# Top 23 Database Schema

valid\_valuesdomainentrydescriptionvarchar(255)varchar(1000)varchar(1000)PKPK(Antipattern!) This table representsthe values that are valid in all of thedomains database, not a single domain.

usersuser\_accountidhashed\_passwordfirst\_namelast\_nameemailcreatedgroup\_iduser\_role\_idintvarchar(100)varchar(100)varchar(100)varchar(254)timestampintintPKFKFKuser\_roleidnamedescriptionintvarchar(100)varchar(1000)PKrole\_permissionuser\_role\_idpermission\_idintintPK FKPK FKpermissionidnameintvarchar(100)PKgroupidnamedescriptionintvarchar(100)varchar(1000)PK

1:N v M:Nentity v entity instanceattribute v entitycomplex attributeclientidnamesurnameintvarchar(255)varchar(255)PKaccountidtypeclient\_idintvarchar(255)intPKFKclientidnamesurnameintvarchar(255)varchar(255)PKaccountidtypeclient\_idintvarchar(255)intPKFKlaptopidmodelintvarchar(255)PKemployeeidnamesurnamelaptop\_idintvarchar(255)varchar(255)intPKFKequipmentidtypemodelintvarchar(255)varchar(255)PKemployeeidnamesurnameequipment\_idintvarchar(255)varchar(255)intPKFKemployeeidnamesurnamedate\_birthid\_departmentname\_departmenthead\_departmentintvarchar(255)varchar(255)dateintvarchar(255)varchar(255)PKemployeeidnamesurnamedate\_birthdepartment\_idintvarchar(255)varchar(255)dateintPKFKdepartmentidnameheadintvarchar(255)varchar(255)PKstudentsidnamesurnamedate\_birthaddressexam\_passedintvarchar(255)varchar(255)datevarchar(255)varchar(255)PKstudentsidnamesurnamedate\_birthaddressintvarchar(255)varchar(255)datevarchar(255)PKexamsiddatestudents\_idscoreintdateintvarchar(255)PKFK

FreelancersCustomers & ProjectsProject phasesTeamsfreelancer:Freelancersiduser\_namepasswordfirst\_namelast\_nameemailmobilecurrent\_availability\_idintvarchar(64)varchar(255)varchar(64)varchar(64)varchar(255)varchar(255)intPKFKskill:Freelancersidskill\_nameintvarchar(64)PKhas\_skillidfreelancer\_idskill\_idskill\_level\_idintintintintPKFKFKFKskill\_levelidlevelintvarchar(32)PKproject:Customers & Projectsidcustomer\_idproject\_namedescriptionbudget\_planbudget\_estimatebudget\_actualamount\_paidproject\_outcome\_idintintvarchar(255)textdecimal(12,2)decimal(12,2)decimal(12,2)decimal(12,2)intPKFKFKcustomer:Customers & Projectsiduser\_namepasswordcustomer\_nameemailmobiledetailsintvarchar(64)varchar(64)varchar(255)varchar(255)varchar(255)textPKskill:Customers & Projectsidskill\_nameintvarchar(64)PKskill\_requiredidproject\_idskill\_idintintintPKFKFKproject:Project phasesidcustomer\_idproject\_namedescriptionbudget\_planbudget\_estimatebudget\_actualamount\_paidproject\_outcome\_idintintvarchar(255)textdecimal(12,2)decimal(12,2)decimal(12,2)decimal(12,2)intPKFKFKphase\_catalogidphase\_catalog\_nameproject\_outcome\_idintvarchar(255)intPKN FKphase\_historyidfreelancer\_idcustomer\_idproject\_idphase\_catalog\_idstart\_timeend\_timecommentintintintintinttimestamptimestamptextPKN FKN FKFKFKNcustomer:Project phasesiduser\_namepasswordcustomer\_nameemailmobiledetailsintvarchar(64)varchar(64)varchar(255)varchar(255)varchar(255)textPKfreelancer:Project phasesiduser\_namepasswordfirst\_namelast\_nameemailmobilecurrent\_availability\_idintvarchar(64)varchar(255)varchar(64)varchar(64)varchar(255)varchar(255)intPKFKphase\_plan:Project phasesidproject\_idphase\_catalog\_idstart\_time\_plannedend\_time\_plannedcommentfreelancer\_idintintinttimestamptimestamptextintPKFKFKNFKteamidteam\_namedescriptionintvarchar(128)textPKfreelancer:Teamsiduser\_namepasswordfirst\_namelast\_nameemailmobilecurrent\_availability\_idintvarchar(64)varchar(255)varchar(64)varchar(64)varchar(255)varchar(255)intPKFKteam\_memberidteam\_idfreelancer\_idhours\_workedintintintintPKFKFKproject:Teamsidcustomer\_idproject\_namedescriptionbudget\_planbudget\_estimatebudget\_actualamount\_paidproject\_outcome\_idintintvarchar(255)textdecimal(12,2)decimal(12,2)decimal(12,2)decimal(12,2)intPKFKFKon\_projectidteam\_idproject\_idstart\_dateend\_dateintintintdatedatePKFKFKNphase\_plan:Teamsidproject\_idphase\_catalog\_idstart\_time\_plannedend\_time\_plannedcommentfreelancer\_idintintinttimestamptimestamptextintPKFKFKNFKin\_chargeidteam\_idphase\_plan\_idintintintPKFKFKproject\_outcome:Project phasesidoutcome\_nameongoingon\_holdis\_completed\_successfullyis\_completed\_unsuccessfullyintvarchar(255)boolboolboolboolPKproject\_outcome:Customers & Projectsidoutcome\_nameongoingon\_holdis\_completed\_successfullyis\_completed\_unsuccessfullintvarchar(255)boolboolboolboolPKavailabilityidavailability\_nameintvarchar(64)PKproject\_status\_historyidproject\_idproject\_outcome\_idetailstsintintinttexttimestampPKFKFK

Cab Bookingsvehiclesvehicle\_idvehicle\_registration\_numberIntegerVarchar(255)M PIMdriversdriver\_iddriver\_namedriver\_phoneIntegerVarchar(255)IntegerM PIMMvehicle\_trip\_trackingvehicle\_trip\_tracking\_idvehicle\_trip\_idgps\_timegps\_latitudegps\_longitudeIntegerIntegerTimestampLongFlatLongFlatM PIMMMMvehicle\_tripvehicle\_trip\_idvehicle\_idtrip\_idIntegerIntegerIntegerM PIMMusersuser\_iduser\_nameuser\_phoneIntegerVarchar(255)IntegerM PIMMtripstrip\_iddriver\_idvehicle\_iduser\_idtrip\_statusIntegerIntegerIntegerIntegerVarchar(255)M PIMMMM

suppliersnameaddressVarchar(70)Varchar(140)MMcustomersnameaddressVarchar(70)Varchar(140)MaccountsnumberbalanceIntegerDecimal(11, 2)MMinvoicesamountDecimal(11, 2)MpaymentsamountDecimal(11, 2)M

1) Define Database.

A prearranged collection of figures known as data is called database.

2) What is DBMS?

Database Management Systems (DBMS) are applications designed especially which enable user interaction with other applications.

👉 Free PDF Download: Database (DBMS) Interview Questions & Answers >>

3) What are the various kinds of interactions catered by DBMS?

The various kind of interactions catered by DBMS are:

Data definition

Update

Retrieval

Administration

Database (DBMS) Interview Questions

4) Segregate database technology’s development.

The development of database technology is divided into:

5) Who proposed the relational model?

Edgar F. Codd proposed the relational model in 1970.

6) What are the features of Database language?

A database language may also incorporate features like:

DBMS-specific Configuration and management of storage engine

Computations to modification of query results by computations, like summing, counting, averaging, grouping, sorting and cross-referencing Constraint enforcement Application Programming Interface

7) What do database languages do?

As special-purpose languages, they have:

Data definition language

Data manipulation language

Query language

8) Define database model.

A data model determining fundamentally how data can be stored, manipulated and organised and the structure of the database logically is called database model.

9) What is SQL?

Structured Query Language (SQL) being ANSI standard language updates database and commands for accessing.

SQL

10) Enlist the various relationships of database.

The various relationships of database are:

One-to-one: Single table having drawn relationship with another table having similar kind of columns.

One-to-many: Two tables having primary and foreign key relation.

Many-to-many: Junction table having many tables related to many tables.

11) Define Normalization.

Organized data void of inconsistent dependency and redundancy within a database is called normalization.

12) Enlist the advantages of normalizing database.

Advantages of normalizing database are:

No duplicate entries

Saves storage space

Boasts the query performances.

13) Define Denormalization.

Boosting up database performance, adding of redundant data which in turn helps rid of complex data is called denormalization.

14) Define DDL and DML.

Managing properties and attributes of database is called Data Definition Language(DDL).

Manipulating data in a database such as inserting, updating, deleting is defined as Data Manipulation Language. (DML)

15) Enlist some commands of DDL.

They are:

CREATE:

Create is used in the CREATE TABLE statement. Syntax is:

CREATE TABLE [column name] ( [column definitions] ) [ table parameters]

ALTER:

It helps in modification of an existing object of database. Its syntax is:

ALTER objecttype objectname parameters.

DROP:

It destroys an existing database, index, table or view. Its syntax is:

DROP objecttype objectname.

16) Define Union All operator and Union.

Full recordings of two tables is Union All operator.A distinct recording of two tables is Union.

17) Define cursor.

A database object which helps in manipulating data row by row representing a result set is called cursor.

18) Enlist the cursor types.

They are:

Dynamic: it reflects changes while scrolling.

Static: doesn’t reflect changes while scrolling and works on recording of snapshot.

Keyset: data modification without reflection of new data is seen.

19) Enlist the types of cursor.

They types of cursor are:

Implicit cursor: Declared automatically as soon as the execution of SQL takes place without the awareness of the user.

Explicit cursor: Defined by PL/ SQL which handles query in more than one row.

20) Define sub-query.

A query contained by a query is called Sub-query.

21) Why is group-clause used?

Group-clause uses aggregate values to be derived by collecting similar data.

22) Compare Non-clustered and clustered index

Both having B-tree structure, non-clustered index has data pointers enabling one table many non-clustered indexes while clustered index is distinct for every table.

23) Define Aggregate functions.

Functions which operate against a collection of values and returning single value is called aggregate functions

24) Define Scalar functions.

Scalar function is depended on the argument given and returns sole value.

25) What restrictions can you apply when you are creating views?

Restrictions that are applied are:

Only the current database can have views.

You are not liable to change any computed value in any particular view.

Integrity constants decide the functionality of INSERT and DELETE.

Full-text index definitions cannot be applied.

Temporary views cannot be created.

Temporary tables cannot contain views.

No association with DEFAULT definitions.

Triggers such as INSTEAD OF is associated with views.

26) Define “correlated subqueries”.

A ‘correlated subquery’ is a sort of sub query but correlated subquery is reliant on another query for a value that is returned. In case of execution, the sub query is executed first and then the correlated query.

27) Define Data Warehousing.

Storage and access of data from the central location in order to take some strategic decision is called Data Warehousing. Enterprise management is used for managing the information whose framework is known as Data Warehousing.

28) Define Join and enlist its types.

Joins help in explaining the relation between different tables. They also enable you to select data with relation to data in another table.

The various types are:

INNER JOINs: Blank rows are left in the middle while more than equal to two tables are joined.

OUTER JOINs: Divided into Left Outer Join and Right Outer Join. Blank rows are left at the specified side by joining tables in other side.

Other joins are CROSS JOINs, NATURAL JOINs, EQUI JOIN and NON-EQUI JOIN.

29) What do you mean by Index hunting?

Indexes help in improving the speed as well as the query performance of database. The procedure of boosting the collection of indexes is named as Index hunting.

30) How does Index hunting help in improving query performance?

Index hunting helps in improving the speed as well as the query performance of database. The followed measures are achieved to do that:

The query optimizer is used to coordinate the study of queries with the workload and the best use of queries suggested based on this.

Index, query distribution along with their performance is observed to check the effect.

Tuning databases to a small collection of problem queries is also recommended.

31) Enlist the disadvantages of query.

The disadvantages of query are:

No indexes

Stored procedures are excessively compiled.

Triggers and procedures are without SET NOCOUNT ON.

Complicated joins making up inadequately written query.

Cursors and temporary tables showcase a bad presentation.

32) Enlist ways to efficiently code transactions.

Ways to efficiently code transactions:

User input should not be allowed while transactions.

While browsing, transactions must not be opened of data.

Transactions must be kept as small as possible.

Lower transaction segregation levels.

Least information of data must be accessed while transacting.

33) What is Executive Plan?

Executive plan can be defined as:

SQL Server caches collected procedure or the plan of query execution and used thereafter by subsequent calls.

An important feature in relation to performance enhancement.

Data execution plan can be viewed textually or graphically.

34) Define B-trees.

A data structure in the form of tree which stores sorted data and searches, insertions, sequential access and deletions are allowed in logarithmic time.

35) Differentiate Table Scan from Index Scan.

Iterating over all the table rows is called Table Scan while iterating over all the index items is defined as Index Scan.

36) What do you mean by Fill Factor concept with respect to indexes?

Fill Factor can be defined as being that value which defines the percentage of left space on every leaf-level page that is to be packed with data. 100 is the default value of Fill Factor.

37) Define Fragmentation.

Fragmentation can be defined as a database feature of server that promotes control on data which is stored at table level by the user.

38) Differentiate Nested Loop, Hash Join and Merge Join.

Nested loop (loop over loop)

An outer loop within an inner loop is formed consisting of fewer entries and then for individual entry, inner loop is individually processed.

E.g.

Select col1.\*, col2.\* from coll, col2 where coll.col1=col2.col2;

It’s processing takes place in this way:

For i in (select \* from col1) loop

For j in (select \* from col2 where col2=i.col1) loop

Results are displayed;

End of the loop;

End of the loop;

The Steps of nested loop are:

Identify outer (driving) table

Assign inner (driven) table to outer table.

For every row of outer table, access the rows of inner table.

Nested Loops is executed from the inner to the outer as:

outer\_loop

inner\_loop

Hash join

While joining large tables, the use of Hash Join is preferred.

Algorithm of Hash Join is divided into:

Build: It is a hash table having in-memory which is present on the smaller table.

Probe: this hash value of the hash table is applicable for each second row element.

Sort merge join

Two independent sources of data are joined in sort merge join. They performance is better as compared to nested loop when the data volume is big enough but it is not good as hash joins generally.The full operation can be divided into parts of two:

Sort join operation :

Get first row R1 from input1

Get first row R2 from input2.

Merge join operation:

‘while’ is not present at either loop’s end.

if R1 joins with R2

next row is got R2 from the input 2

return (R1, R2)

else if R1 < style=””> next row is got from R1 from input 1

else

next row is got from R2 from input 2

end of the loop

39) What is Database partitioning?

Division of logical database into independent complete units for improving its management, availability and performance is called Database partitioning.

40) Explain the importance of partitioning.

Splitting of one table which is large into smaller database entities logically is called database partitioning. Its benefits are:

To improve query performance in situations dramatically when mostly rows which are heavily accessed are in one partition.

Accessing large parts of a single partition

Slower and cheaper storage media can be used for data which is seldom used.

41) Define Database system.

DBMS along with database is called Database system.

42) What do you mean by Query Evaluation Engine?

Query Evaluation Engine executes the low-level instructions that are generated by the compiler.

43) Define DDL Interpreter.

DDL statements are interpreted and recorded in tables called metadata.

44) Define Atomicity and Aggregation.

Atomicity: It’s an all or none concept which enables the user to be assured of incomplete transactions to be taken care of. The actions involving incomplete transactions are left undone in DBMS.

Aggregation: The collected entities and their relationship are aggregated in this model. It is mainly used in expressing relationships within relationships.

45) Enlist the various transaction phases.

The various transaction phases are:

Analysis Phase.

Redo Phase

Undo Phase

46) Define Object-oriented model.

Compilations of objects make up this model in which values are stored within instance variables which is inside the object. The object itself comprises bodies of object for its operation which are called methods. Objects containing same kind of variables and methods are called classes.

47) Define Entity.

It can be defined as being a ‘thing’ with an independent existence in the real world.

48) What do you mean by Entity type?

A set of entries having similar attributes are entity types.

49) Define Entity Set.

Compilation of all entries of any particular type of entry in the database is called Entity Set.

50) What do you mean by Entity type extension?

Compilation of similar entity types into one particular type which is grouped together as an entity set.

