Normalization

- *Instructor*

  Dr. Sanam Shahla Rizvi

  PhD in Information and Communication from Ajou University, Korea

Normalization

- Normalization is the process of efficiently organizing data in a database.
- It is a foundation for relational database design.
- It involves removing redundant data from relational tables by decomposing a relational table into smaller tables.

Objectives

- Eliminating redundant data (storing the same data in more than one table).
- Ensuring data dependencies make sense (only storing related data in a table).

Database Normalization

- **Proposed by Codd (1972)**
- **Introduced 3 normal forms, the first, second and third normal form**
- **A stronger definition of 3NF - called Boyce-Codd normal form (CDNF) was proposed later**
- **Later, 4NF and 5NF were proposed**

The minimum, and most common, goal is to achieve 3NF.

Database Normalization

Normalization **Is the process of analyzing the given relational schema based on its functional dependencies and keys to achieve the desirable properties of:**

- **Minimizing redundancy**
- **Minimizing the insertion, deletion, and updating anomalies**
- **Minimize data storage**
- **Unsatisfactory relation schema that do not meet a given normal form test are decomposed into smaller relational schemas that meet the test and hence possess the desired properties.**
- **Key Concepts in normalization are Functional Dependency and keys.**

The issue of keys

- The goal of database normalization is to ensure that every non-key column in every table is directly dependent on the key, the whole key and nothing but the key.

Candidate and Primary Keys

- **Superkey** – a set of one or more attributes that uniquely identifies a specific instance of an entity.

- **Candidate key** – any subset of the attributes of a superkey that is also a superkey and not reducible to another superkey.

- **Primary key** – a selection from the set of candidate keys -used to index a relation.

Primary Keys

- Every relation (entity) must have a **primary key.**
- To qualify as a primary key, an attribute must have the following properties:
- it must have a non-null value for each instance of the entity,
- the value must be unique for each instance of an entity,
- the values must not change or become null during the life of each entity instance.

Composite Keys

- Sometimes more than one attribute is required to uniquely identify an entity.
- A primary key made up of more than one attribute is known as a *composite key*.

Foreign Keys

- A foreign key identifies a column or a set of columns in one (referencing) table that refers to a column or set of columns in another (referenced) table.
- Columns in the referencing table must be the primary key or other candidate key in the referenced table.
- A foreign key completes a relationship by identifying the parent entity.

Full Functional Dependence

- Applies to tables with composite keys.
- Full functional dependence means that when a primary key is composite, then the other columns must be identified by the entire key and not just some of the columns that make up the key.

Anomalies

- If we insert a new customer, which has no invoices, we have to insert null values for all attributes relating to invoice (insert anomaly).
- If we insert a new invoice for a customer, we have to insert customer details (name, address, etc) correctly so that it will be consistent with the existing values (insert anomaly).
- If we delete an invoice for a customer and that customer happen to be to have only one invoice, the information concerning this customer will be lost from the database (delete anomaly).
- If we update the address of a customer, we have to update all invoices for that customer as well (update anomaly).

First Normal Form (1NF)

Each attribute must be atomic

- *No repeating columns within a row.*
- *No multi-valued columns.*

- ***Therefore,***

The fact that all our data and columns are atomic and we have a primary key means that we are in 1NF!

1NF

Employee (unnormalized)

emp_no	name	dept_no	dept_name	skills
1	Kevin Jacobs	201	R&D	C, Perl, Java
2	Barbara Jones	224	IT	Linux, Mac
3	Jake Rivera	201	R&D	DB2, Oracle, Java

Employee (1NF)

emp_no	name	dept_no	dept_name	skills
1	Kevin Jacobs	201	R&D	C
1	Kevin Jacobs	201	R&D	Perl
1	Kevin Jacobs	201	R&D	Java
2	Barbara Jones	224	IT	Linux
2	Barbara Jones	224	IT	Mac
3	Jake Rivera	201	R&D	DB2
3	Jake Rivera	201	R&D	Oracle
3	Jake Rivera	201	R&D	Java

Second Normal Form (2NF)

- A database in 2NF must also be in 1NF:
  - Data must be atomic.
  - Every row must have a unique primary key.
- Plus:
  - Non key attributes are fully functionally dependent on key attributes.

