Agenda

- What is Schema Design?
- Database models
- Normalization
- ACID properties

What is Schema Design ?

A database schema is a blueprint that addresses every functional element of a database, such as tables, fields, records and keys and their relationshi to each other. Schemas can be designed either for minimal redundancy and maximum interpretability, or for performance and reducing the complexit time and cost of the queries needed to pull data from a database. Good database schema design can be the difference between a query taking secon and the same query lasting many hours.

Why design database schemas?

A well-designed schema helps ensure data integrity. You may design a database schema when designing a database from scratch, importing data from different source into your own application or a data warehouse or reverse-engineering an existing database for better performance. In all of thes scenarios, database schemas can ensure consistent formatting and the maintenance of unique primary and foreign keys.

When a schema is well-designed for analytics:

- Analysts don’t have to clean data or preprocess it before analyzing it.
- Analysts don’t have to reverse-engineer the underlying data model.
- Analysts have a clear, easily understood starting point for analytics.

A schema designed for analytics means faster access to data for creating reports and dashboards. The opposite means extra data modeling, the slo retrieval of data and the consumption of more resources (such as time and processing power) when creating reports for analysis.

In the data analytics world, both data sources and data warehouses use schemas to define data elements. But the schemas for data sources — wheth they’re databases such as MySQL, PostgreSQL or SQL Server, or SaaS applications such as Salesforce, Facebook Ads, or Zuora — are often designe with operations rather than analytics in mind.

Types of database models

There are many kinds of data models. Some of the most common ones include:

- Hierarchical database model
- Relational model
- Network model
- Object-oriented database model
- Entity-relationship model
- Document model
- Entity-attribute-value model
- Star schema
- The object-relational model, which combines the two that make up its name

You may choose to describe a database with any one of these depending on several factors. The biggest factor is whether the database manageme system you are using supports a particular model. Most database management systems are built with a particular data model in mind and require the users to adopt that model, although some do support multiple models.

In addition, different models apply to different stages of the database design process. High-level conceptual data models are best for mapping o relationships between data in ways that people perceive that data. Record-based logical models, on the other hand, more closely reflect ways that th data is stored on the server.

Selecting a data model is also a matter of aligning your priorities for the database with the strengths of a particular model, whether those priorities includ speed, cost reduction, usability, or something else.

Let’s take a closer look at some of the most common database models.

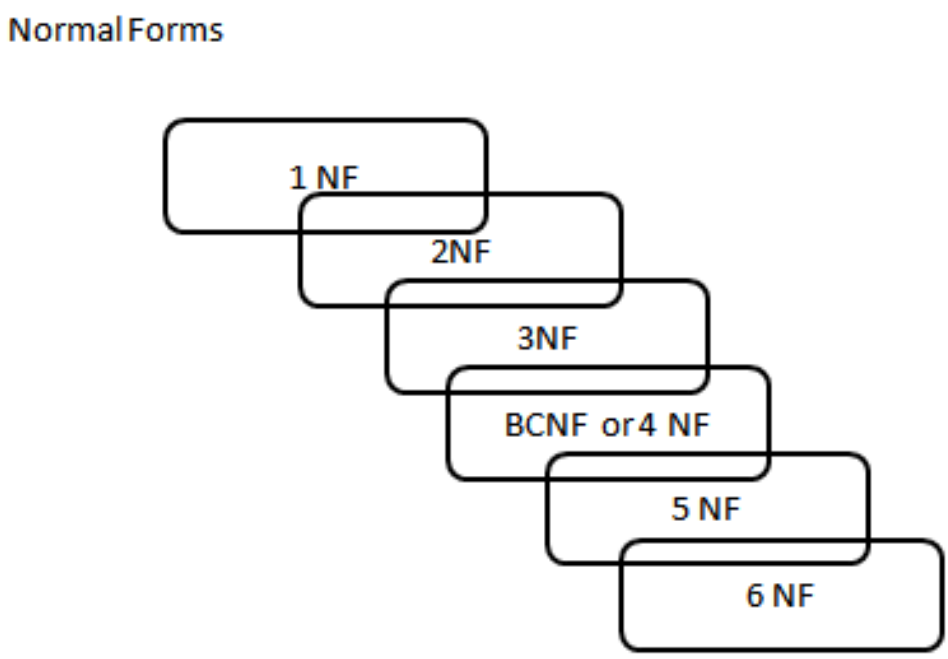
What is Database Normalization

Database normalization is the process of restructuring a relational database in accordance with a series of so-called normal forms in order to reduce da redundancy and improve data integrity.