These interview questions will also help in your viva(orals)

# 31 Database Design Interview Questions (With Sample Answers)

[Indeed Editorial Team](https://www.indeed.com/career-advice/about-us)

Updated June 25, 2022

*The Indeed Editorial Team comprises a diverse and talented team of writers, researchers and subject matter experts equipped with Indeed's data and insights to deliver useful tips to help guide your career journey.*

Data collection and analysis form a vital part of many businesses' operations. Due to this, companies often search for employees with expertise in designing databases and maintaining data management systems. If you're planning to interview for a position in database administration or data management, it can be helpful to learn some of the questions you might encounter. In this article, we explore 31 common database design interview questions and provide some example answers to help you develop your own.

**Related:** [**Database Interview Questions and Answers To Help You Prepare**](https://www.indeed.com/career-advice/interviewing/database-interview-questions)

**Explore jobs on Indeed**

[**Part-time jobs**](https://www.indeed.com/jobs?from=careerguidepromo-US&q=part time)

[**Full-time jobs**](https://www.indeed.com/jobs?from=careerguidepromo-US&q=full time)

[**Remote jobs**](https://www.indeed.com/jobs?from=careerguidepromo-US&q=remote jobs)

[**Urgently hiring jobs**](https://www.indeed.com/jobs?from=careerguidepromo-US&q=urgently hiring)

[**View more jobs on Indeed**](https://www.indeed.com/?from=careerguidepromo-US)

## 25 general database design interview questions

Here are 25 more interview questions about database design that you may encounter:

1. What is the structure of a database?
2. How would you explain an iconic model?
3. What is the CAP theorem and what are its components?
4. Explain the difference between an AP and a CP database.
5. Explain horizontal scaling and describe its benefits and disadvantages.
6. What is vertical scaling? What advantages and disadvantages does it provide?
7. What is a relational database and when would you use one?
8. How would you explain a non-relational database?
9. What are some types of non-relational databases?
10. List some of the types of database keys.
11. Explain your previous work in database design.
12. How would you explain a schema?
13. What are the different schemas that database designers commonly use?
14. What are the five primary types of dimension?
15. Explain the difference between DBMS and RDBMS.
16. Explain normalization and denormalization as they apply to database design.
17. How would you explain scalar and aggregate functions?
18. What is OLTP?
19. What is OLAP and how does it differ from OTLP?
20. What is collation sensitivity and what are the different types?
21. Define the terms "table" and "field" as used in data collection and storage.
22. Explain the backup procedures you might use to prepare for data loss.
23. What DBMS are you familiar with?
24. What is a clustered versus a non-clustered index?
25. Define the terms "query" and "subquery."

[**Get interview-ready with tips from Indeed**](https://www.indeed.com/career-services/interview-help?collectorID=careerguidepromo-US&from=careerguidepromo-US)

Prepare for interviews with practice questions and tips

## 6 database design interview questions

Here are six database design interview questions and sample answers to help you practice for your next interview:

### 1. How would you describe database design and its role in business?

This is a general question that interviewers may ask early in an interview. The purpose of this question is often to assess your understanding of basic database design concepts and learn about your approach to work. This question can also help interviewers determine how your approach to database design can contribute to the overall success of the company.

When answering this question, it's often helpful to avoid specifics and to focus on the role of data and databases in the current business environment. You can give a brief explanation of why companies collect data and why it's important. You can then describe how databases and database designers allow businesses to make better use of the data they collect.

**Example answer:** “*Database design is the technique professionals like myself use to create a detailed data model of a database. This involves making physical and logical design decisions, such as setting the physical storage parameters required to develop a design in a data definition language. I can then use this information to generate a new database.*  
*Throughout this process, I consider various elements, such as the availability of data, data replication, database partitioning, data backup and the system's security. The purpose of this process in business is to identify relationships between various entities in the data and normalize the data so a company can understand, manage and analyze the patterns within its data.”*

### 2. In general terms, what are the primary steps in database design?

Database design generally takes place in three phases, and familiarity with these steps is essential for employees who work in the field. This general question can help interviewers evaluate your knowledge of important database design concepts. It can also give them an idea of how you might approach building a database if the company hires you.

When answering this question, it's often helpful to demonstrate your knowledge while keeping your answer concise. This can show employers that you're ready to move on to more advanced topics. You can begin by describing the three primary steps in database design and then explain why these phases are important.

**Example answer:** *"Database design generally includes into three stages. The first stage is the conceptual database design, in which we build a conceptual model of the data system. This includes identifying the entities within the data ecosystem, as well as their relationships and attributes. The second stage is logical database design. In this stage, we choose a data model and translate the conceptual model into a logical framework.*

*The final step of database design is creating a physical database model. At this stage, we find ways to physically implement the logical model. This includes choosing a database management system, determining security measures and developing storage mechanisms and file organization systems."*

**Related:** [**What Are the Different Types of Database Management?**](https://www.indeed.com/career-advice/finding-a-job/what-is-database-management)

### 3. What is a logical model and what are the steps of developing one?

A logical database model is often the second step in developing a database solution. It allows designers to apply a certain data model to their concept before designing a database that suits a company's needs. Hiring managers may ask you this question to assess your understanding of how to complete this important task.

When answering this question, you can briefly describe what a logical model is. You can then continue by explaining what role it has in the database design process. Finally, you can briefly list the steps you would take when creating a logical model. This can help you show interviewers that you have a practical understanding of this process.

**Example answer:** “*A logical database model, also called a logical schema, is a data model that focuses on a particular problem domain expressed separately from a specific database storage technology or management product. This model uses data structures, such as object-oriented classes, relational tables and columns or XML tags to relay information.*

*I can use seven steps to develop one. I define the goals for database logical model design, then analyze the existing database, create a database structure and establish the table relationships of the model. Then, the final three steps include identifying and outlining the business rules of the model, defining the database model views and finally checking the integrity of the schema I created.”*

**Related:** [**Client Database Systems: Definition, Benefits and Types**](https://www.indeed.com/career-advice/career-development/client-database)

### 4. Explain what a conceptual database model is and explain its features.

Creating a conceptual database model is often the first stage of database design. It gives a generalized overview of the requirements of a database. Understanding conceptual database models is an important ability for professionals who work in database design, and your answer to this question can help interviewers understand your qualifications in this area. When you answer this question, consider starting with a brief description of what a conceptual database model is and what role it has in the database design process. Next, you can explain some of the primary features that a conceptual model needs to be effective.

**Example answer:** “*A conceptual database model is a description of a company's database requirements that I can then use to design and implement a database that takes its priorities and preferences into account. This model typically includes the primary concepts that a business wants to focus on and the relationships between these concepts.*

*The fundamental purpose of this kind of model is to provide a data-centered overview of a company. It does this by analyzing how different organizational entities relate to each other to support business processes and recording internal business events. Instead of trying to understand a database's physical characteristics or processing flow, this model only focuses on recognizing the data used within an organization. Primary features of a conceptual database model include data types, names and the characteristics of business entities and their qualities.”*

### 5. What is a physical database model and how might you design one?

Physical database models are often the last stage of database design. These models represent the actual structures of the database and combine the logical and conceptual components of the design. Completing a physical database model is one of the most important tasks for database designers and interviewers may ask you this question to evaluate your expertise in the field. Your answer can also help them get a better understanding of how you might perform if the company hires you.

When answering this question, you can begin by describing the concept of a physical database model. You can then describe what role it has in the process of database design. Finally, you can briefly describe the steps that you might take when creating a physical database model. If you have completed one in the past, you can include details to demonstrate your experience.

**Example answer:** “*A physical database model is a database-specific tool that includes all the logical database elements professionals require to build a database, such as a column name, column data type, primary key, foreign key, column constraints and relationships between tables. This model is a representation of a data design as professionals implemented it or hope to implement it within a database management system. Depending on the project, it may derive from a logical database model.*

*To create a physical database model, I follow a series of eight steps. I identify tables and normalize them, then I define columns and review stored procedures. Next, I apply naming conventions according to the organization's standards and guidelines and then identify relationships between tables. Finally, I apply data model patterns and assign relevant keys.”*

**Related:** [**How To Become a Database Manager**](https://www.indeed.com/career-advice/finding-a-job/how-to-become-database-manager)

### 6. What are the features of a physical database model?

A physical database normally has several distinct features that separate it from logical and conceptual database models. It's important that database designers understand these features so that they can incorporate them into their designs. When asking this question, interviewers are often testing your knowledge of common database design concepts and your ability to use them in a practical way.

When answering this question, it's often helpful to offer precise but brief information. Try listing four or five attributes that are important for physical database models. If you have additional time, you can consider describing why these attributes are important and what role they play.

**Example answer:** *"A physical database model is a plan for how to implement a logical model in a real-world setting. There are several features that distinguish physical models from logical models. First, they're DBMS-specific, while logical models are DBMS-independent. Physical database models also specify storage structures, access methods, security features, table structures and table constraints. These elements are all part of creating a functional and secure database that professionals can use in a work setting."*

*Please note that none of the companies mentioned in this article are affiliated with Indeed.*

# Design Interview Questions

[Radu Gheorghiu](https://vertabelo.com/authors/radu-gheorghiu/)

[Radu works as a full-time data engineer. He loves SQL and anything data-related. In his spare time, he likes to put his data to work by building data pipelines. He even trains machine learning models to predict the best time of the day to drink his coffee. He also enjoys helping others and sharing his knowledge on our blog and Stack Overflow.](https://vertabelo.com/authors/radu-gheorghiu/)

Tags:

* [database design](https://vertabelo.com/tags/database-design)

Got a job interview coming up? If you know you’ll be talking about database design, make sure you can answer these questions on database modeling and schemas.

Job interviews typically cover core knowledge areas, but the interviewer also often throws in some questions on tricky topics. We’ll help you prepare by going over these common database schema design questions that cover both basic and more advanced topics.

## Basic Database and Data Modeling Questions

The following database schema design interview questions can be considered basic or beginner level. You should be able to answer them fully or at least partially to demonstrate that you are knowledgeable on the topic. We’ve also provided some examples of answers to those questions, but more details would help you stand out.

### 1. What is a data model?

Answer: A data model is a way of organizing and storing data in a database. It determines how data is organized and structured and how it relates to one another. Some common database models include the relational model, the object-oriented model, and the hierarchical model.

### 2. What are the three types of data models for relational database design?

Answer: There are [**THREE MAIN TYPES OF DATA MODELS**](https://vertabelo.com/blog/conceptual-logical-physical-data-model/) that are commonly used when designing a relational database:

* Conceptual data model: This is a high-level model that represents the overall structure of the data, including the entities and the relationships between them. It is used to communicate the overall design of the database to stakeholders and does not include implementation details.
* Logical data model: This is a more detailed model that represents the structure of the data as it will be stored in the database. It includes more implementation details and specifies the attributes and data types for each entity.
* Physical data model: This is the most detailed model; it represents how the data will be physically stored on the database server. It includes information about the specific database management system (DBMS) that will be used, the storage structures, and the indexes and other performance-enhancing features that will be implemented.

These three data models are typically developed in a top-down fashion, starting with the conceptual model and then progressing to the logical and physical models as the design becomes more refined.

### 3. What is a database diagram?

Answer: [**A DATABASE DIAGRAM**](https://vertabelo.com/blog/what-is-database-diagram/) is a visual representation of a database schema. It shows the relationships between different entities (tables or other structures) in the database, and how the data is organized and related.

A database diagram can be used to understand the overall structure of a database and how the different entities are related to each other. It can also be used to design and plan the structure of a database, as well as to communicate the design to others.

There are several types of database diagrams. These include entity-relationship diagrams (ER diagrams), which show the relationships between entities, and object-oriented diagrams, which show the relationships between objects in an object-oriented database.

### 4. What are some best practices when designing a database?

Answer: There are [**SEVERAL BEST PRACTICES THAT ARE GENERALLY RECOMMENDED WHEN DESIGNING A DATABASE**](https://vertabelo.com/blog/database-design-guide/):

* Use a clear and consistent naming convention: It is important to have a clear and consistent naming convention for database objects, such as tables and columns. This helps to make the database more understandable and easier to use.
* Normalize the data: Normalization is the process of organizing the data in a database to minimize redundancy and dependency. Normalized databases are generally more efficient and easier to maintain.
* Use appropriate data types: It is important to use appropriate data types for each column in a table. Using the correct data type helps to ensure that the data is stored and processed efficiently and accurately.
* Define primary and foreign keys: A primary key is a column that uniquely identifies each row in a table (e.g. a student ID number, a driver’s license number, etc.) A foreign key refers to another table’s primary key and establishes the relationship between two tables. It is important to define these keys carefully to ensure that the relationships between tables are correctly enforced.
* Use indexes to improve performance: Indexes improve the performance of database queries by allowing the database server to quickly locate the data it needs. However, it is important to use indexes wisely, as too many indexes can negatively impact performance.
* Test and optimize the database design: It is important to test the database design and optimize it as needed to ensure that it performs well and meets the needs of the application. This may involve fine-tuning queries, adding indexes, or making other changes to the database structure.

### 5. What is normalization?

Answer: [**NORMALIZATION**](https://vertabelo.com/blog/normalization-1nf-2nf-3nf/) is the process of organizing the data in a database to minimize redundancy and dependency. It involves breaking down data into smaller, more atomic pieces and organizing them into separate tables. The goal of normalization is to create a more efficient and flexible database design that is easier to maintain and modify over time.

### 6. What is denormalization and what is its purpose?

Answer: Denormalization is a process of intentionally adding redundancy to a database design in order to improve performance. It is used to reduce the number of joins required to retrieve data and improve query performance.

Denormalization is often used in read-heavy environments, such as data warehousing or OLTP systems, but can make the database more complex and prone to inconsistencies. It should be carefully considered and only used when the benefits outweigh the potential drawbacks.

### 7. What is a surrogate key and how is it different from a primary key?

Answer: A surrogate key is a unique identifier that is artificially generated by the database and used to uniquely identify each row in a table. It is often used in place of a natural primary key, which is a column or set of columns that naturally and uniquely identifies each row in the table.

Surrogate keys are often used when a natural primary key does not exist, is too long or complex, or may change over time. They provide a stable and efficient means of identifying rows in the table and establishing relationships between tables.

### 8. How can you enforce that only data that is valid from a business point of view is inserted in database tables and/or columns?

Answer: There are several ways to enforce the insertion of valid data only; the most common is to use constraints. Constraints are rules that are applied to a database table or column to enforce data integrity. They can be used to specify the types of data that are allowed in a given column as well as to enforce other rules like uniqueness or foreign key relationships. Some examples of constraints that can be used to enforce data integrity include NOT NULL, UNIQUE, PRIMARY KEY/FOREIGN KEY, and CHECK constraints.

### 9. What is cardinality in data modeling?

Answer: [**IN DATA MODELING, CARDINALITY REFERS TO**](https://vertabelo.com/blog/cardinality-in-data-modeling/) the relationship between two entities in a database. It can be one of three types: one-to-one, one-to-many, or many-to-many. The type and direction of the cardinality determines how many records in one table (or entity) can be associated with a record in the other table. Cardinality is important in defining the structure and relationships between entities in a database.

### 10. Give me some examples of index types and briefly explain how they impact  performance and how they work in relation to the data model.

Answer: There are several types of indexes that can be used to improve the performance of a database, including clustered, non-clustered, and full-text indexes.

Clustered indexes physically rearrange the data in the table to match the index order, while non-clustered indexes create a separate data structure with index key values and pointers to the rows in the table.

Full-text indexes are used to search large amounts of text data. Indexes can be created on specific columns or combinations of columns in a table and can improve the performance of queries by allowing the database to quickly locate the requested data. However, it's important to carefully consider which indexes to create, as adding too many indexes can negatively impact the performance of insert, update, and delete operations.

### 11. What is a schema in a database?

Answer: A schema in a database is a logical structure that represents the organization of data in a database. It defines the tables, columns, and relationships between the data, as well as the data types and other constraints for each column.

A schema can be thought of as a blueprint for the database, providing a clear and organized structure for storing and accessing data. Schema design is an important aspect of database development, as it affects the performance, security, and overall functionality of the database.

### 12. What is the difference between OLTP and OLAP?

Answer: OLTP (Online Transaction Processing) and OLAP (Online Analytical Processing) are two different types of database systems that are optimized for different types of tasks.

OLTP systems are designed to handle a large number of short, transactional queries quickly and efficiently. These systems are used to store and retrieve data for day-to-day business operations, such as processing transactions or updating customer information.