Example

CustID	FirstName	LastName	Address	City	State	Zip
1	Bob	Smith	123 Main St.	Tucson	AZ	12345
2	John	Brown	555 2nd Ave.	St. Paul	MN	54355
3	Sandy	Jessop	4256 James St.	Chicago	IL	43555
4	Maria	Hernandez	4599 Columbia	Vancouver	BC	V5N 1M0
5	Gameil	Hintz	569 Summit St.	St. Paul	MN	54355
6	James	Richardson	12 Cameron Bay	Regina	SK	S4T 2V8
7	Shiela	Green	12 Michigan Ave.	Chicago	IL	43555
8	Ian	Sampson	56 Manitoba St.	Winnipeg	MB	M5W 9N7
9	Ed	Rodgers	15 Athol St.	Regina	SK	S4T 2V9

This data is in 1NF: all fields are atomic and the CustID serves as the primary key

- Pay attention to the City, State, and Zip fields:
  - There are 2 rows of **repeating data**: one for Chicago, and one for St. Paul.
  - Both have the same city, state and zip code

City	State	Zip
Tucson	AZ	12345
St. Paul	MN	54355
Chicago	IL	43555
Vancouver	BC	V5N 1M0
St. Paul	MN	54355
Regina	SK	S4T 2V8
Chicago	IL	43555
Winnipeg	MB	M5W 9N7
Regina	SK	S4T 2V9

- The CustID determines all the data in the row, but **Zip** codes determine the **City** and **State**.

  (eg. A given Zip code can only belong to one city and state so storing Zip codes with a City and State is redundant)
- This means that **City** and **State** are ***Functionally Dependent*** on the value in **Zip** code and not only the primary key.

- To be in 2NF, this repeating data must be in its own table.
- So:
  - create a Zip code table that maps Zip codes to their City and State.

Data in 2NF

Customer Table

CustID	FirstName	LastName	Address	Zip
1	Bob	Smith	123 Main St.	12345
2	John	Brown	555 2nd Ave.	54355
3	Sandy	Jessop	4256 James St.	43555
4	Maria	Hernandez	4599 Columbia	V5N 1M0
5	Gameil	Hintz	569 Summit St.	54355
6	James	Richardson	12 Cameron Bay	S4T 2V8
7	Shiela	Green	12 Michigan Ave.	43555
8	Ian	Sampson	56 Manitoba St.	M5W 9N7
9	Ed	Rodgers	15 Athol St.	S4T 2V9

Zip Code Table

Zip	City	State
12345	Tucson	AZ
54355	St. Paul	MN
43555	Chicago	IL
V5N 1M0	Vancouver	BC
S4T 2V8	Regina	SK
M5W 9N7	Winnipeg	MB
S4T 2V9	Regina	SK

- *We see that we can actually store 2 rows in the Zip Code table by removing these redundancies: 9 customer records only need 7 Zip code records.*
- *Zip code becomes a foreign key in the customer table linked to the primary key in the Zip code table*

Advantages of 2NF

- Saves space in the database by reducing redundancies.
- Example:
  - If a customer calls, you can just ask them for their Zip code and you'll know their city and state.
  - If a City name changes, we only need to make one change to the database.

Summary So Far…

- 1NF:
  - All data is atomic.
  - All rows have a unique primary key.
- 2NF:
  - Data is in 1NF
  - Non key attributes are fully functionally dependent on key attributes.
  - Subsets of data in multiple columns are moved to a new table.
  - These new tables are related using foreign keys.

3NF

- To be in 3NF, a database must be:
  - In 2NF
  - All columns must be fully functionally dependent on the primary key (no transitive dependencies).

- A **transitive dependency** is a type of functional dependency in which the value in a non-key field is determined by the value in another non-key field and that field is not a candidate key.