Normalization entails organizing the columns (attributes) and tables (relations) of a database to ensure that their dependencies are properly enforced b database integrity constraints. It is accomplished by applying some formal rules either by a process of synthesis (creating a new database design) decomposition (improving an existing database design).The process of taking a database design, and apply a set of formal criteria and rules, is calle Normal Forms.

The database normalization process is further categorized into the following types:

1. First Normal Form (1 NF)
2. Second Normal Form (2 NF)
3. Third Normal Form (3 NF)
4. Boyce Codd Normal Form or Fourth Normal Form (BCNF or 4 NF)
5. Fifth Normal Form (5 NF)
6. Sixth Normal Form (6 NF)



One of the driving forces behind database normalization is to streamline data by reducing redundant data. Redundancy of data means there are multip copies of the same information spread over multiple locations in the same database.

The drawbacks of data redundancy include:

1. Data maintenance becomes tedious – data deletion and data updates become problematic
2. It creates data inconsistencies
3. Insert, Update and Delete anomalies become frequent. An update anomaly, for example, means that the versions of the same record, duplicated in different places in the database, will all need to be updated to keep the record consistent
4. Redundant data inflates the size of a database and takes up an inordinate amount of space on disk

Normal Forms

This article is an effort to provide fundamental details of database normalization. The concept of normalization is a vast subject and the scope of th article is to provide enough information to be able to understand the first three forms of database normalization.

1. First Normal Form (1 NF)
2. Second Normal Form (2 NF)
3. Third Normal Form (3 NF)

A database is considered third normal form if it meets the requirements of the first 3 normal forms.

First Normal Form (1NF):

The first normal form requires that a table satisfies the following conditions:

1. Rows are not ordered
2. Columns are not ordered
3. There is duplicated data
4. Row-and-column intersections always have a unique value
5. All columns are “regular” with no hidden values

In the following example, the first table clearly violates the 1 NF. It contains more than one value for the Dept column. So, what we might do then is g back to the original way and instead start adding new columns, so, Dept1, Dept2, and so on. This is what’s called a repeating group, and there should b no repeating groups. In order to bring this First Normal Form, split the table into the two tables. Let’s take the department data out of the table and put in the dept table. This has the one-to-many relationship to the employee table.

Let’s take a look at the employee table:

EmpID	Employee	Age	Dept
1001	ABC	30	Sales,Finance
1002	CDE	30	Sales,Finance,DevOps

Now, after normalization, the normalized tables Dept and Employee looks like below:

DeptID	DeptName
1	Sales
2	Finance
3	DevOps

EmpId	Employee	Age	DepID
1001	ABC	30	1
1001	ABC	30	2
1002	CDE	40	1
1002	CDE	40	2
1002	CDE	40	3

Second Normal Form and Third Normal Form are all about the relationship between the columns that are the keys and the other columns that aren’t th key columns.