OLAP systems, on the other hand, are designed to handle complex, analytical queries that involve large amounts of data. These systems are used to analyze and summarize data for business intelligence and decision-making purposes, such as identifying trends or forecasting future performance.

In general, OLTP systems are optimized for inserting, updating, and deleting small amounts of data quickly; OLAP systems are optimized for querying large amounts of data for analysis.

### 13. What are fact and dimension tables?

Answer: Fact tables contain measurements or metrics that are tracked, while dimension tables contain descriptive attributes or characteristics of the data. Fact and dimension tables are used together in a data warehouse to provide insights and context for the data.

### 14. What is a composite key in a data model?

Answer: A composite key is also known as a compound key or a concatenated key. It is a combination of multiple attributes that can together uniquely identify a row in a table. For example, the combination of first name, last name, and birth date could be used as a composite key; neither of these three attributes could uniquely identify a person, but together they could. Thus, composite keys are often used in data models when there is no single attribute that could uniquely identify a record. They can also be used to enforce relationships between tables in a database. For example, a composite key could be used to link a fact table to a dimension table in a data warehouse.

### 15. What is an ERD?

Answer: An entity-relationship diagram (ERD) is a graphical representation of the relationships between entities in a database. It is used to design and model databases and to communicate the structure of a database to stakeholders. ERDs use symbols and notation to represent the entities, attributes, and relationships in the database.

### 16. Give an example of a database antipattern.

Answer: One example of a database antipattern is not having a primary key in a table.

A primary key is a field that uniquely identifies each row in a table. It is important to have a primary key in a database table because it helps ensure the integrity of the data and allows for the easy identification and organization of records.

Not having a primary key can lead to several problems in a database. For example:

* It can make it difficult to identify and update specific records, as there is no way to uniquely identify them.
* It can lead to data inconsistencies, as multiple records may have the same data but there is no way to differentiate between them.
* It can make it difficult to enforce data integrity, as there is no way to ensure that each record has a unique value for a particular field.

Overall, not having a primary key can make it difficult to manage and maintain a database. It can lead to a variety of problems that can impact the accuracy and reliability of the data.

## Advanced Data Model / Database Schema Design Questions

Advanced-level database design interview questions typically have topics that revolve around data modeling and various details; this is to test the depth of the candidate’s knowledge. Some advanced questions are below:

### 1. What is the difference between the [star schema and the snowflake schema](https://vertabelo.com/blog/data-warehouse-modeling-star-schema-vs-snowflake-schema/) in database design?

Answer: In a star schema, a central fact table is surrounded by a number of smaller dimension tables. The fact table contains the measure or fact data, while the dimension tables contain descriptive attributes that provide context for the measures. A star schema is a simple and efficient design that is easy to understand and query.

A snowflake schema is a variation of the star schema. In this schema, the dimension tables are further normalized into multiple tables. This results in a more complex design with more tables, but it can provide more granularity and flexibility in querying the data.

The main difference between a star schema and a snowflake schema is the level of normalization. A star schema is more denormalized, with fewer tables and a simpler design. A snowflake schema is more normalized, with more tables and a more complex design. The choice between a star schema and a snowflake schema depends on the specific requirements and trade-offs of the database design.

### 2. What are some common database design errors?

Answer: There are several common database design errors:

* Not normalizing the data: Normalization is the process of organizing data in a way that reduces redundancy and dependency. Failing to normalize the data can result in data inconsistencies and update anomalies.
* Not choosing the right data types: Choosing the wrong data types for the data can result in data corruption or loss of precision. For example, using a small integer data type to store large numbers could cause data to be truncated or lost.
* Using too many indexes: Indexes can improve the performance of queries, but using too many indexes can negatively impact the performance of insert, update, and delete operations. It's important to carefully consider which indexes are necessary and which can be removed.
* Not defining primary keys: Every table should have a primary key, which is a unique identifier for each row. Failing to define a primary key can result in data inconsistencies and difficulties in querying and updating the data.
* Not defining foreign keys: Foreign keys are used to enforce relationships between tables. Failing to define foreign keys can result in data integrity issues and difficulties in querying the data.
* Not considering security: It's important to consider the security of the data and implement appropriate measures, such as encryption, access controls, and backup and recovery processes.

### 3. What is collation and how does it impact the data model design?

Answer: Collation refers to the rules and characteristics that define how data is sorted and compared in a database. Collation can impact data model design in several ways:

* It determines the order in which data is sorted and displayed, which can affect how data is queried and analyzed.
* It determines how data is compared, which can impact the way data is searched and indexed.
* It determines the case sensitivity of data, which can affect the way data is matched and joined.
* It determines the character set and language of data, which can impact how data is displayed and stored.

It's important to choose the appropriate collation for the data in the database, as it can impact the performance, functionality, and accuracy of the database.

### 4. What is a data mart?

Answer: A data mart is a subset of a data warehouse that is tailored to a specific business function or subject area. It provides targeted data to a specific group of users (i.e. a department or business unit) for decision-making purposes.

Data marts can be created from a subset of the data in a data warehouse or from scratch by extracting data from operational databases or other sources. They are typically smaller and simpler than data warehouses and are designed for use by a specific group.

### 5. What are the steps of designing a conceptual data model?

Answer: The key to this answer is focusing more on communication with the business stakeholders. However you decide to phrase your answer and approach the technical solution, it’s always important to keep close communication with the business team.

As for the technical steps, they generally include:

1. Define the purpose and scope of the data model. This involves identifying the business requirements and goals of the data model and determining the types of data that need to be included.
2. Identify the entities and their attributes. This involves identifying the key entities and the characteristics or attributes of each entity that need to be captured in the data model.
3. Define the relationships between the entities. This involves identifying the relationships between the entities and determining how the entities are connected to each other.
4. Refine and validate the data model. This involves reviewing and testing the data model to ensure that it accurately represents the data requirements and meets the needs of the business.
5. Document the data model. This involves creating a clear and concise document that describes the entities, attributes, and relationships in the data model. This document can be used as a reference for database design and implementation.

### 6. What’s the difference between clustering and partitioning?

Answer: Clustering and partitioning are both techniques that can be used to organize data in a database. However, they are used for different purposes and work in different ways.

Clustering is a method of organizing data in a database so that data that is frequently accessed together is physically stored together. This can improve the performance of queries that access the data, because the data is stored in a contiguous block rather than being scattered across the disk. Clustering is often used to improve the performance of data warehouses, where large amounts of data are accessed in bulk.

Partitioning is a method of dividing a large table or index into smaller, more manageable pieces, called partitions. Each partition is stored in a separate file or set of files, and can be managed and accessed independently of the other partitions. Partitioning is often used to improve the scalability and manageability of large databases by allowing different partitions to be processed in parallel and by making it easier to perform maintenance tasks on the data.

### 7. What is a junk dimension in data modeling?

Answer: A junk dimension is a type of dimension table used to store miscellaneous, low-cardinality attributes in a data warehouse. It is typically implemented as a single table with many columns and is used to store data that does not fit into any other dimension table. Junk dimensions are often used to store data about customer feedback or other low-cardinality data that is not used frequently but that you still want to keep.

## Best of Luck on Your Database Design Interview!

Interviews are always varied. Fortunately, database schema design interview questions mostly follow this pattern of basic questions for beginner roles. Whether you’re called on to answer more advanced topics will always depend on your interviewer and their technical expertise.

There’s no way to know all potential database interview topics in depth. Even so, focus on your strengths and be open about your experience. Be confident in your knowledge, but also brave enough to admit your less-knowledgeable areas. In my experience, this is much better than trying to always be correct.

We hope the list of questions we shared above and the answers were helpful in getting you prepared for your next database job interview!

# A Data Model for a Freelancers Collective

[Emil Drkušić](https://vertabelo.com/authors/emil-drkusic/)

[Database designer and developer, financial analyst.](https://vertabelo.com/authors/emil-drkusic/)

Tags:

* [collaboration](https://vertabelo.com/tags/collaboration)
* [example data model](https://vertabelo.com/tags/example-data-model)
* [example ER diagram](https://vertabelo.com/tags/example-er-diagram)
* [example ERD diagram](https://vertabelo.com/tags/example-erd-diagram)
* [database model](https://vertabelo.com/tags/database-model)
* [template](https://vertabelo.com/tags/template)

Freelancing is becoming more and more popular these days. While most freelancers are a one-man band, that’s not the only option. You could be a part of a collective and collaborate on larger and more complex projects. A data model that could power a freelancers collective’s app is the topic of today’s article.

Freelancing is not new, but it’s becoming more and more popular. Working from 9:00 to 17:00 has certain advantages, but it also comes with many disadvantages. Therefore, an increasing number of people decide to become freelancers.

Freelancers work mostly in creative industries (e.g. writers, graphic designers, translators, interpreters), in the IT industry, or on IT-related jobs. In this article, we’ll focus on IT-related projects, but the model could be used for other complex projects where collaboration is essential.

In many cases, IT projects require more than one person. If you want to build a team to complete the entire project, you’ll definitely need a way to collaborate with them. It would be great if you personally knew freelancers with the desired skills and if those freelancers had the time and desire to work on your project. But that usually is not the case, so we’ll need an IT solution that allows a large number of freelancers with different skills to work together on a project. And they won’t need to be located in the same place or even know each other.

The idea is that freelancers (and customers) could use a system that would allow them to collaborate on many different projects. It wouldn’t be a freelancing platform (e.g. Upwork) or a project management tool, but something in between the two.

### The Data Model

The data model that could run one such application is pictured below. We’ll focus only on the most important application elements, so we won't go into features like chats, payment processing, and project management tools in this article.

EDIT MODEL IN YOUR BROWSER

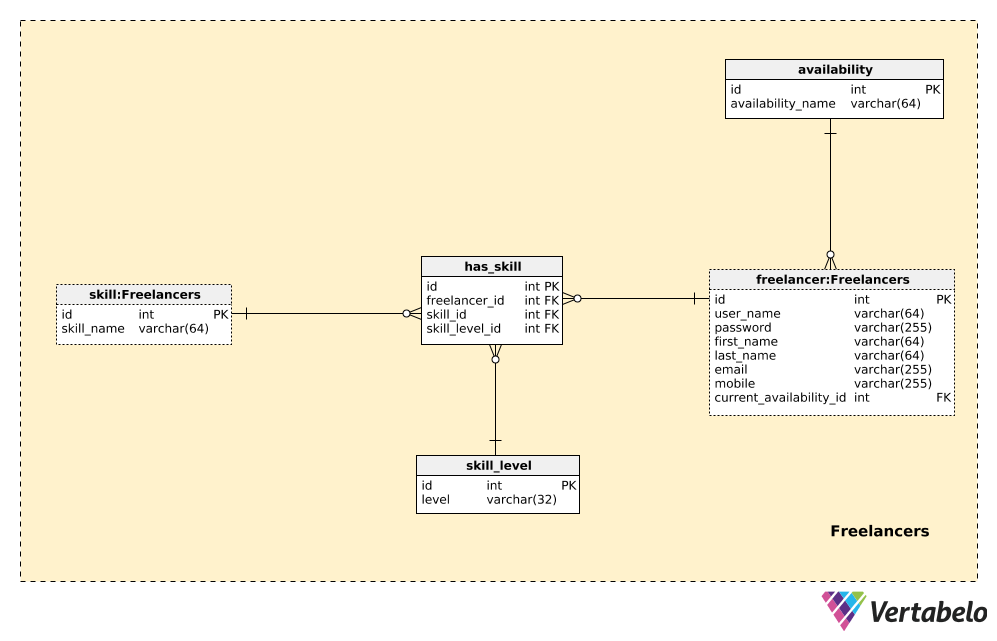
The model consists of four subject areas:

* Freelancers
* Customers & Projects
* Teams
* Project phases

We’ll describe each of these subject areas in the order they are listed.

### Section 1: Freelancers

The first and most important subject area is Freelancers. This is where we’ll store information for all the freelancers registered in our application, including their skills.

[](https://vertabelo.com/blog/a-data-model-for-a-freelancers-collective/uploads/freelancers.png)

The central table here is the freelancer table and it contains a list of all freelancers who are members of our collective. They can log into our application (via web, mobile, or both) and collaborate on different projects. For each freelancer, we’ll store:

* **user\_name** – A UNIQUE username used in the sign-in process.
* **password** – A password hash value.
* **first\_name** and **last\_name** – The freelancer’s first name and last name.
* **email** and **phone** – The freelancers’ contact details.
* **current\_availability\_id** – References the availability dictionary and denotes if the freelancer is currently available and in what way (i.e. full-time, less than 20 hours a week, etc).

A list of all possible availability types is stored in the availability dictionary. This table contains only one UNIQUE value – **availability\_name**. This value should be set by the freelancer according to their current engagements.

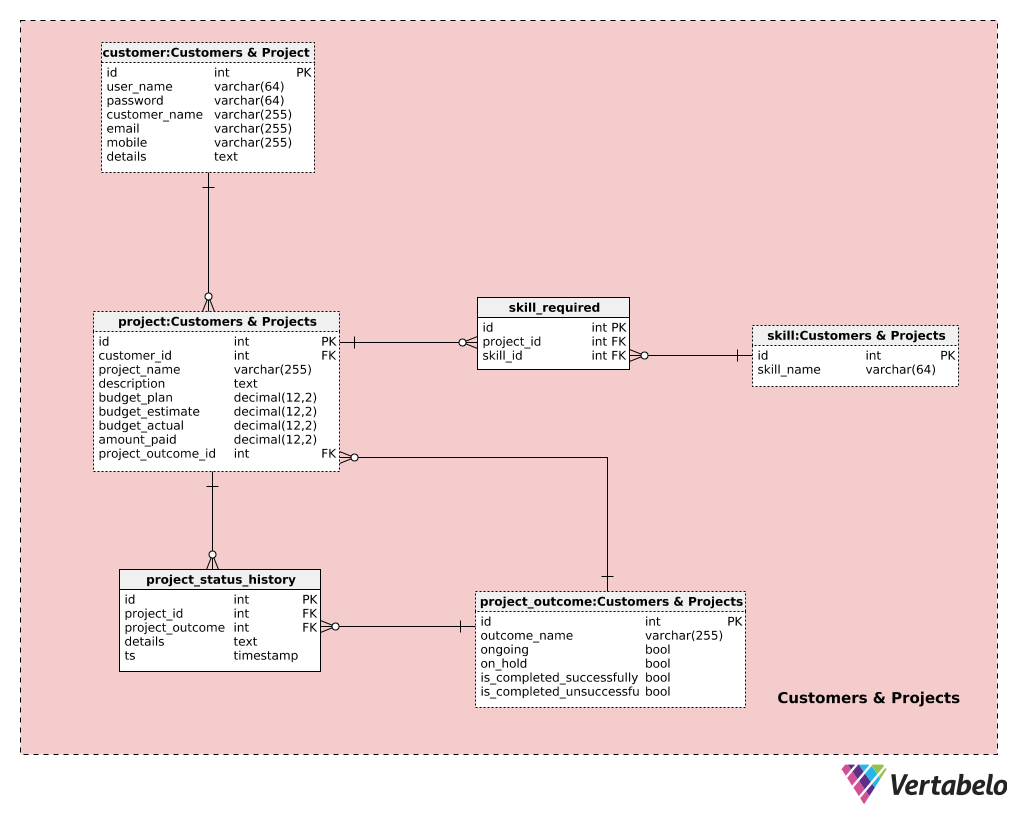
The next thing we need to store in our database is a list of freelancers’ skills and their skill levels. Skills can be used in combination with freelancers’ availability to find the best fit for new projects. We’ll use three tables to handle this requirement.

The first one, skill, is a simple dictionary containing the **skill\_names** of all possible skills we expect we’ll need on any project. This list can include technologies, but also other skills like project management, writing, or design. Next, we need to define all the skill levels we could have. We could use numbers (e.g. from 1 to 10), or words like “basic”, “advanced”, “expert”. No matter which option we choose, we’ll need a dictionary to store these values. In our model, that is the skill\_level dictionary and it also contains only one UNIQUE value, **level**.

The last table in this subject area, the has\_skill table, is the one that will relate the freelancer, skill, and skill\_level tables. It contains only references to these three tables. For each UNIQUE pair of **freelancer\_id** – **skill\_id**, we’ll store the current **skill\_level\_id**.