OrderID	CustID	ProdID	Price	Quantity	Total
1	1001	AB-111	50	1,000	50,000
2	1002	AB-111	60	500	30,000
3	1001	ZA-245	35	100	3,500
4	1003	MB-153	82	25	2,050
5	1004	ZA-245	42	10	420
6	1002	ZA-245	40	50	2,000
7	1001	AB-111	75	100	7,500

- In this table:
  - CustomerID and ProdID depend on the OrderID and no other column.
  - Stated another way, "If you know the OrderID, you know the CustID and the ProdID".
- So: OrderID → CustID, ProdID

OrderID	CustID	ProdID	Price	Quantity	Total
1	1001	AB-111	50	1,000	50,000
2	1002	AB-111	60	500	30,000
3	1001	ZA-245	35	100	3,500
4	1003	MB-153	82	25	2,050
5	1004	ZA-245	42	10	420
6	1002	ZA-245	40	50	2,000
7	1001	AB-111	75	100	7,500

- But there are some fields that are not dependent on OrderID:
  - Total is the simple product of Price*Quantity. As such, has a transitive dependency to Price and Quantity.
  - Because it is a calculated value, doesn't need to be included at all.

OrderID	CustID	ProdID	Price	Quantity	Total
1	1001	AB-111	50	1,000	50,000
2	1002	AB-111	60	500	30,000
3	1001	ZA-245	35	100	3,500
4	1003	MB-153	82	25	2,050
5	1004	ZA-245	42	10	420
6	1002	ZA-245	40	50	2,000
7	1001	AB-111	75	100	7,500

- Also, we can see that Price isn't really dependent on ProdID, or OrderID.

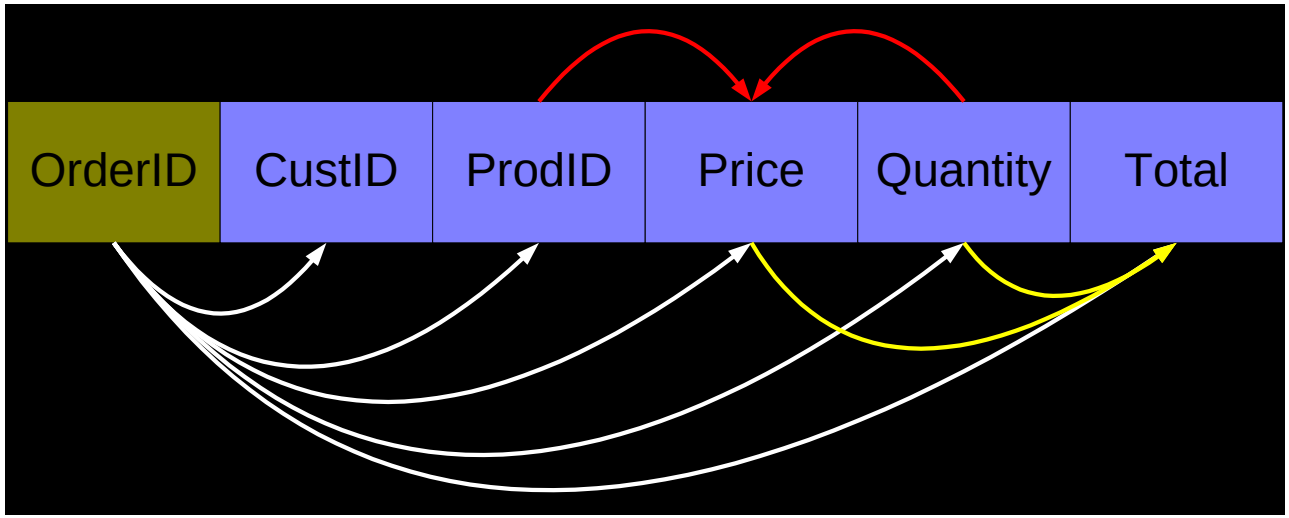
- Example: Customer 1001 bought AB-111 for $50 (in order 1) and for $75 (in order 7), while 1002 spent $60 for each item in order 2.