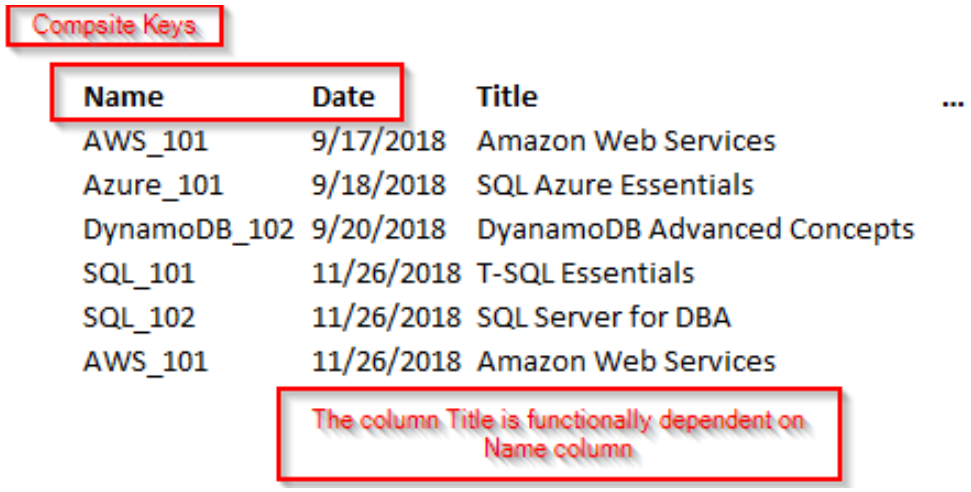
Second Normal Form (2NF):

An entity is in a second normal form if all of its attributes depend on the whole primary key. So this means that the values in the different columns have dependency on the other columns.

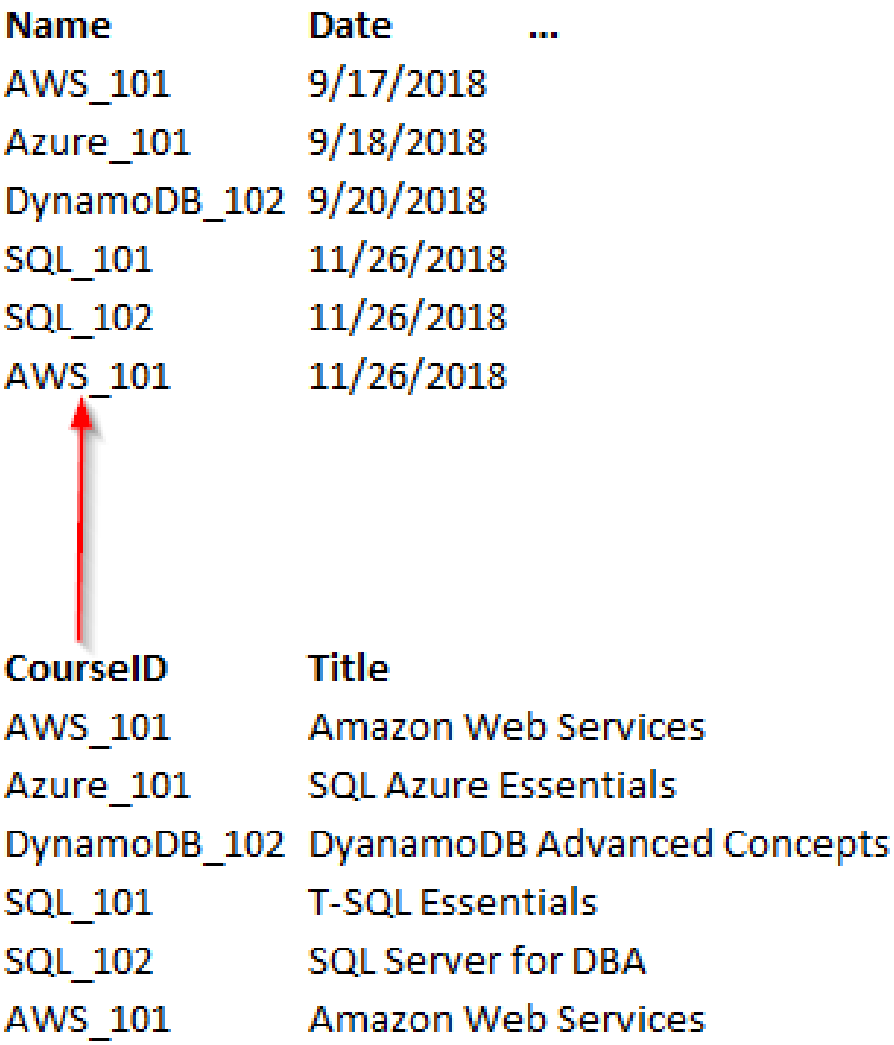
1. The table must be already in 1 NF and all non-key columns of the tables must depend on the PRIMARY KEY
2. The partial dependencies are removed and placed in a separate table

Note: Second Normal Form (2 NF) is only ever a problem when we're using a composite primary key. That is, a primary key made of two or mo columns.