#### Section 2: Customers and Projects

Besides freelancers, we need customers that will use our system to manage their projects. We’ll cover all of that in the Customers & Projects subject area. We have a copy of the skill table here, but we’ve already discussed it. The other four tables are new, and we’ll explain them here.

[](https://vertabelo.com/blog/a-data-model-for-a-freelancers-collective/uploads/customers-projects.png)

We’ll start with the customer table. Customers are also registered users of our application. They register on a different form than the one freelancers use. For each customer, we’ll store:

* **user\_name** – A UNIQUE username used during the login process.
* **password** – A password hash value.
* **customer\_name** – The customer’s full name.
* **email & mobile** – The customer’s contact details.
* **details** – All additional customer details.

Customers can post their projects and include all the relevant details. All projects are stored in the project table. For each project, we’ll have a:

* **customer\_id** – A reference to the customer initiating the project.
* **project\_name** – The name chosen by the customer for that project.
* **description** – A full description of the project, written by the customer.
* **budget\_plan** – The planned budget for the project. This value, together with the previous two values, will be inserted by the customer when the project is created. This should give the collective at least an idea of where the customer stands financially.
* **budget\_estimate** – The collective’s estimated price for this project. This value is inserted after the project has been viewed. It should give the customer a feeling of how much the project will actually cost and the difference between their planned budget and the collective’s estimate.
* **budget\_actual** – The actual budget for this project. This amount shall be defined along the way, according to the agreement between the customer and the collective. This could be defined at the start (i.e. a fixed-price project), but changes could be made as project requirements change.
* **amount\_paid** – The actual amount paid for this project. This will be updated throughout the project, and after the project is completed successfully it should match the **budget\_estimate** amount.
* **project\_outcome\_id** – A reference to the current status of this project. After the project is initialized, this status will be that it’s a new project. All changes in status after that point will be based on actions taken by customers or by freelancers.

Besides the project description, the customer should also insert a list of all the skills required for this project. This list could be later edited by the collective or the customer (e.g. when the customer wasn’t initially sure about all skills needed). In the skill\_required table, we’ll store a list of UNIQUE **project\_id** – **skill\_id** pairs.

To track historical changes during the project, we’ll have two more tables. One is a catalog of all possible project outcomes, and the other contains historical data.

The list of all possible outcomes is stored in the project\_outcome dictionary. Outcomes could vary from the negotiation phase all the way to project close. We could have outcomes like “negotiation phase”, “project started”, “project paused by client”, “project paused by collective” , “in progress”, “delivered”, “failed” etc. For each outcome, we’ll define a UNIQUE outcome\_name and set one of four values -- **ongoing**, **on\_hold**, **is\_completed\_successfully**, **is\_completed\_unsuccessfully** -- to True.

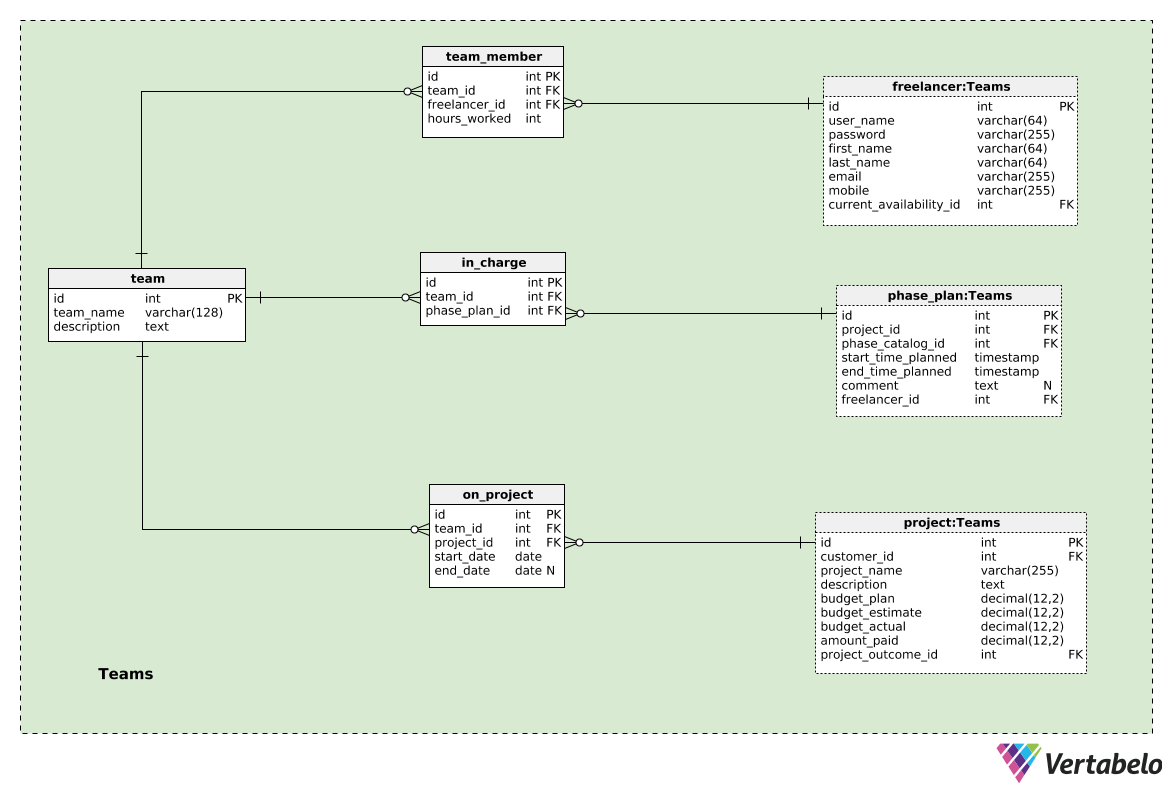
The final table in this subject area contains the history of the project. For each record in the project\_status\_history table, we’ll store:

* **project\_id** – References the related project.
* **project\_outcome\_id** – References the related project outcome.
* **details** – All project details, inserted in a textual format.
* **ts** – The timestamp when this outcome was inserted.

Records in this table are inserted either automatically, after a certain phase is completed, or manually.

#### Section 3: Teams

So far, we have freelancers and their skills and we have projects inserted by clients. Now let’s discuss how the collective forms a team and assigns it to a project. In this subject area, we’ll look at the structure needed to achieve that. We already met the freelancer and project tables in previous subject areas, so we won’t discuss them here.

[](https://vertabelo.com/blog/a-data-model-for-a-freelancers-collective/uploads/teams.png)

Forming a team is the main advantage of joining forces in a collective. I’ll go with the assumption that we’ll form a new team for each project, but I’ll also leave it open to the possibility that the same team could work on multiple projects. The team itself is stored in the **team** table. For each team, we’ll store the **team\_name** as well as its **description**.

Each team is composed of team members, and they are stored in the team\_member table. For each UNIQUE combination of **team\_id** – **freelancer\_id**, we’ll store a value denoting the number of **hours\_worked** on that project.

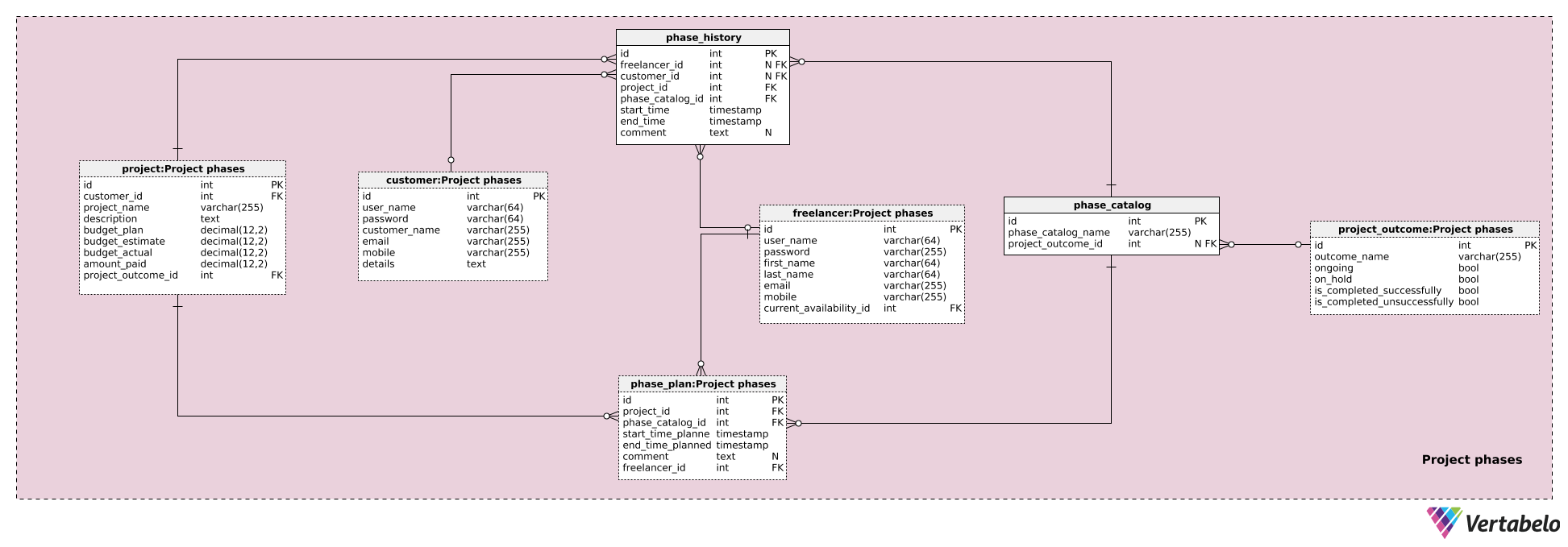
Teams are assigned to projects. That relationship is stored in the on\_project table. We’ll again have references to the team table (**team\_id**) and the project table (**project\_id**). Besides that, we’ll define the **start\_date** and the **end\_date** of the time when that team was working on that project. The **team\_id** – **project\_id** pair is NOT UNIQUE, and that allows us to assign the same team to the same project more than once, although, of course, in different time periods. We should programmatically check that there is no overlapping when the same pairs are used.

We’ll talk about phases in the next section, but it’s enough to know at this point that the phase\_plan table is used to store the list of planned phases.

Now we can relate teams and planned phases by storing the UNIQUE pair **team\_id** – **phase\_plan\_id** in the in\_charge table. Please notice that more than one team could be in charge of some phase and this could be the desired behavior.

#### Section 4: Project Phases

In the last subject area, we’ll talk more about project execution. We have discussed four of the tables in this section before: freelancer, customer, project, and project\_outcome. The remaining tables are new.

[](https://vertabelo.com/blog/a-data-model-for-a-freelancers-collective/uploads/project-phases.png)

We’ll start with the phase\_catalog table. This is where we’ll list all possible phases we could have during any project. We can’t know all the possible phases upfront (though we can assume most of them). If we need a new one, we’ll simply add a new value in this table. Each record is also UNIQUELY defined by the **phase\_catalog\_name** and could have the **project\_outcome\_id** defined. Some possible project phases are “new project inserted by client”, “project revised”, “proposal sent to client”, “client responded” (all these belong to the “negotiation phase” outcome); “data model developed”, “back-end development - in progress”, “front-end development – in progress” (all part of the “in progress” outcome), etc.

If a project outcome is defined, insertions and changes related to that phase will also trigger changes in the project\_status\_history table and a change of the **project.project\_outcome\_id** attribute value. For example, if the project previously had an outcome status set to “project paused by client”, starting a certain phase could change the current project outcome to “in progress”.

The remaining two tables are related to project phases. We’ll separately store our plan as well as how phases were actually executed.

The plan is stored in the phase\_plan table, which we’ve already mentioned in a previous subject area. We’ll store our original plan of how the project should be divided into phases and how each phase should be executed. For each record in this table, we’ll store:

* **project\_id** – The ID of the related project.
* **phase\_catalog\_id** – The ID of the related phase name.
* **start\_time\_planned** and **end\_time\_planned** – The start and end times planned for that phase.
* **comment** – An optional comment, inserted at the time we insert this record. This is a good place to define the details of what we want to achieve during this phase.
* **freelancer\_id** – References the freelancer who inserted this record.

The last table in this subject area and in our model is the phase\_history table. It has almost the same structure as the phase\_plan table. There are two important differences:

1. We have the **freelancer\_id** and **customer\_id** attributes here. Only one of them can be set at a time, and that will denote who inserted this record. This is most likely to be a team member, but the customer may do it.
2. The **start\_time** and **end\_time** attributes denote the actual start time of that phase and the actual end time. While **start\_time** will be set when we insert a record, the **end\_time** attribute will be set later.
3. The comment attribute can be used to insert the outcome of this phase.

Inserting or changing a record in this table (e.g. setting **end\_time**) will trigger an insert or update of the project outcome if that phase has a related outcome in the phase\_catalog. Please note that we could have two phases running at the same time, e.g. the development of a web application and a mobile application could both start after we completed work on the database. Therefore, some intervals during the planning and execution phases could overlap.

### Let’s Talk About the Freelancers’ Collective Data Model

Today we have discussed a data model that could be used by a freelancers’ collective to collaborate together on complex projects. Have you ever been a part of such a collective? Would you add something to the model? Or remove something from it? Please tell us in the comments below.

# How to Prepare a Database Model

[Vertabelo Team](https://vertabelo.com/authors/vertabelo/)

Tags:

* [database model](https://vertabelo.com/tags/database-model)
* [ER diagram](https://vertabelo.com/tags/er-diagram)

Create good logical, physical, and conceptual data models using these expert database model preparation tips.

To put it simply, a database model is a data model that determines the logical or physical structure of a database. Database design is the process of creating a database model. A database model is used to capture information about the data that must be stored in a database.

If you're a bit unsure about the steps involved in the database design process, I suggest you read this [**DESCRIPTION OF DATABASE DESIGN STEPS**](https://www.vertabelo.com/blog/steps-in-database-design/). And if you are unsure about how to perform these steps, check out these [**TIPS FOR BETTER DATABASE DESIGN**](https://vertabelo.com/blog/9-tips-for-better-database-design/).

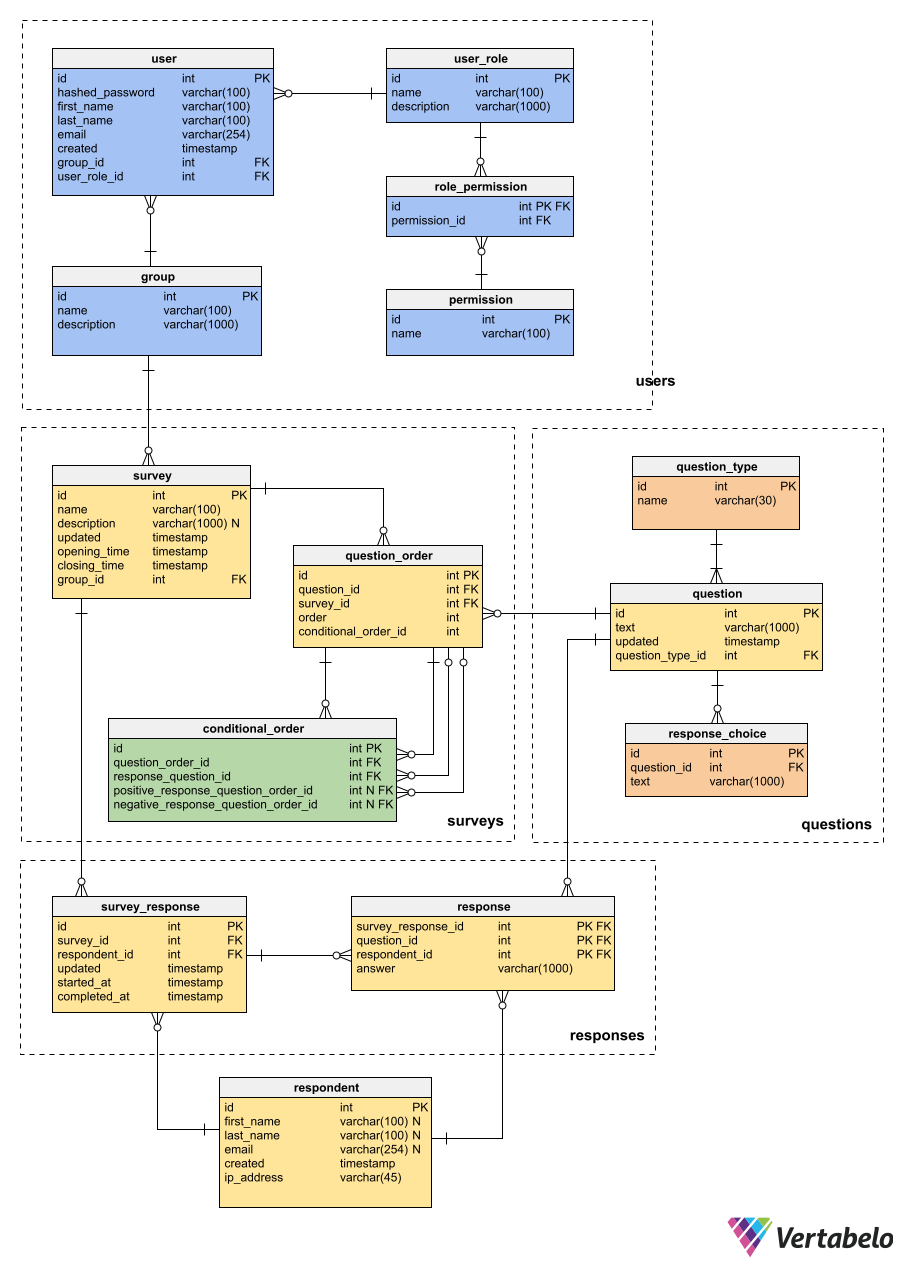
In the data modeling process, Step 1 is gathering business requirements. This is where the database model is key.

