SQL

Relational model

Introduction to NoSQL: Lecture 2

Piotr Fulmański

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
University of Lodz

NoSQL Theory and examples

by Piotr Fulmański

Piotr Fulmański, 2021

PIOTR FULMAŃSKI

NoSQL
Theory and examples

SIMPLE INTRODUCTION SERIES

- Relational theory key concepts
- Normal forms
- Transactional model and ACID

Why we are talking about SQL?

- If there are situations where relational databases aren’t the best match for our business problem, we should know why this pattern is not suitable for us and what solution(s) we can choose instead.

- Relational model is well known and has very good support in terms of software, documentation, specialist – before you will resign from this patter, you should be sure what you are doing.

- To correctly understand NoSQL databases and all pros and cons we need some reference – in this case we will compare them with one mode we know best – the relational model.

Toward relational supremacy

Pre-relational era

- US census was a first large-scale example of storing some data in a form suitable for further automatic processing.
- The appearance of the term database coincided with the availability of versatile, high-speed access storages like tapes and next drums and disks which made work with individual records possible and effective.
- It became quite natural to do all database related stuff once and use many times for different databases. This way database handling logic was moved from the applications to the intermediate layer – the Database Management System, or DBMS.
- Early database management systems, from performance reasons and lack of any other preceding experiences and premises, enforced both schema and path to get an access to data.

Toward relational supremacy

Pre-relational era

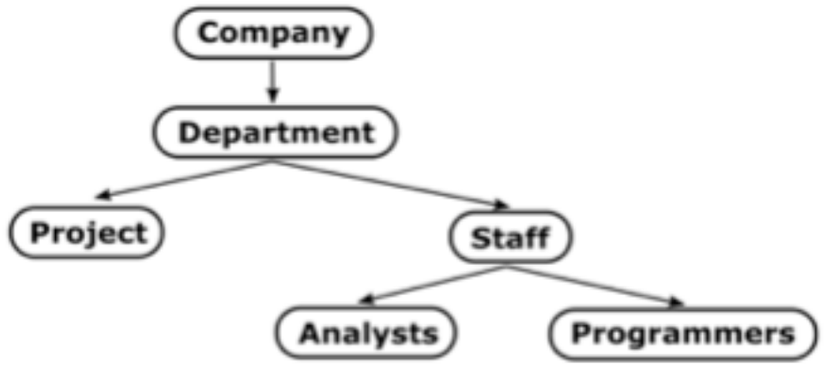
- A **schema** defines the physical structure of the data within the database. The system to deliver adequate performance enforced data representation so it was fine-tuned to the underlying access mechanisms.

- An **access path** (a way we get the data) defines a fixed sequence of navigating steps from one record to another – there was no option to have a free access to any record we want as it is in today's systems.

Toward relational supremacy

Pre-relational era

- Pre-relational databases were designed by manufacturers, not users.



Hierarchical model

Network model

Toward relational supremacy

Relational era

- Existing databases **mixed logical and physical implementations.** The representation of data in existing databases matched the format of the physical storage in the database, rather than a logical representation of the data that could be comprehended by a nontechnical user.

- There were **no common formal structures and operations** we could do on different databases.

- Existing databases **lacked a theoretical foundations guarantee their reliability**. Based on arbitrary representations that did not ensure logical consistency it wasn't possible to deal with failures and consistencies problems.

- In consequence, existing databases were **too hard to use**. Databases of the day could only be accessed by people with specialized technical and programming skills which was very distant from business needs and applications.

Toward relational supremacy

Relational era

- While all of them differentiate in terms of performance, availability, functionality, or economy, they all share three key principles:
- Codd’s relational model,
- the SQL language,
- and the ACID transaction model.

Toward relational supremacy

Object-oriented era

Relational theory key concepts

Row oriented model

The relational model has a very well-ordered and organized nature. It is a row oriented model and row is its fundamental concept.

- **Row** is a basic storage unit used to keep data.
- Set of rows is called a **table**.
- One single data field in a row is called **column**.
- All rows within one table consist of a fixed number of column.
- Every data field is associated with a column name and a data type.
- Columns must have unique names within a table and a single data type which is created when a table is first defined.
- The entire table, with all column definitions and their data types, must be created before the first row is inserted into the table.
- Rows are always added, changed and deleted as atomic units.
- **Relation between data stored in different tables are only through data itself and no any other "external" properties**.
- We get data in usable form by selecting all related rows from different tables with **JOIN** statements.
- Data are stored as tables, however the physical storage of the data is independent of the way the data are logically organized.

Relational theory key concepts

Row oriented model

All the constraints helps to make this model "predictable" in a sense that can be described in a strict and formalized way. Formalization is important because allow us to get rid of all physical dependencies among data (as it is in the hierarchical and network model case) and focus only on its (data) logical structure.