OrderID	CustID	ProdID	Price	Quantity	Total
1	1001	AB-111	50	1,000	50,000
2	1002	AB-111	60	500	30,000
3	1001	ZA-245	35	100	3,500
4	1003	MB-153	82	25	2,050
5	1004	ZA-245	42	10	420
6	1002	ZA-245	40	50	2,000
7	1001	AB-111	75	100	7,500

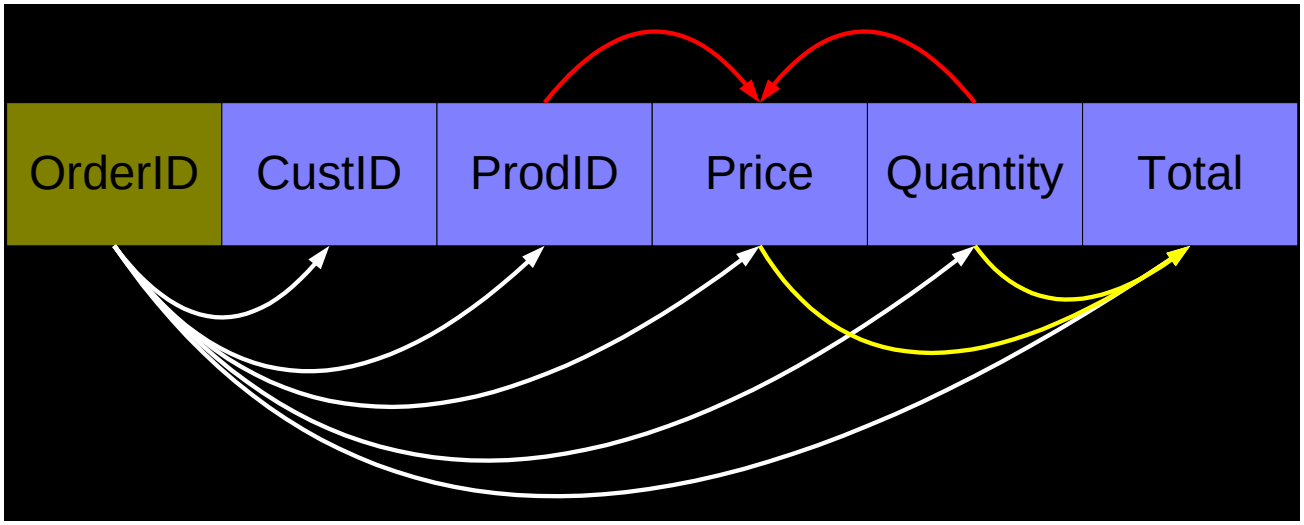
- Maybe price is dependent on the ProdID and Quantity:
  - The more you buy of a given product the cheaper that product becomes.

OrderID	CustID	ProdID	Price	Quantity	Total
1	1001	AB-111	50	1,000	50,000
2	1002	AB-111	60	500	30,000
3	1001	ZA-245	35	100	3,500
4	1003	MB-153	82	25	2,050
5	1004	ZA-245	42	10	420
6	1002	ZA-245	40	50	2,000
7	1001	AB-111	75	100	7,500

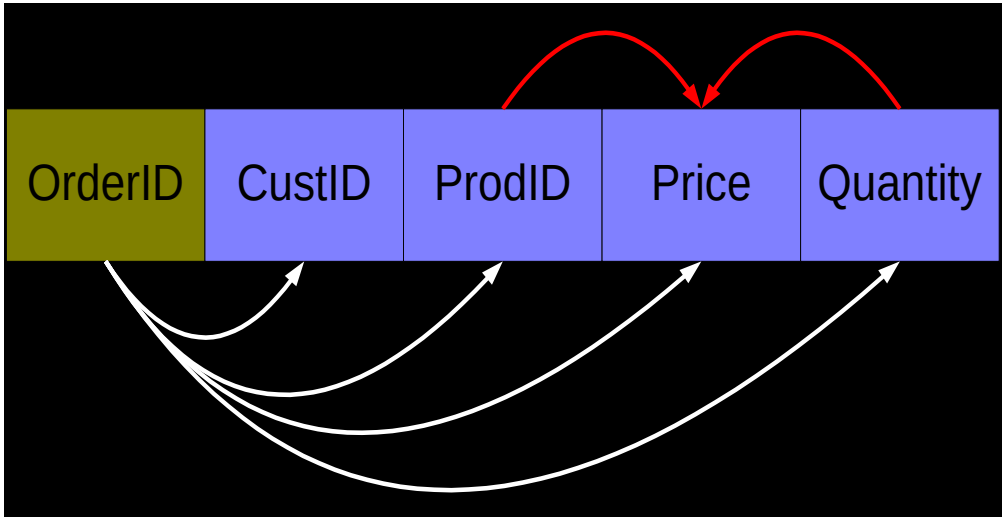
- We say that Price has a **transitive dependency** on ProdID and Quantity.
  - This means that Price isn't just determined by the OrderID. It is also determined by the quantity of the order and what is ordered (ProdID).