## Start Your Model

To create the model, you first need to talk with business users, which will help you understand what the requirements are. They will advise you on the information that you need to store. This will help lay the foundation for tables and their attributes (columns) and how the information is linked together (the relationships between tables).

In addition to the business domain and its associated challenges, you need to know how the database and its data will be used. For example, if you are storing customer data, what information do you need to record Why are you storing customers’ addresses – will you be sending them invoices? If so, how? By snail mail or electronically? The answer to those questions will impact your data model by indicating what information is mandatory or optional.

Once you fully know the requirements, you can start getting your modeling ideas together. You could write out the model for your database, but that would probably need a lot of words. Remember: A picture is worth a thousand words. That is where a database model diagram, referred to as an entity-relationship diagram (ERD), is used. This "picture" of a database model provides a straightforward way to communicate between business users, data modelers, and the development team. ERDs use a standard notation (boxes, lines, etc.) to document the database design. You can also find additional information about [**ERD SYMBOLS HERE**](https://vertabelo.com/blog/symbol-in-erd-diagram/).



However, [**DATABASE DESIGN IS MORE THAN JUST AN ERD**](https://vertabelo.com/blog/database-design-more-than-just-an-erd). A good database model needs to go beyond the ERD. Unfortunately, you may find data modelers that think that their only contribution is the ERD. They attempt to create a data model that is a work of art, while some data modelers just worry about creating a few boxes and lines. The ERD is a key contribution from the data modeler, but it is also essential to keep the model up to date throughout the software development lifecycle. (More on that later in this article.)

## Ingredients of a Data Model

Depending on the type of data model that you are creating (conceptual, logical or physical), your data model will contain more or less detail. Generally speaking, a data model contains information about the data to be stored.

1. At its most basic level, that data is organized into entities which are stored in database tables. The basic element of any data model is a table.
2. Within a table, we store rows of data. Each row may contain many columns, which are the attributes of the item that we are storing in the table.
3. Each column (or attribute) will store a particular type of data, whether that’s a string of characters, a number with or without decimal digits, a Boolean (true or false) value, etc.
4. Some columns may be required (mandatory); some may be optional. Some columns have constraints on the types of values they can store.
5. And tables are generally linked together through relationships. How is the information related?

Here is a simple, not-yet-complete [**LOGICAL DATA MODEL**](https://my.vertabelo.com/doc/N5LBNnGbCXqmrUuKNc1v8ouPbOK04ah1) for data on customers, suppliers, invoices, payments, and accounts.

I used Vertabelo to create the above model. With a database design tool, I don't need to worry too much about the exact notation. I focus on the design while the modeling tool implements the [**ENTITY-RELATIONSHIP DIAGRAM NOTATIONS AND SYMBOLS**](https://vertabelo.com/blog/symbol-in-erd-diagram).

EDIT MODEL IN YOUR BROWSER

## Reuse Your Data Models

Different types of data models are often created during the database development process. Typically, we work with [**3 LEVELS OF MODELING**](https://vertabelo.com/blog/conceptual-logical-physical-data-model):

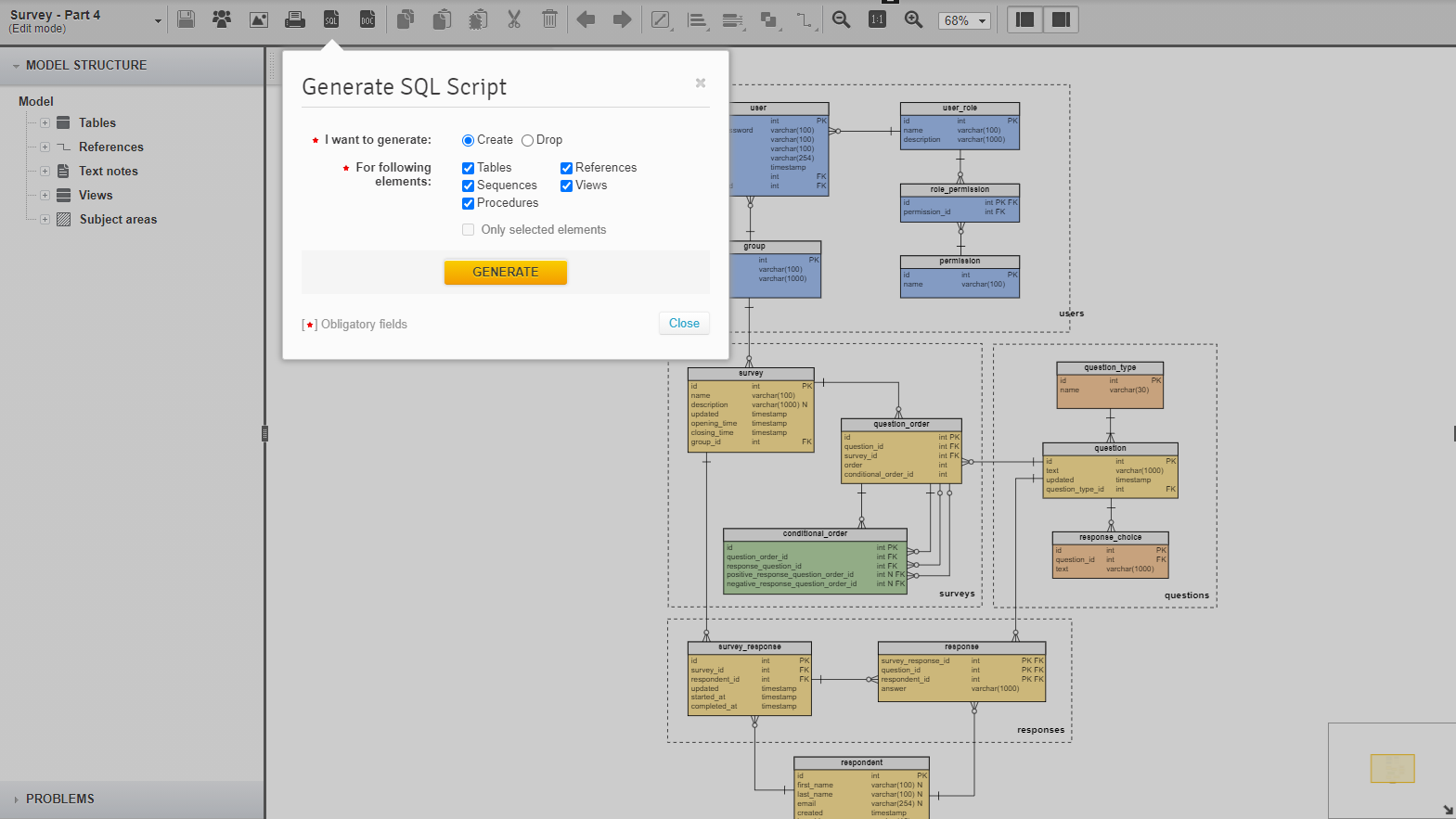
1. Conceptual: As its name implies, this high-level model generally has the least amount of detail. It answers the question: What are the main concepts involved in the data model?
2. Logical: The [logical data model](https://vertabelo.com/blog/logical-diagrams/)is where we get into the details. We plan what we are storing, but we don’t go into the details of how we’ll store it. This model answers the question: What are all the data elements that we need to manage?
3. Physical: The physical data model is basically a conversion of the logical data model into a detailed implementation for a specific database management system (DBMS). We need to follow the rules imposed by the particular DBMS and the particular version with which we will be working.

If you've heard that logical data modeling is dead or no longer required in the Agile software development process, you might find [**THIS ARTICLE**](https://vertabelo.com/blog/what-a-concept-is-logical-data-modeling-obsolete/)interesting.

A good data modeling tool will help us re-use our modeling work to [**GO FROM ONE LEVEL TO THE NEXT**](https://vertabelo.com/blog/logical-physical-data-model/) (e.g. from logical to physical). In this way, you’re not throwing away the work you do at the conceptual level. If you want to learn more, I recommend these articles on [**EXPANDING A CONCEPTUAL DATA MODEL INTO A LOGICAL DATA MODEL**](https://vertabelo.com/blog/conceptual-logical-physical-data-model/) and [**GENERATING A PHYSICAL DATA MODEL FROM YOUR LOGICAL DATA MODEL**](https://vertabelo.com/blog/conceptual-logical-physical-data-model/).

Of course, a physical data model is not the finished solution: you need to actually create the physical database. Good data modeling tools provide you with the means to [**GENERATE THE SQL (OR DDL: DATA DEFINITION LANGUAGE) SCRIPTS FROM YOUR PHYSICAL MODEL**](https://vertabelo.com/blog/create-physical-diagrams-in-vertabelo). This gives us the technical definition of the database, which is used to automatically generate the necessary database with which the application will interact.

### Working Together on Data Modeling

In addition, a good database modeling tool allows [**THE MODEL TO BE CREATED COLLABORATIVELY**](https://www.vertabelo.com/blog/remote-collaboration-on-database-design-just-got-easier/). It is important to remember that [**DATABASE DESIGN IS MORE THAN JUST AN ERD**](https://vertabelo.com/blog/database-design-more-than-just-an-erd). Database design needs to go beyond the ERD; it is part of the software development lifecycle. We collaborate with business users to create the initial logical design; we also collaborate with the development team to update the logical and physical data models during the software development lifecycle.

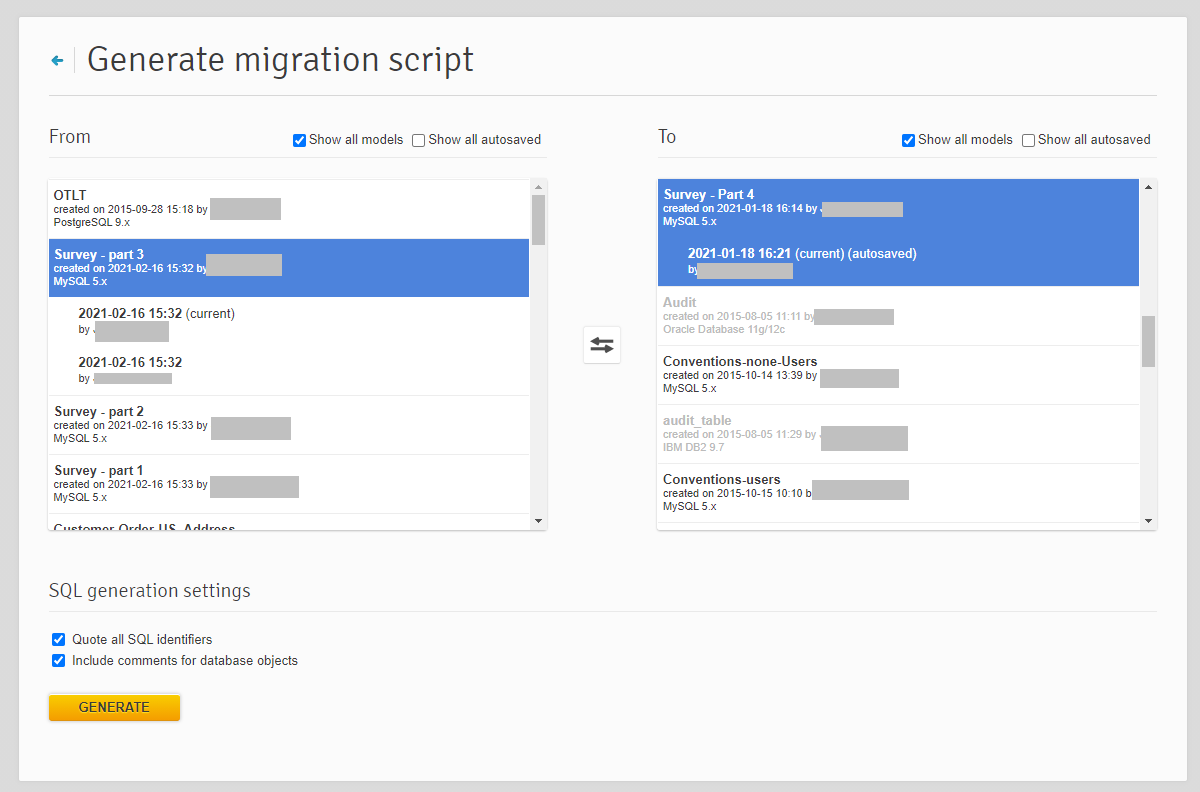
If you are interested in using collaborative database design, follow this advice to [**GET STARTED USING VERTABELO**](https://vertabelo.com/blog/start-online-data-modeling/).

## Other Database Modeling Challenges

If you’re modeling collaboratively, it can happen that someone accidentally makes a change that negatively impacts the model. Obviously, they can undo their changes – but if the changes were made a few days ago, that may no longer be possible. Vertabelo includes a [**VERSION MANAGEMENT SYSTEM**](https://vertabelo.com/documentation/general-knowledge/version-control/)that includes manual and automatic saving modes and version tagging. If your database is actually used with a particular version of the associated software, you can track that tagged version and compare any future changes.

Another challenge that I have often faced is working with an existing database for which I have no data model. Vertabelo includes tools to [**REVERSE ENGINEER** **FROM AN SQL DDL SCRIPT INTO A DATA MODEL**](https://vertabelo.com/blog/suitecrm-and-the-database-model-behind-it-2-how-to-organize-a-large-database-diagram/). Now, I doubt that you would reverse engineer from the SQL DDL scripts all the way back up to a conceptual data model, but you theoretically could. (Start from the scripts to the physical data model. Then, from the physical data model work back to the logical data model, and finally simplify your logical data model to a conceptual data model that accurately reflects the concepts of the business domain modeled in the existing database. This bottom-up reverse engineering can be useful when you are creating a new data model but you have an existing database that’s similar to your requirements.)

Another common challenge is migrating from a currently active database version to a new database model. Vertabelo can auto-magically [**GENERATE MIGRATION SCRIPTS**](https://vertabelo.com/documentation/physical-model/migration-scripts/) between versions of your data model. Manually writing and testing (and re-testing) migration scripts can be a real nightmare. Imagine if you could count on a tool to do the hard work for you! Vertabelo easily generates migration scripts between two versions of a database.



I must admit this is a dream come true! The tool manages versions and determines what the impact is of the changes made between different versions.

## Data Modeling Best Practices

There are numerous best practices to be considered in data modeling. I will not even try to list them all here, but I will point you to other sources, such as [**THIS ARTICLE**](https://www.vertabelo.com/blog/13-blog-articles-with-database-design-tips-and-best-practices/).  For now, here are a few that I think are essential:

* Think outside the database.
* Consider the nature of your application.
* Break your model into logical pieces.
* Document your model.
* Use naming, notation, and design conventions.
* Test early and test often.

## Getting Started with Database Modeling

A database model captures the information to be stored in a database. There are several steps in the [**DATABASE DESIGN**](https://www.vertabelo.com/blog/steps-in-database-design/) process, and (as you might expect) some [**DATABASE DESIGNS ARE BETTER THAN OTHERS**](https://vertabelo.com/blog/9-tips-for-better-database-design/).