The following example, the relationship is established between the Employee and Department tables.



In this example, the Title column is functionally dependent on Name and Date columns. These two keys form a composite key. In this case, it on depends on Name and partially dependent on the Date column. Let's remove the course details and form a separate table. Now, the course details a based on the entire key. We are not going to use a composite key.

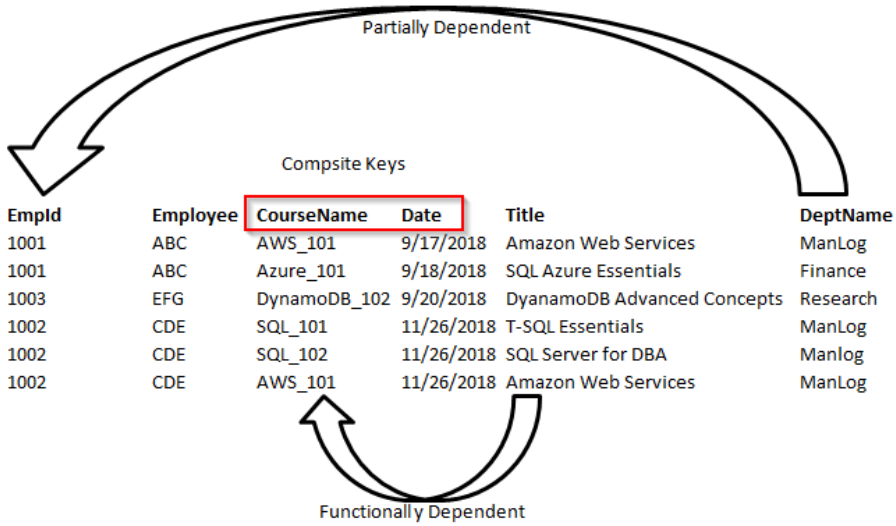


Third Normal Form (3NF):

The third normal form states that you should eliminate fields in a table that do not depend on the key.

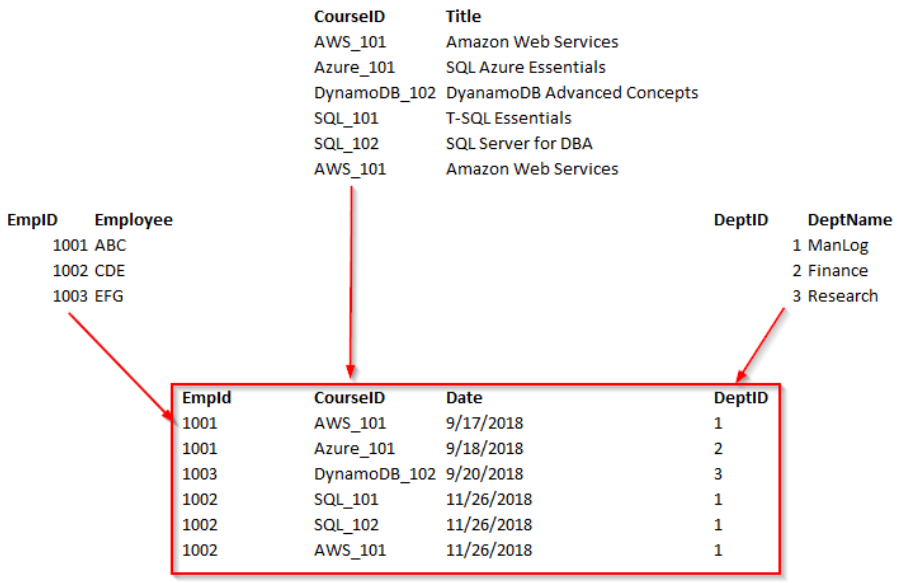
1. A Table is already in 2 NF
2. Non-Primary key columns shouldn't depend on the other non-Primary key columns
3. There is no transitive functional dependency

