price
Smith	St. John Str. 5	pen	1.00
Smith	St. John Str. 5	brush	0.50
Smith	St. John Str. 5	comb	0.50
...	...	...	...
Gray	Moulin Rouge 1	pen	1.15
Gray	Moulin Rouge 1	pencil	1.10
Gray	Moulin Rouge 1	comb	0,75
...	...	...	...

Properties of the relation schema `Supplier`:

- redundancy: some information is stored multiple times, e.g. an address of a supplier,
- consistency: redundancy leads to potential inconsistency,
- insertion anomaly: it is impossible to insert a new supplier (key constraint),
- delete anomaly: it is impossible to delete information concerning supply without deleting information concerning a supplier,
- update anomaly: the address in the first tuple could be updated without making a similar change in the second tuple.

What we can do to solve the problem?

Use of decomposition

Supplier

Name	Address
Smith	St. John Str. 5
Gray	Moulin Rouge 1
...	...

Supply

Name	Product	Price
Smith	pen	1.00
Smith	brush	0.50
Smith	comb	0.50
Gray	pen	1.15
Gray	pencil	1.10
Gray	comb	0,75
...	...	...

The essential idea is that many problems arising from redundancy can be addressed by replacing a relation with a collection of "smaller" relations. Each of the smaller relations contains a (strict subset of the attributes of the original relation.

We refer to this process as **decomposition** of the larger relation into smaller relations.

Problems Related to Decomposition

Decomposing a relation schema can create more problems than it solves. Two important questions are:

1. What problems (if any) does a given decomposition cause?
2. Do we need to decompose a relation schema?

Ad. 1

Two properties of decompositions are of particular interest: the **lossless-join property** and the **dependency-preservation property**.

The **lossless-join property** enables us to recover any instance of the decomposed relation from corresponding instances of the smaller relations.

The **dependency-preservation property** enables us to enforce any constraint on the original relation by simply enforcing some constraints on each of the smaller relations. That is, we need not perform joins of the smaller relations to check whether a constraint on the original relation is violated.

Supplier

Name	Address
Smith	St. John Str. 5
Gray	Moulin Rouge 1
...	...

Supply

Name	Product	Price
Smith	pen	1.00
Smith	brush	0.50
Smith	comb	0.50
Gray	pen	1.15
Gray	pencil	1.10
Gray	comb	0,75
...	...	...

Query: "Find the address of all suppliers that supply pens"

To answer the question it is necessary to join both relations:

Supplier (join) Supply $\Rightarrow$ original Supplier

Debts

Bank_Branch	Account	Amount	Name
III	17	1	Smith
III	23	1,5	Brown
V	5	3	Gray
V	10	1,5	White
I	77	3	Kerr
II	17	1	O'Neil
II	21	1,3	Snake

decomposition

Clients

Amount	Name
1	Smith
1,5	Brown
3	Gray
1,5	White
3	Kerr
1	O'Neil
1,3	Snake

Accounts

Bank_Branch	Account	Amount
III	17	1
III	23	1,5
V	5	3
V	10	1,5
I	77	3
II	17	1
II	21	1,3

Recover of the original relation

- spurious tuples -

Clients (join) Accounts $\not\Rightarrow$ Debts

Debts

Bank_Branch	Account	Amount	Name
III	17	1	Smith
III	23	1,5	Brown
III	17	1	O'Neil
III	23	1,5	White
V	5	3	Kerr
V	10	1,5	Brown
V	5	3	Gray
V	10	1,5	White
I	77	3	Kerr
I	77	3	Gray
II	17	1	O'Neil
II	17	1	Smith
II	21	1,3	Snake

Incorrect decomposition - we loose information
(spurious tuples)

Ad. 2

With respect to the second question, several **normal forms** have been proposed for relations. If a relation is in one of these normal forms, we know that certain kinds of problems cannot arise. Considering the normal form of a given relation schema can thus help us to decide whether or not to decompose it further. If we decide that a relation schema must be decomposed further, we must choose a particular decomposition.

Basic definitions

We are given a relation schema $R = \{A_1, A_2, ..., A_n\}$. A **superkey** for the relation schema R is a set of attributes $S \subseteq R$ that uniquely identifies each tuple of relation instance r(R).

In other words, if u and v are two distinct tuples of r(R) then $u(S) \neq v(S)$; that is, there will always exist at least one attribute, $A_i \in S$ such that $u(A_i) \neq v(A_i)$.

Comments:
We use the notation u(X) to refer to the projection of tuple u onto the set of attributes in X.

A **key** K for a relation schema R is a "minimal" superkey for R, $K \subseteq R$, such that there is no proper subset $K' \subset K$ that is a superkey for R.

This property assures us that a key is a minimal set of attributes with property of a superkey.

Candidate keys:

- A primary key
- Secondary keys

Attributes:

- Primary attributes – an attribute A is a primary attribute iff it belongs to a key.
- Secondary attributes - an attribute A is a secondary attribute iff it does not belongs to any key.

Functional Dependencies

A functional dependency (FD) is a kind of IC that generalizes the concept of a key.

Let R be a relation schema and let X and Y be nonempty sets of attributes in R. We say that an instance r of R **satisfies** the FD $X \rightarrow Y$ if the following holds for every pair of tuples u and v in r:

If u(X) = v(X) then u(Y) =v(Y)

An FD $X \rightarrow Y$ essentially says that if two tuples agrees on the values in attribute X, that must also agree on the values in attribute Y.

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a2	b1	c3	d1

A relation instance that satisfies $AB \rightarrow C$

Functional Dependencies (cont.)

A **legal instance** of a relation must satisfy all specified ICs, including all specified FDs. A functional dependency defines a dependency between attributes of a relation schema. This dependency has a semantic character, i.e. it has to be fulfilled by all instances of the relation schema.

A functional dependency is a property of a relation schema not of a relation instance!!!

`Examples:`

Name $\rightarrow$ Address
Name, Product $\rightarrow$ Price