In other words, **having formal description we can discuss database properties with no knowledge about its physical organization**.

Relational theory key concepts

Row oriented model

- The **source of the power** of relational model is a **separation of logical and physical structure of data**.
- In other words, the way we navigate through our data depends **only** on data itself and **nothing** else.
- This model is totally **abstract** – it may no assumption about real object (represented as data) dependency.
- Whether we will make only on-table select or very complex multi-join queries data organization into tables will be exactly the same.

Normal forms

invoice number	customer number	customer name	customer location	item (name, quantity, price)
1	10	Dart Vader	Star Destroyer	{lightsaber, 1, 100}, {black cloak, 2, 50}, {air filter, 10, 2}
2	30	C3PO	Tatooine	{battery, 1, 25}
3	20	Luke Skywalker	Naboo	{lightsaber, 5, 75}, {belt, 1, 5}
4	30	C3PO	Tatooine	{wires, 1, 10}

Normal forms

First normal form (1NF)

A relation is in first normal form if and only if the domain of each attribute contains only atomic (indivisible) values, and the value of each attribute contains only a single value from that domain. Moreover, for every row there should exist primary key – one or more columns uniquely identifying that row.

In practice it means that we have to:

- Eliminate repeating groups in individual tables.
- Create a separate table for each set of related data.
- Identify each set of related data with a primary key.

Normal forms

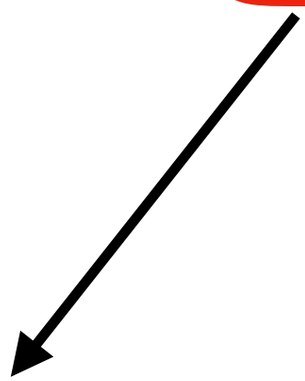
First normal form (1NF)

invoice number	customer number	customer name	customer location	item (name, quantity, price)
1	10	Dart Vader	Star Destroyer	{lightsaber, 1, 100}, {black cloak, 2, 50}, {air filter, 10, 2}
2	30	C3PO	Tatooine	{battery, 1, 25}
3	20	Luke Skywalker	Naboo	{lightsaber, 5, 75}, {belt, 1, 5}
4	30	C3PO	Tatooine	{wires, 1, 10}

Normal forms

First normal form (1NF)

invoice number	customer number	customer name	customer location	item (name, quantity, price)
1	10	Dart Vader	Star Destroyer	{lightsaber, 1, 100}, {black cloak, 2, 50}, {air filter, 10, 2}
2	30	C3PO	Tatooine	{battery, 1, 25}
3	20	Luke Skywalker	Naboo	{lightsaber, 5, 75}, {belt, 1, 5}
4	30	C3PO	Tatooine	{wires, 1, 10}



invoice number	invoice item	customer number	customer name	customer location	item name	item quantity	item price
1	1	10	Dart Vader	Star Destroyer	lightsaber	1	100
1	2	10	Dart Vader	Star Destroyer	black cloak	2	50
1	3	10	Dart Vader	Star Destroyer	air filter	10	2
2	1	30	C3PO	Tatooine	battery	1	25
3	1	20	Luke Skywalker	Naboo	lightsaber	5	75
3	2	20	Luke Skywalker	Naboo	belt	1	5
4	1	30	C3PO	Tatooine	wires	1	10

Normal forms

Second normal form (2NF)

Second normal form (2NF) addresses the concept of removing duplicative data.

No non-prime attributes have part-key dependencies on any of candidate keys.
Other words, any functional dependency on part of any candidate key is a violation of 2NF.

In practice it means that we have to:

- Remove subsets of data that apply to multiple rows of a table and place them in separate tables.

Normal forms

Second normal form (2NF)

invoice number	invoice item	customer number	customer name	customer location	item name	item quantity	item price
1	1	10	Dart Vader	Star Destroyer	lightsaber	1	100
1	2	10	Dart Vader	Star Destroyer	black cloak	2	50
1	3	10	Dart Vader	Star Destroyer	air filter	10	2
2	1	30	C3PO	Tatooine	battery	1	25
3	1	20	Luke Skywalker	Naboo	lightsaber	5	75
3	2	20	Luke Skywalker	Naboo	belt	1	5
4	1	30	C3PO	Tatooine	wires	1	10