# What Are the Steps in Database Design?

[Kovid Rathee](https://vertabelo.com/authors/kovid-rathee/)

[Kovid is a Data & Infrastructure Engineer working as a Senior Consultant at Servian in Melbourne, Australia. He is originally from Rohtak, India. In a career spanning 10 years, he has worked for startups and corporates, product companies and service companies alike. He started his career as an Assistant Professor at an engineering college and later moved to a full stack developer role for a while, before finally moving into the data space where he has been working for the past 8 years. He likes to write about data engineering, infrastructure-as-code, DevOps and SRE. He is a semi-trained Indian classical singer, writes poetry in Hindi and Urdu, and hopes to get published someday.](https://vertabelo.com/authors/kovid-rathee/)

Tags:

* [database design](https://vertabelo.com/tags/database-design)
* [Vertabelo Features](https://vertabelo.com/tags/vertabelo-features)

Database design is one of the most important factors contributing to the performance of an application. Consequently, how well the database is designed is of utmost importance. Database design is all about efficiently organizing data based on product workflows, future roadmap, and expected usage patterns.

The output of a database design exercise is a data model. A data model represents all the objects, entities, attributes, relationships, and constraints in the system. Broadly speaking, data models can be of two types: logical or physical. The representation of the data model is done by creating an ER diagram, also known as an entity relationship diagram, an ERD diagram, or a database diagram.

The physical data model relates to the actual implementation details in the database. The logical data model, on the other hand, abstracts away the implementation technicalities. This makes the logical data model consumable for the business. One key difference between the two models is that the logical model is database-agnostic while the physical model has to be specific to the database in use.

Proper database design is often understated and neglected during application development. The cost of this neglect is realized usually much later when new application features come in or when old features require change. This is when the database design ceases to make sense. While it is not possible to future-proof the design of a database, it is very much possible to make the effort to best understand the business use cases and design the database accordingly. Read more about tips on better database design [**HERE**](https://vertabelo.com/blog/9-tips-for-better-database-design/).

With that in mind, let’s go through the steps in database design.

## Step 1: Gather Business Requirements

The first step is to talk to the business about their requirements. If the conversation is effective, it should result in enough information to start working on a conceptual data model, which is an abstraction of the logical model. Talking to the business, first of all, provides a complete picture of the business processes, which, in turn, provides information about the various data points that are important for the business to capture and track. For instance, in a taxicab booking model, it is worth asking the following questions:

* Does the business want the vehicle tracking data in the database irrespective of whether there's an active trip or not? If yes, then the field **vehicle\_trip\_id** in the table vehicle\_trips would be nullable. Otherwise, it won't be nullable.
* Does the business want the history of changes to **trip\_status** stored in the database? If yes, then every time the **trip\_status** changes, there'll be another record in the trips table. Otherwise, every time the **trip\_status** changes, **trip\_status** will be updated in place.

As shown in this example, based on inputs from the business, you would end up choosing one option over the other. It would result in changing the concerned entities and their relationships.

Requirement gathering also generally involves initiating a conversation about data security, such as which data to be masked and encrypted. The requirement gathering exercise results in a requirement document often supported by a working draft of the conceptual data model.

## Step 2: Understand Business Roadmap

Businesses change their processes all the time; their ability to adapt makes them less likely to fail. Changing business processes means changing workflows and data models. Although it is not possible to know these changes way ahead of time, it is certainly possible to be up to date with the business roadmap.

For instance, if a company has plans to target a new geographical region, the model would have to cater to language support, currencies, time zones, and so on. The benefits of understanding the long-term business roadmap often show up in a smoother transition to new business processes.

Let me share one more example, which is more about continuously evolving business priorities. The taxi business was impacted badly at the beginning of COVID-19. As a cab company, you want to act preemptively to assure people that you're doing everything to make sure that your travel in the cab is as safe as possible, that the vehicle is disinfected every day, that the driver wears a mask at all times, and that there's hand sanitizer available in the cab. Now, to capture all this information, changes to two entities, drivers and vehicles, would be required. Several Boolean flag fields need to be added to these entities to cater to this business use case.

## Step 3: Identify Entities and Attributes

Once the business requirements are gathered, the information can be used to identify entities along with the essential set of attributes. One or more entities generally map directly to business processes, and the relationship between those entities also mimics the business process workflow.

This step is also used to identify which attributes will act as identifiers in the entities. Identifiers translate to primary keys in the physical model. In addition, it is also common to specify data types for all the attributes in this step.

For instance, in the taxicab booking model, you would have to identify the attributes which will act as the mandatory fields for the registration of users and drivers from the booking app. User registration would be done using **user\_phone** and driver registration would be done using **driver\_phone**.

Similarly, other entities and attributes are identified during this step, after having been mapped to the business process workflows.

## Step 4: Identify Relationships

After identifying the entities and their attributes, the next step is to define the relationships between entities based on the business workflows that were documented in the requirement gathering phase. In addition to establishing that there's a relationship between two entities, it is also important to identify which of the following four types of relationship exists between them. Consider two arbitrary entities, A and B:

1. One-to-one → One record in A corresponds to at most one record in B.
2. One-to-many → One record in A corresponds to many records in B.
3. Many-to-one → Many records in A correspond to at most one record in B.
4. Many-to-many → Many records in A correspond to many records in B.

In the taxicab booking model, only one type of relationship has been used, i.e., one-to-many. Take the relationship between users and trips as an example. In the model, there's an assumption that only one user can be related to a trip, which implies that there are no shared or pooled cabs. But if there were shared or pooled cabs, there would possibly have been a many-to-many relationship between users and trips, if many users shared the same **trip\_id**.

## Step 5: Create a Logical ER Diagram

With entities, attributes, and entity relationships defined, the immediate next step is to draw the ER diagram. All of the steps listed above can be done within [**VERTABELO**](https://vertabelo.com/). There are no hard and fast rules for the way logical modeling is supposed to be done, with the possible exception of the reference notation.

For instance, take a look at the [**FOLLOWING EXAMPLE**](https://my.vertabelo.com/logical/public-model-view/Wu2PfjXa9uups6pCHSYMRvFOuqw7bDQVf11OuHOigkp2OmcZF82EbzNdy1cNdD4u?x=4986&y=5345&zoom=0.7943) of a logical ER diagram. It captures a simple business workflow of a cab company, where a user can book a ride with the ability to track the vehicle.

## Step 6: Validate the Logical ER Diagram

EDIT MODEL IN YOUR BROWSER

Logical modeling is a process in which a lot of business information needs to be translated into a database design. Without thorough checks, this phase of database development is prone to errors that can prove to be quite costly at a later stage.

To take care of this, [**VERTABELO**](https://vertabelo.com/) has a thorough list of checks that can be performed on a logical model. Checks can be performed at all granularities, from the model as a whole to individual attributes, and everything in between. Some of the simple checks are:

* Names of entities, attributes, relationships, etc., cannot be null and have to be unique.
* An entity must have at least 1 attribute.
* Identifiers (PKs) must be defined for every entity.
* The model must use one of the listed data types for attributes.

All of these checks are optional and can be configured to be skipped, if there's another validation framework in place. Proper validation from [**VERTABELO**](https://vertabelo.com/) helps you move to the next step with the minimum amount of friction possible.

## Step 7: Create a Physical ER diagram

Once the logical ER diagram is created, the next step is to create a physical data model. The physical data model will be specific to the database where you want to deploy the data model. All databases have their unique implementation of nomenclature rules, data types, and constraints. Due to this, the Data Definition Language (DDL) often differs from one database to another.

To create a physical data model, follow these steps:

1. Find the closest data type in the target database to replace the generic data type selected in the logical data model.
2. Follow the nomenclature rules for tables, columns, and constraints as prescribed by the target database.
3. Modify the model to align with predefined query workflows. This generally results in increasing redundancy to save joins.
4. Finally, you can create indexes, partitions, distribution keys, and sort keys. This is when you can create any performance-enhancing modifications to the model.

These steps can be performed using any data modeling tool you can use to create a data model from scratch. However, [**VERTABELO**](https://vertabelo.com/) has an option to convert a logical data model to a full-fledged physical data model for all the major database systems like MySQL, PostgreSQL, Oracle, Microsoft SQL Server, Amazon Redshift, Google BigQuery, and more. Once the logical data model is converted to a physical data model, you can carry on with the four steps we discussed.

[**VERTABELO**](https://vertabelo.com/) also has an option to add pre- and post-deployment scripts at the table level along with any comments at the very granular level of the model. The comments turn out handy when the documentation generation feature offered by [**VERTABELO**](https://vertabelo.com/) is used. The database document can be exported in any of the following three formats: HTML, PDF, or DOCX.

Continuing with the cab booking example, let's take a look at the physical data model generated by [**VERTABELO**](https://vertabelo.com/).

## Step 8: Validate the Physical ER Diagram

Just like the logical ER diagram was validated, [**VERTABELO**](https://vertabelo.com/) has a tool to validate physical ER diagrams with several additional checks, like whether or not FKs exist and whether the length of a table name or a column name exceeds the limit based on the database selected.

The validation doesn't need to be run explicitly. It happens as the diagram is modified. The issues with the model fall in one of three categories: errors, warnings, and hints, in order of decreasing severity. There's a useful, well-written [**ARTICLE**](https://vertabelo.com/blog/the-9-most-common-database-design-errors/) which talks about the common mistakes made during the database design process.

## Step 9: Fix Issues With the Physical ER Diagram

The results of the validation may identify issues which need to be fixed. Some of the most common issues are:

* Missing foreign keys where entity relationships have been defined.
* Missing primary keys from tables.
* Unsupported data types for the selected database.

Once these and other similar issues are resolved, the model is ready to be exported to a deployable SQL script for the selected database management system.

## Step 10: Generate the DDL Scripts for Deploying the Model

[**DATABASE DESIGN IS NOT JUST ABOUT CREATING ER DIAGRAMS.**](https://vertabelo.com/blog/database-design-more-than-just-an-erd/) A data modeling exercise using ER diagrams is successful only if it results in something deployable. [**VERTABELO**](https://vertabelo.com/) has a convenient option to export the physical model to a ready-to-deploy SQL script. Once it is generated, any pending issues can be resolved directly in the SQL script.

However, changing the generated SQL script is not recommended. It causes a drift between the physical data model and the SQL script deployed on the database, which can also mean a drift between the actual tables and the database documentation.

Now that we've reached the end of the database design process, let's have a look at the [**SQL CODE**](https://my.vertabelo.com/doc/Dp6gjLh3PHPbndoWoKxvI5ftFvvqDJ1V) generated by [**VERTABELO**](https://vertabelo.com/).

## Share Your Thoughts

Database design is a high-impact activity in software development. The field of database design has evolved over the years with new ways to represent the design for the business, for the engineers, and for the data analysts. This has often resulted in new types of diagrams, modeling standards and notations. Much of the evolution has been covered in the [**DESIGN FUNDAMENTALS**](https://www.vertabelo.com/blog/design_fundamentals/) section.

We'll be happy to see what your experiences have been in designing databases. Write to us at contact@vertabelo.com.

# A Detailed Guide to Database Schema Design

[Gustavo du Mortier](https://vertabelo.com/authors/gustavo-du-mortier/)

[Gustavo du Mortier is a functional and data analyst at MasterSoft, an Argentinean software company specializing in ERP and healthcare solutions. He’s written many books and articles on different aspects of programming and databases. In his spare time, he plays guitar and helps his two sons build and enhance their gaming computers.](https://vertabelo.com/authors/gustavo-du-mortier/)

Tags:

* [database design](https://vertabelo.com/tags/database-design)

When you don't have a guide, certain tasks seem more difficult than they really are. In this article, I offer you a complete database design guide so you won’t get lost when trying to build a robust and effective database.

A [**DATABASE SCHEMA IS BASICALLY AN ABSTRACT CONCEPT**](https://vertabelo.com/blog/what-is-database-schema/). Even so, it’s better to think of it as a practical tool that helps you create structures to turn large amounts of data into something useful. In the process of designing a database, creating the diagrams and transforming them into a functional database is only the final step; the majority of the work starts much earlier.

The work of an SQL data modeler begins with a planning phase. In that phase, object discovery will shape the whole design. It will be the basis for all the necessary details to turn the design into a functional database.

The input elements for the planning stage are the results of the requirements engineering process. These results commonly take the form of narrative descriptions of how a system should behave. As the system analysis proceeds, other design artifacts – such as user stories, use cases, sequence diagrams, etc. – may emerge. Thus, our first step in database design is to find (in all this documentation) the objects for which information needs to be stored.

Wondering about all[**THE STEPS IN DATABASE DESIGN?**](https://www.vertabelo.com/blog/steps-in-database-design/) Read this article on the [**5 STEPS FOR AN EFFECTIVE DATABASE MODEL**](https://vertabelo.com/blog/5-steps-for-an-effective-database-model/) to get an overview of the whole process.

## The Database Design Guide, Part 1: Planning

The most convenient and common way to create a data model is to go from general to particular while following the steps of database design. Following this path, you'll come across three different types of models: [**CONCEPTUAL, LOGICAL, AND PHYSICAL DATA MODELS**](https://vertabelo.com/blog/conceptual-logical-physical-data-model/).

Using the elements identified during the planning stage, you can outline a conceptual data model and use it to obtain initial validation from users and stakeholders. In the conceptual model, we normally include the main data entities and the main relationships between them. This is so that the usefulness of the data model can be broadly understood. You can think of entities, attributes, and relations as the necessary ingredients to [**PREPARE A DATABASE MODEL**](https://vertabelo.com/blog/how-to-prepare-database-model/).

According to our database design guide, once the conceptual model has been validated, we can expand the level of detail of the diagram and build the logical model with our data modeling tool. This model will show all the entities with all their attributes, and all the existing relationships between them. We also distinguish the different types of relationships (1-to-1, 1-to-many, many-to-many, inheritance or generalization/specialization) and the generic data type (i.e. integer, decimal, date, time, varchar, etc.) of each attribute.

The logical model should guide the development and evolution of the schema during the lifecycle of the software solution of which it is a part. Its independence from any particular RDBMS keeps it valid even when the data repository changes between different environments or technologies. That is why it is highly recommended to do all the design work on the logical schema using a database diagram tool and convert it into a physical schema only when it is necessary to “give life” to it. Then, you’ll be able to populate it with data and attach it to a software solution.

In the physical diagram, the schema structure is defined according to the peculiarities of a given relational database management system (RDBMS). This means that the data types of a table’s columns are no longer generic, but specific to the target RDBMS. In addition, relationships between entities become constraints between columns of the tables involved and inheritance or generalization/specialization relationships become sets of tables and constraints between fields in them.

Let’s check out some [**TIPS FOR BETTER DATABASE DESIGN**](https://vertabelo.com/blog/9-tips-for-better-database-design/) and see how this way of working is brought to reality with a practical example.

## The Database Design Guide, Part 2: Design the Model

Suppose you need to create a database schema for a theatre ticket booking app. Since the app doesn’t exist yet, you must initially rely on a narrative description of its behavior. For example:

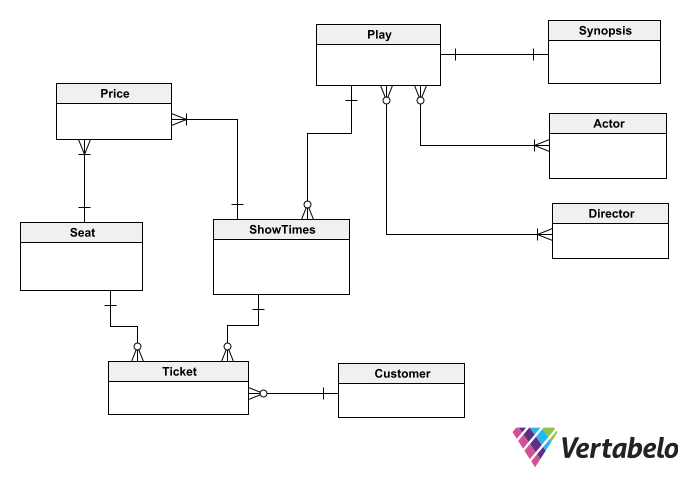
Customers select the play for which they wish to book a ticket. The app displays the show times for the selected play along with its synopsis, cast, director(s), ticket prices, and any other relevant information. Customers choose a date and time from the available show times. The app shows the availability of seats for the chosen date and time. Customers select the seats to book. The app shows the total price of the booked seats and refers the customer to the payment and check-out module.