Consider the following example, in the table employee; empID determines the department ID of an employee, department ID determines the departme name. Therefore, the department name column indirectly dependent on the empID column. So, it satisfies the transitive dependency. So this cannot be third normal form.



EmpId	Employee	CourseName	Date	Title	DeptName
1001	ABC	AWS_101	9/17/2018	Amazon Web Services	ManLog
1001	ABC	Azure_101	9/18/2018	SQL Azure Essentials	Finance
1003	EFG	DynamoDB_102	9/20/2018	DyanamoDB Advanced Concepts	Research
1002	CDE	SQL_101	11/26/2018	T-SQL Essentials	ManLog
1002	CDE	SQL_102	11/26/2018	SQL Server for DBA	Manlog
1002	CDE	AWS_101	11/26/2018	Amazon Web Services	ManLog

In order to bring the table to 3 NF, we split the employee table into two.



CourseID	Title
AWS_101	Amazon Web Services
Azure_101	SQL Azure Essentials
DynamoDB_102	DyanamoDB Advanced Concepts
SQL_101	T-SQL Essentials
SQL_102	SQL Server for DBA
AWS_101	Amazon Web Services

EmpID	Employee
1001	ABC
1002	CDE
1003	EFG

DeptID	DeptName
1	ManLog
2	Finance
3	Research

EmpId	CourseID	Date	DeptID
1001	AWS_101	9/17/2018	1
1001	Azure_101	9/18/2018	2
1003	DynamoDB_102	9/20/2018	3
1002	SQL_101	11/26/2018	1
1002	SQL_102	11/26/2018	1
1002	AWS_101	11/26/2018	1

Now, we can see the all non-key columns are fully functionally dependent on the Primary key.

Although a fourth and fifth form does exist, most databases do not aspire to use those levels because they take extra work and they don't truly impact th database functionality and improve performance.

ACID Properties

The transaction properties are referred to as ACID (Atomicity, Consistency, Isolation, Durability) property, which are discussing in detail below:

Atomicity: This property ensures that all statements or operations included in the transaction must be performed successfully. Otherwise, the who transaction will be aborted, and all operations are rolled back into their previous state when any operation is failed.

Consistency: This property ensures that the database changes state only when a transaction will be committed successfully. It is also responsible f protecting data from crashes.

Isolation: This property guarantees that all transactions are isolated from other transactions, meaning each operation in the transaction is operate independently. It also ensures that statements are transparent to each other.

Durability: This property guarantees that the result of committed transactions persists in the database permanently even if the system crashes or failed

---

Thank You !

Interview Questions

- **List the different types of relationships in SQL.**

- **One-to-One** - This can be defined as the relationship between two tables where each record in one table is associated with the maximum of one record in the other table.
- **One-to-Many & Many-to-One** - This is the most commonly used relationship where a record in a table is associated with multiple records in the other table.
- **Many-to-Many** - This is used in cases when multiple instances on both sides are needed for defining a relationship.

- **What is Normalization?** Normalization represents the way of organizing structured data in the database efficiently. It includes the creation of tables, establishing relationships between them, and defining rules for those relationships. Inconsistency and redundancy can be kept in check based on these rules, hence, adding flexibility to the database.

- **What are the disadvantages of not performing database Normalization?**

The major disadvantages are: The occurrence of redundant terms in the database causes the waste of space in the disk. Due to redundant terms, inconsistency may also occur. If any change is made in the data of one table but not made in the same data of another table, then inconsistency will occur. This inconsistency will lead to the maintenance problem and effects the ACID properties as well.