Normal forms

Second normal form (2NF)

invoice number	invoice item	customer number	customer name	customer location	item name	item quantity	item price
1	1	10	Dart Vader	Star Destroyer	lightsaber	1	100
1	2	10	Dart Vader	Star Destroyer	black cloak	2	50
1	3	10	Dart Vader	Star Destroyer	air filter	10	2
2	1	30	C3PO	Tatooine	battery	1	25
3	1	20	Luke Skywalker	Naboo	lightsaber	5	75
3	2	20	Luke Skywalker	Naboo	belt	1	5
4	1	30	C3PO	Tatooine	wires	1	10

invoice details table

invoice number	invoice item	item name	item quantity	item price
1	1	lightsaber	1	100
1	2	black cloak	2	50
1	3	air filter	10	2
2	1	battery	1	25
3	1	lightsaber	5	75
3	2	belt	1	5
4	1	wires	1	10

invoice table

invoice number	customer number	customer name	customer location
1	10	Dart Vader	Star Destroyer
2	30	C3PO	Tatooine
3	20	Luke Skywalker	Naboo
4	30	C3PO	Tatooine

Normal forms

Third normal form (3NF)

Third normal form (3NF) removes columns that are not dependent upon the primary key which is one step further in the concept of removing duplicative data.

In practice it means that we have to:

- Remove columns that are not dependent upon the primary key. Each column must depend directly on the primary key and nothing else than primary key.

Normal forms

Third normal form (3NF)

invoice details table

invoice number	invoice item	item name	item quantity	item price
1	1	lightsaber	1	100
1	2	black cloak	2	50
1	3	air filter	10	2
2	1	battery	1	25
3	1	lightsaber	5	75
3	2	belt	1	5
4	1	wires	1	10

invoice table

invoice number	customer number	customer name	customer location
1	10	Dart Vader	Star Destroyer
2	30	C3PO	Tatooine
3	20	Luke Skywalker	Naboo
4	30	C3PO	Tatooine

Normal forms

Third normal form (3NF)

invoice details table

invoice number	invoice item	item name	item quantity	item price
1	1	lightsaber	1	100
1	2	black cloak	2	50
1	3	air filter	10	2
2	1	battery	1	25
3	1	lightsaber	5	75
3	2	belt	1	5
4	1	wires	1	10

invoice table

invoice number	customer number
1	10
2	30
3	20
4	30

invoice table

invoice number	customer number	customer name	customer location
1	10	Dart Vader	Star Destroyer
2	30	C3PO	Tatooine
3	20	Luke Skywalker	Naboo
4	30	C3PO	Tatooine

customer details table

customer number	customer name	customer location
10	Dart Vader	Star Destroyer
20	Luke Skywalker	Naboo
30	C3PO	Tatooine

Primary key

Normal forms

For what?

Now we have data in the third normal form and we may ask

For what? What profits do I have making this? Is it worth it?

Normal forms help keeping data consistent.

We can say, that **normal forms just ensures that we don’t make rookie mistakes when designing the architecture of a database**.

Side **negative effect of normal forms is high data "fragmentation"** which we have mentioned before (recall analogy to a garage and parking a car in it). This force us to massively use joins – additional join tables as well as JOIN statements to combine pieces of informations into one real object.

Normal forms

Final word

1. You want to have your data to be consistent.
2. Because of 1, you have to limit redundancy.
3. To achieve 2, you need data organization supporting it.
4. The way to realize 3 is follow normal forms rules.

Transaction

Relational model itself, even with all data organized according to normal form rules, doesn’t guarantee data consistency, and system reliability. This is achieved by implementing **transaction model**.

Transaction

ACID

ACID transaction model ensures that:

- Data are always as we expect them to be.
- If we change value now, it will be changed also in the future – if there are no other actions, we will never see the value before change again.
- If two attempts to data change occur at the same time, the result is the same as for their sequential execution.
- If one change depends on the other, then both will be performed in a specific, desired order. If one fails, the effect of any of them will not be visible.
- After finalizing current changes other new changes cannot appear till next action is taken.

Transaction

ACID

- ***A****tomic* The transaction can not be divided – either all the statements in the transaction are applied to the database or none are.

- ***C****onsistent* It refers to the correctness of a database. The database remains in a consistent state before and after transaction execution. All stored data are correct and in accordance with their logic order of processing.

- ***I****solated* While multiple transactions can be executed by one or more users simultaneously, one transaction should not see the effects of other in-progress transactions. This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.

- ***D****urable* Once a transaction is saved (committed) to the database, its changes are expected to persist even if there is a failure of operating system or hardware. The effects of the transaction, thus, are never lost.

Transaction

ACID

Note that:

- ACID features are not independent – lack of one of them may affect others. For example, lack of isolation may affect consistency.

- Paradoxically by trying to increase the overall system performance by adding more machines we achieve the opposite effect: the more machines constitutes our system, the more time we may lost trying to ensure ACID properties.

Bibliography

- [Ful] Piotr Fulmański, *NoSQL. Theory and examples*, Piotr Fulmański, 2021