### 1. Identify Entities

To build a schema design that fits the described system, you must pay attention to the objects mentioned in the previous description. A first reading allows you to identify the following objects:

* Customer
* Play
* Actor
* Director
* Synopsis
* Ticket
* Show Time
* Seat
* Price

With some knowledge of the app’s logic and the list of objects above, you can use a database diagram tool to outline a conceptual model. Remember, this model shows the main entities and the main relationships between them:



The conceptual model shows the main entities of our schema and the main relations between them.

This conceptual diagram will be used to discuss the model with customers and stakeholders to make sure it is correct. Having validated the conceptual model, you can convert it into a logical diagram. To do so, you must add attributes (and their respective data types) to each entity.

It is important not to expect your model to be perfect at this stage. It is very likely that you will have to make many changes to your logical diagram before you can convert it into a physical diagram – and, finally, into a database schema.

Modern development teams usually employ Agile development methodologies; these put adapting to change as a top priority. In database schema design, the principle of agility and adaptability to change goes hand-in-hand with the use of an intelligent database design tool like [**VERTABELO**](https://vertabelo.com/). Such tools give a lot of flexibility to the design process. This includes the ease of modifying a diagram as well as applying those modifications to a live database without the risk of losing data or breaking productive applications or processes.

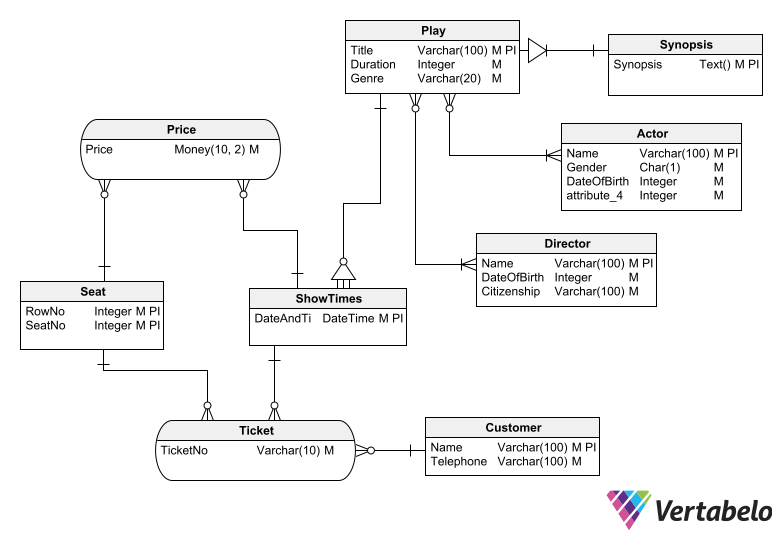
### 2. Adding Attributes to Entities

To add attributes to entities, go through the objects in the conceptual diagram and think about what information you need for each of them. You can leverage the narrative descriptions of the system, your own knowledge of the app domain, and information provided by users and stakeholders.

In the ticket booking system, you can list the following objects to start outlining entities, attributes, and relations:

* Customer: Name, Telephone number.
* Play: Title, Duration, Genre, Synopsis, Director/s, Actors, Show Times, Prices.
* Ticket: Customer, Play, Show Time, Seats

The next step is to enter the attributes of each entity and assign a data type to each one. It is important to use a tool that provides enough flexibility and lets you easily move objects around, link/unlink them, add and remove attributes, etc. The right schema design might begin to take shape in your head as you draw it out. (At least, that’s the way it works for me.)



Don’t worry about your design being perfect on the first try. It’s better to have a tool that makes it easy to make as many changes as needed.

Remember that each entity must have a unique identifier – one or more attributes that will identify each row of the table univocally when the schema becomes a working database.

When defining a unique identifier for each entity, ask yourself if it could be useful to use a surrogate (or artificial) key to function as a unique identifier. To make this decision, consider if the identifying attributes are prone to frequent changes, if they are defined as complex data types, and so on.

### 3. Add Relationships Between Entities

Once all the entities with their attributes have been drawn, you can establish the relationships between them. These relationships should reflect the relationships that exist between the objects in the real world. Some entities may become attributes of other entities during this process, i.e. when you find out that a 1-to-1 relation doesn’t make sense between two tables. To clarify this concept, you can read about [**THE DIFFERENCE BETWEEN ENTITIES AND ATTRIBUTES IN A DATA MODEL**](https://vertabelo.com/blog/entities-attributes-data-model/).

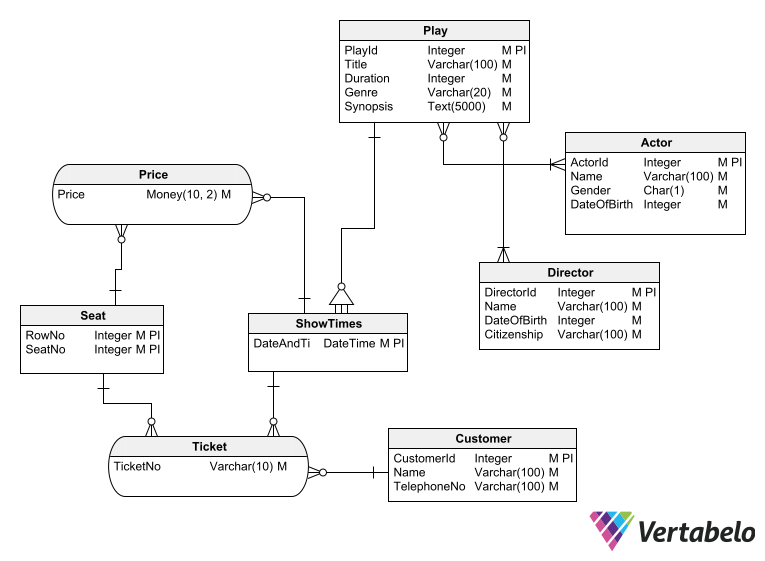
#### Normalization

Depending on the use you want to make of the data, the previous diagram can be improved. These improvements are achieved by normalizing the schema – that is, by modifying it according to the normal forms established by the relational model.

Normalization will save you from inconsistencies in the data, such as having a reservation for a play on a date or time when that play is not shown or having two reservations for the same seat at the same time.

In addition, applying normalization techniques can make your schema more versatile for querying – e.g. if you want your database to be able to inform a user of all the plays in which a certain actor is working, all the plays directed by a certain director, or all the plays that will be shown within a range of dates.

Once corrected, your schema might look like this:



After a couple of design iterations, you should arrive at a schema design that will efficiently solve any requirements of the app.

#### Model Validation

When you use a drawing tool to design your database schema, you can make mistakes that will make it difficult to implement that design on a database management system (DBMS) later on. You might unknowingly assign the same name to two tables, create a table without a primary key, or have a foreign key relationship between two columns of different data types.

If you use Vertabelo to design your database schema, the risk of missing errors like these is reduced to zero. This is because Vertabelo validates your model as you work on it; if it finds any flaws, it shows them to you in a panel with a warning icon so that they don’t go unnoticed.

### 4. Preparing the Physical Schema

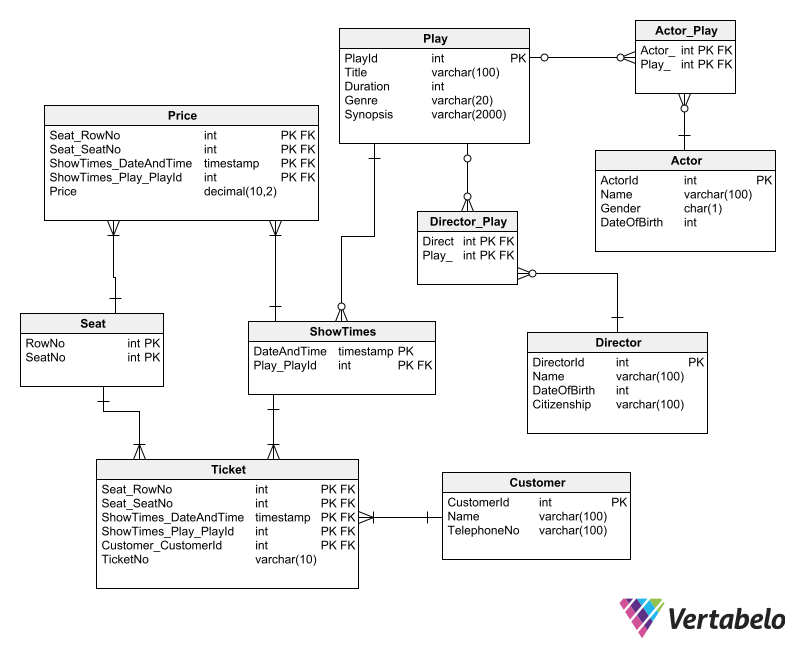
The step before implementing the schema design on a DBMS to get it up and running is to create a physical diagram. Using Vertabelo, you can derive the physical diagram from the logical schema automatically. In fact, the process of getting the physical diagram from a logical one takes only a few seconds. It is just a matter of choosing a logical diagram and giving Vertabelo the command to convert it into a physical diagram.

The conversion from logical to physical requires you to choose a target database engine and verify that there are no errors in the diagram (using the same validation tool). Vertabelo will then create a physical schema. To do this, entities and associations in the logical schema are converted into tables in the physical schema, and relationships between entities in the logical schema are converted into foreign key constraints in the physical schema. For more information, read this article on [**HOW TO GENERATE A PHYSICAL DIAGRAM FROM A LOGICAL ONE IN VERTABELO**](https://vertabelo.com/blog/logical-physical-data-model/).

In a physical schema, all foreign key constraints should have an index that improves query performance. Vertabelo will issue a warning if you don’t. It’s also important to be clear about [**WHAT A DATABASE INDEX IS AND WHAT IT IS USED FOR**](https://vertabelo.com/blog/what-is-database-index/).

Other elements of the physical schema – such as views and various SQL scripts – can be added in Vertabelo before implementing the schema on a DBMS. This ensures that all the elements needed to create the database are included with the schema.

To create the database, simply ask Vertabelo to generate the necessary SQL scripts and then run them on the chosen DBMS.



In Vertabelo, a physical schema is much more than meets the eye. It includes indexes and SQL statements that, once you convert it into a database, will require practically no tweaking under the hood.

## Best Practices for Database Schema Design

The following set of best practices will help your database schema retain its usefulness throughout the whole lifecycle of the application it supports. To better organize your database diagrams, you may want to read about [**VERTABELO’S SHORTCUT TABLES AND HOW TO USE THEM EFFECTIVELY**](https://vertabelo.com/blog/vertabelo-feature-shortcut-tables/) and [**HOW TO ADD REFERENCES TO ENTITY-RELATIONSHIP DIAGRAMS IN VERTABELO**](https://vertabelo.com/blog/adding-references-to-er-diagrams/).

### Naming Conventions

It may seem trivial, but proper naming conventions are extremely important for an effective database schema. It’s less important which naming convention you use; what is critical is to choose one and stick to it at all times. Keep in mind that you will not be the only one creating objects in your database schema. It is vital to specify the naming convention used in your database schema and make it known to everyone who works with the database.

It is also critical to clearly establish the naming convention used in a database schema before you start populating it with objects – and especially before implementing the schema in a DBMS. Once the database is operational, any name change can be catastrophic.

Regarding object names, avoid using reserved words in table names, column names, indexes, etc. at all costs. Also avoid using special characters, spaces, inverted commas, hyphens, or language-specific letters. The use of such characters forces the use of delimiters when referring to those tables or columns in SQL statements, which adds yet another cumbersome task for the database programmer.

An important part of defining a naming convention is whether to use plural or singular for table names. Some argue that a table representing an entity should have a singular name – e.g. **Customer** instead of **Customers**. But this is just a matter of preference; what is really important is to adopt a criterion and always stick to it.

The use of prefixes or suffixes in table and column names is discouraged. It may seem useful to add (for example) a prefix denoting the data type of a column. But over time, the uselessness of this rule becomes apparent and everyone stops using it. This creates confusion due to the coexistence of names with and without prefixes.

The use of unnecessary descriptors in names is also discouraged. There is no need, for example, for a customer table to be called **CustomerTable** or **CustomerList**. It is sufficient if it is called **Customer** or **Customers** (depending on the plural/singular convention adopted).

### Security Policies

As designers, we may fall into the error of believing that our databases will always be well protected from any risk of data loss or theft. However, a good security practice is based on always considering the possibility of a malicious user managing to circumvent all defense mechanisms and gaining free access to the data.

Database designers must apply mechanisms that serve as the last line of defense against unauthorized access to the information. A popular one is the encryption of sensitive data, such as passwords, credit card numbers, or personally identifiable information (such as a personal ID number). In your schema design, columns with sensitive information should be able to store encrypted information. You may also want to employ sub-schemas for proper user authentication.

### Documentation

Once an application goes into production, its database schema can remain stable for a long time – and, for long periods of time, no one needs to view its schema. But sooner or later the need will arise to make some changes to the application or the data model. Then it will be necessary to revisit the design diagram to modify it and work on the changes to the data model. But the people who need to do this work may not be the same people who designed the database. Or even if they are the same people, they may not remember why they made certain design decisions.

In these situations, the documentation of a database schema becomes essential. The simplest and most effective way to document a data model is to accompany diagrams with annotations and explanatory text; they can guide the person who has to analyze it long after it has been created.

[**VERTABELO ALLOWS YOU TO ADD TEXT NOTES**](https://vertabelo.com/blog/text-notes-in-vertabelo/) to your diagrams in sticky-note style, which is a valuable aid to designers who need to understand why you related two tables or why you assigned a certain data type to a column.

## You Can Design a Robust Database Schema!

If you carefully follow this guide to good database design, you will substantially improve the quality of your database schema. As a result, you will also contribute to improving the quality of the software that uses your schemas. The bottom line is that your database designs will be robust and durable, and that they will make life easier for everyone on the development team who has to work with them.

5 Steps for an Effective Database Model

Jeffrey Edison

Over the past 25 years, Jeffrey has worked in various roles in financial services software, with a focus on transaction banking and automation with artificial intelligence methods. He has extensive knowledge and expertise in financial payments, combating financial crime, and virtual account management. His experience spans various disciplines ranging from architecture and design to product management and sales.

Tags:

database modeling

database

design

modeling

tip

Database design is the process of producing a detailed model of a database. This model contains the necessary logical (table names, column names) and physical (column datatypes, foreign keys) choices to translate the design into a data definition language (aka SQL), which can be used to create the actual physical database.

When I need to create the design for a new database, in other words, the data layer for an application, I follow a few mental steps that I think can help others when they need to go through the same process. And, to be honest, for me, I progress through the first steps mentally without actually working on the technical details – and sometimes at a more subconscious level.

For me, the first step is to get a high-level grasp of the topic and an understanding of the business or functional area. Generally this is referred to as the business domain.

1. Domain

What is the domain that this solution needs to address? What are the issues in this domain? What are the types of information that need to be held in the database? Take the example of a human resources database for a company: you would need to model employees, their marital status, employment status, salary, holiday periods, etc.

2. Functionality

What is the functionality that is required? In other words, what are the Use Cases related to this data? What types of functionality do you need to support: creating and maintaining (update, delete, edit) items, reporting and analysis, etc? Hopefully, the functional requirements of the application have already been defined, but that is not always the case. Make a real effort to have a high-level understanding of how the data will be used.

Now this gets interesting: what functionality is allowed for an employee? Should all basic CRUD (Create, Retrieve, Update, Delete) functionality be allowed – creating new employees, editing employees when their situation or employment status changes (s/he gets married or divorced, resigns, is fired, etc)? Most likely you will allow only Create-Retrieve-Update functionality since employee records may need to be kept for a very long period (e.g. 10 years) and should not be immediately deleted.

3. Entities

Now you should have a concept in your head of what you need to create and you know the types of interactions that are necessary with the data (and therefore with the database).

To actually build the database, you need to start working with the database entities: modelling the main entities of the system.

The good thing about thinking about the domain and the functionality is that you probably have actually defined what the main entities in the database are likely to be. You know what the contents of the database are and how the content will be used.

When considering the domain, we already mentioned most of the entities for a human resources database: employees’ marital status, employment status and salary. Can marital status and salary simply be columns on the employees table or is it necessary to keep a history of what an employee’s salary was in the past? Usually, you need to keep the employment history so we should add tables for status history, salary history, and probably also marital history.

4. Relationships

The next level is to understand how the entities are related. What entities are linked to what other entities (e.g. users to the items that they have created)?

Should these relationships be well-defined or casual in the database (foreign keys or loose relations with the related ids stored, but not actually defined as a foreign key in the physical model)?