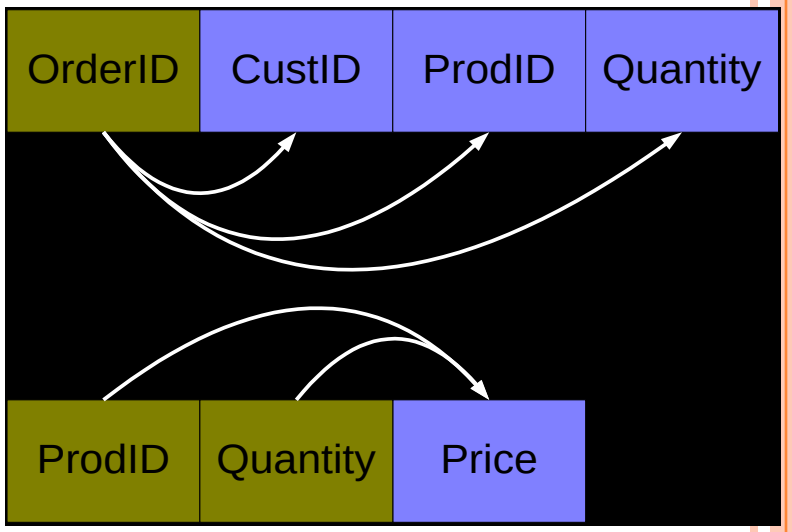


- Let's diagram the dependencies.
- We can see that all fields are dependent on OrderID, the Primary Key (white lines)



- But Total is also determined by Price and Quantity (yellow lines)
  - This is a derived field/attribute (Price x Quantity = Total)
  - We can save a lot of space by getting rid of it altogether and just calculating total when we need it.





- Price is also determined by both ProdID and Quantity rather than the primary key (red lines). This is called a **transitive dependency**.
- We must get rid of transitive dependencies to have 3NF.
- We do this by moving the transitive dependency into a second table…

OrderID	CustID	ProdID	Quantity
1	1001	AB-111	1,000
2	1002	AB-111	500
3	1001	ZA-245	100
4	1003	MB-153	25
5	1004	ZA-245	10
6	1002	ZA-245	50
7	1001	AB-111	100

ProdID	Quantity	Price
AB-111	1	75
AB-111	101	60
AB-111	501	50
ZA-245	1	42
ZA-245	11	40
ZA-245	51	35
MB-153	1	82

- The second table is our pricing list.
- Think of Quantity as a range:
  - AB-111: 1-100, 101-500, 501 and more
    ZA-245: 1-10, 11-50, 51 and more
- The primary Key for this second table is a composite of ProdID and Quantity.

OrderID	CustID	ProdID	Quantity
1	1001	AB-111	1,000
2	1002	AB-111	500
3	1001	ZA-245	100
4	1003	MB-153	25
5	1004	ZA-245	10
6	1002	ZA-245	50
7	1001	AB-111	100

ProdID	Quantity	Price
AB-111	1	75
AB-111	101	60
AB-111	501	50
ZA-245	1	42
ZA-245	11	40
ZA-245	51	35
MB-153	1	82

- We're now in 3NF!
- We can also quickly figure out what price to offer our customers for any quantity they want.

To Summarize (again)

- A database is in 3NF if:
  - It is in 2NF
  - It has no transitive dependencies
    - *A transitive dependency exists when one attribute (or field) is determined by another non-key attribute (or field)*
    - *We remove fields with a transitive dependency to a new table and link them by a foreign key.*