We said that several columns of the employee table will have a well-defined value, such as their status: single, married, divorced. Here is a perfect example where we might link a column to a table of appropriate values via a foreign key so that the database itself ensures the integrity of the data. It is also possible to rely on the application that is creating rows in the database, but why not use the power of a database’s foreign keys to ensure data integrity? That way, you can avoid having the application introduce errors into the data.

5. Formal Design

Now that you know the entities and relationships, you are ready to build a model or an Entity Relationship Diagram (ERD) of the database, and that should not take too long as you know what you want to create. This is where tools come in handy.

Let us consider VERTABELO for creating the formal design.

First, create a model for the database and start adding in the entities that you thought of previously. VERTABELO will remind you that you need to define primary keys for each table; I recommend using id fields as that will give you more potential flexibility for the future. Next, add in the relationships that you considered previously.

I typically add timestamps with the date/time of the creation of each row, so that the information can be displayed in the application (for example “Created 24 December 2014”).

In the business area that I work in, financial services, it is also very important to keep a record of the last user that modified a row and when the row was modified to have at least some traceability of changes. Of course, other business areas may not have this need for traceability. So we want a reference between “user last changed” to the table of users. However, we may want to allow a user to be deleted even if he or she was the last user that changed a row. If that is the case (that a user can be deleted), then we need to loosen that referential integrity constraint and remove the foreign key from the “user last changed” to the table of users.

After creating the basic model, you should be able to start thinking about improvements. What additional information might be stored in each entity? What additional details and attributes exist for each entity?

Conclusion

I have found these steps to be very effective in helping me create my database models.

What steps do you use to design yours?

# Common ER Diagram Mistakes

[Tihomir Babic](https://vertabelo.com/authors/tihomir-babic/)

[Database designer](https://vertabelo.com/authors/tihomir-babic/)

Tags:

* [data modeling](https://vertabelo.com/tags/data-modeling)
* [database design](https://vertabelo.com/tags/database-design)
* [tip](https://vertabelo.com/tags/tip)

Get to know the ER (Entity Relationship) diagram, its parts, and what often goes wrong when creating it.

Have you ever created a relational database model? Or maybe you're trying to create your first one? You know (or you'll soon find out) that translating real-world problems to database logic can sometimes be quite difficult.

One of the tools that might help you is the ER diagram. Common database design wisdom holds that the better your ER diagram, the easier it will be to build the database model. This important item sets the tone for all future frustrations or successes. With a good ER diagram, creating a relational database model is quite straightforward. Of course, mistakes can be made in any phase of database modeling. However, having a good ER diagram can help you avoid some of those mistakes.

So, let's see what the ER diagram is and how we can avoid its common mistakes.

### What Is an ER Diagram?

“ER Diagram”, or ERD, is short for Entity Relationship Diagram. It maps out the problem to be modeled, but in a structured way that shows the relationships between entities.

### An ER Diagram’s Building Blocks

ER diagrams consist of the following elements:

* Entity
* Relationships
* Entity or relationship attributes

The first element of the ER diagram is the **entity**. The entity is an object or occurrence that we want to store information about. Basically, it's anything on which we can collect data. For instance, we might store data on employees, students, teachers, buyers, products, departments, payments, locations, etc.

Once we have entities, it is necessary to create **relationships**. A relationship shows how one entity is connected to and associated with one or more other entities.

The final element of the ER diagram is an entity or relationship **attribute**. An attribute is a description of a property belonging to an entity or relationship. Attributes have values. Some attributes for the entities mentioned above could be:

* **Employee, student, teacher, buyer** – ID, name, surname, date of birth, address, etc.
* **Product** – ID, category, description, color, serial number, etc.
* **Department** – ID, department name, department head, number of employees, etc.
* **Payments** – ID, date and time, amount.
* **Location** – City, ZIP code, region, country, continent.

### Types of Relationships

Before we get into the usual mistakes found in ER diagrams, it is important to understand the possible relationship types. Most ERD mistakes are essentially erroneously-defined relationships between entities.

There are three types of relationships between entities:

* One-to-one (1:1)
* One-to-many (1: N)
* Many-to-many (M: N)

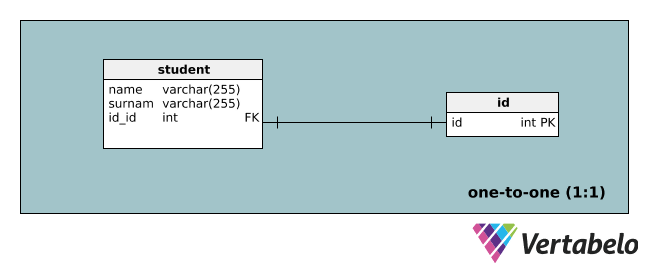
#### One-to-one (1:1) relationships

The first relationship type is **one-to-one**, or **1:1**. In this relationship, a single instance of one entity can be connected only with a single instance of another entity (and vice versa, of course).

Let's say that we have the entity student with the attributes name and surname. We also have the entity id with the sole attribute id. The 1:1 relationship would mean that one student can have only one ID number. It also means that one ID number can belong to only one student.

This relationship is very rarely seen in databases. If only one ID can be connected with only one student, there is no need to separate them into two different entities.

Here's an example of this relationship:

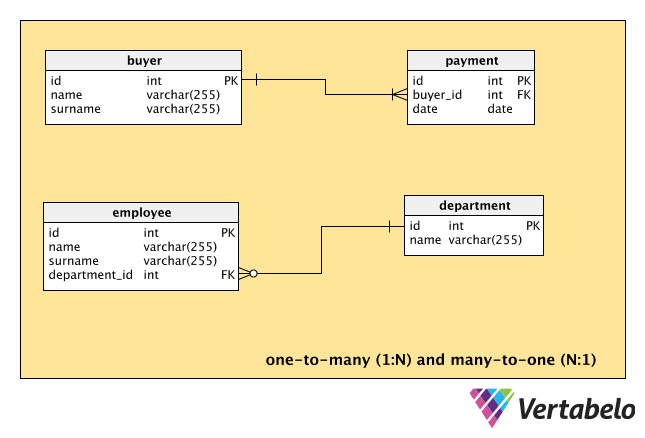


#### One-to-many (1: N) relationships

The most common type of database relationship is **one-to-many** or **1: N**. A One-to-many relationship means that each single instance of one entity can be connected with multiple instances of another entity. It also means that every instance of the second entity can be associated with only one instance of the first entity.

For example, there is an entity buyer with the attributes id, name, and surname. We want to establish a relationship with the entity payment that has the attributes id, date, and value. This is a 1: N relationship because one buyer can make one or many payments. However, one payment cannot be made by several buyers; it can be made only by one buyer.

Here's the example:



This relationship can also be seen the other way round. In this situation, it is called **many-to-one** or **N:1**. Of course, this is not a new type of relationship. It is the same as 1: N, but it is looked at from the opposite direction.

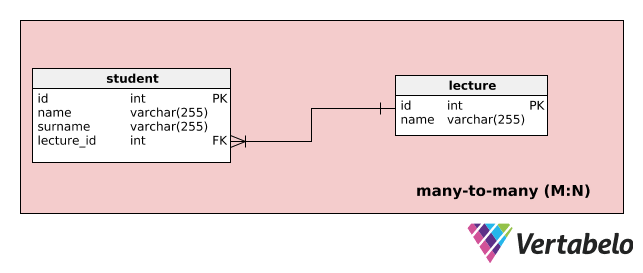
As an example, suppose we have the entity employee with the attributes id, name, and surname. We want to establish employee’s relationship with the entity department that has the attributes id and name. The relationship between these two entities is N:1. This means that every employee can work in only one department, but multiple employees can work in the same department.

#### Many-to-many (M: N) relationships

A **Many-to-many** or **M: N** relationship means that every instance of the first entity can be associated with more than one instance of the second entity. It also means that every instance of the second entity can be associated with multiple instances of the first entity.

Let's see how this works between the entities student and lecture. Let's say that student has the attributes id, name, and surname. The entity lecture has the attributes id and name. A many-to-many relationship can be interpreted in the following way: one student can attend one or more lectures, while one lecture can be attended by one or more students.

Here's the diagram for this example:



In database modeling, such relationships are usually split into two or more 1: N or N:1 relationships by introducing new entities.

### Typical Mistakes Made When Creating an ER Diagram

Many ER diagram mistakes fit into one of these four categories:

* Incorrect relationships between entities
* Using an entity instance instead of an entity
* Confusing an attribute with an entity
* Complex attributes

Let’s look at each one individually.

#### Incorrect relationships between entities

The most common mistakes occur when **defining the relationship between entities**. There are usually no mistakes in a 1:1 relationship. However, it is very easy to confuse a 1: N relationship with an M: N relationship. This usually stems from not understanding the requirements provided by the end user. It is vital to have very clearly defined requirements and a deep understanding of why the database is needed and how it will be used. If we create an ER diagram with insufficient data and incomplete understanding, it will most probably result in relationships between entities being wrongly defined.

Let's look at an example. If you're creating a database for a bank, you'll most probably create an ER diagram with the entity client having the attributes id, name, and surname. You'll also have an entity called account with the attributes id and type. If you lack experience in the banking industry, you'll probably think there is always a 1: N relationship between the client and account entities, as shown below.

One customer can have multiple accounts in one bank. However, an account can be owned by only one customer. Is this actually true? Maybe it is, maybe it isn't. Quite a lot of banks offer joint accounts that can be used by several clients. Are you creating an ER diagram for a bank that offers such a service? If the bank doesn't offer joint accounts, then you’re right: the relationship between client and account is 1: N. However, if the bank does offer joint accounts, then the relationship is M: N.

EDIT MODEL IN YOUR BROWSER

#### Using an entity instance instead of an entity

Another common mistake is using an entity instance instead of an entity. An entity instance is a single occurrence of a certain entity – i.e. an entity that could actually be an attribute of a larger category.

Let's say we work in a company that allocates mobile phones and laptops to certain employees. So, in our database, we’d have an entity called laptop with the attributes id and model and an entity called employee with the attributes id, name, and surname, right?

There’s a problem here: a laptop is actually not an entity – there are also mobile phones to account for. The solution is to replace the entity laptop with a more general entity, such as equipment. This entity could have the attributes ID, type, and model, as shown below. The type could consist of values such as phone, PC, tablet, and laptop. This way, there is no need to create a separate entity for every type of equipment.

You can find the example here:

### Confusing an attribute with an entity

EDIT MODEL IN YOUR BROWSER

The next common mistake is confusing an attribute with an entity. Let's say we have decided to create an entity called employee that will consist of the attributes id, name, surname, date\_birth, id\_department, name\_department, and head\_department. This entity will get us in trouble when we’re creating a database model because it **consists of too many attributes that don't uniquely define this particular entity**.

Remember, we defined entities as anything that we can collect data on. With that in mind, we can see that the current employee entity can be split into two entities: employee and department, as shown below.

#### Complex attributes

EDIT MODEL IN YOUR BROWSER

The last common mistake we’ll talk about is including a complex attribute in an entity. In other words, we have **an attribute that should actually be its own entity**.

Let's say we have an entity called students that’s defined by the following attributes: id, name, surname, date\_birth, address, and exam\_passed. Here, exam\_passed is a complex attribute that consists of more than one piece of information, i.e. the exam ID and date and the student’s score. Leaving it that way would be a mistake. Instead, we should **make a new entity out of this complex attribute**. The new entity could be named exams and consist of the following attributes: id, date, students\_id, and score.

You can find the example here:

### Did You Get Any Useful ER Diagram Tips?

EDIT MODEL IN YOUR BROWSER

These four types of mistakes are the most common ones in the ER diagram creation process. Of course, there's no complete list of typical mistakes or types of mistakes. In real life, many kinds of mistakes are likely to happen. A lack of planning, insufficient technical support, and a rushed database design process all contribute their own problems. If you’ve ever created databases or participated in it from the business side, you’ve probably experienced some of them! All those circumstances lead to various mistakes, some of which are quite unique.

Do you have your own example of a not-so-good ER diagram? Or maybe there are some other mistakes that you find frequently? Please let me know in the comments section.

# Tips for Better Database Design

[Jeffrey Edison](https://vertabelo.com/authors/jeffrey-edison/)

[Over the past 25 years, Jeffrey has worked in various roles in financial services software, with a focus on transaction banking and automation with artificial intelligence methods. He has extensive knowledge and expertise in financial payments, combating financial crime, and virtual account management. His experience spans various disciplines ranging from architecture and design to product management and sales.](https://vertabelo.com/authors/jeffrey-edison/)

Tags:

* [database design](https://vertabelo.com/tags/database-design)
* [database modeling](https://vertabelo.com/tags/database-modeling)
* [database](https://vertabelo.com/tags/database)
* [design](https://vertabelo.com/tags/design)
* [keys](https://vertabelo.com/tags/keys)
* [naming convention](https://vertabelo.com/tags/naming-convention)
* [tip](https://vertabelo.com/tags/tip)

Over the years, working as a data modeler and database architect, I have noticed that there are a couple rules that should be followed during data modeling and development. Here I describe some tips in the hope that they might help you. I have listed the tips in the order that they occur during the project lifecycle rather than listing them by importance or by how common they are.

### 1. Plan Ahead

Failing to plan is planning to fail.

**Alan Lakein**

One problem that I have seen is when data modeling occurs at the same time as software development. This is like building the foundation before completing the blueprints. In the past, planning seemed an obvious step before starting development. Development teams would not create databases without planning just like architects would not build buildings without blueprints.

In application development, it is critical to understand the nature of the data.

Planning is often ignored so that developers can just “start coding”. The project starts and when issues come up, there is no slack in the schedule to address them. Developers take shortcuts with the intent to fix them later but this rarely if ever happens.

Careful planning is how to ensure that you end up with a proper database that is not hacked together. If you don’t spend the time and effort upfront addressing the data required by the processes, you’ll pay for it later with a database that must be reworked, replaced or scrapped.

Even if planning isn’t always done, many data modelers still follow these guidelines. That’s not to say we can predict every design need in advance, but most modelers believe that it’s worth the effort to understand the data and its usage. You would not want a design for transaction processing when the need is analytic report creation.

Times have changed; Agile methodologies are more prevalent so database teams must rethink their approach to data modeling. In Agile, the Domain Model from Use Cases is used instead of Entity Relationship Diagrams. However, the need for planning has not diminished. We need to understand the data and what it’s supposed to do. In general, the first few Sprints must focus on data design.

So it’s not Agile that is the issue for database modelers, but rather individuals who do not grasp the nature of data. Some see database development as the same as application development. Database modeling and software development are different and need appropriate focus.

The database is the core of most software applications. You must take the time to analyze the requirements and how the data model will meet them. This decreases the chance that the development will lose course and direction.

The developers must understand the importance of data and its contribution to the development process. We live in the information age. Applications display and manipulate data. It is the information contained in the data that gives meaning to the application.

It is not possible to foresee every requirement nor every issue, but it is important to prepare for problems by careful planning.



### 2. Document Your Model

When making a data model, everything seems obvious. You name the objects so that their purpose is evident and everyone will understand the meaning just by reading the name. This may be true, but it isn’t as obvious as you might think. When choosing names for tables and columns, make it clear what the usage of each object will be. Over time, the meaning of objects will be unclear without documentation.

Using a naming convention is one step towards effective documentation. When you have to make changes in the future, you will appreciate any existing documentation. A short, simple document that describes the decisions that you made and describes the design will help explain the design choice at that time.

You want enough documentation so that the database can be managed by a new administrator and they can understand the meaning without having to come back to you for explanation. If the data model and the environment are not documented, it is difficult to maintain or change it as requirements change.

To some extent, documentation has little to do with the data modeling. Documentation is about communicating the design and making it understandable in the future.

Documentation is often an afterthought. When schedules are short, documentation gets ignored. Yet, this is technical debt with a high cost. Cutting corners during the development cycle will accrue costs in the future for database changes, problem identification, tracking bugs and for understanding the data model and nature of the data.

As an example, data models often have an “ID” field as the primary key for a table or a portion of the name of a key. This might be a primary key like **TransactionID** on the Transaction table. If some tables use “Number” as part of the name of a key, then it is good to document why. Perhaps **ReferenceNumber** is used as the name of the primary key on the Message because that is what the reference is called in the business area. For example, in financial services, financial messages typically include a reference number.

Document the definition of tables, columns and relationships so that programmers can access the information. The documentation must describe expectations of the database structure.

In the [**VERTABELO**](http://www.vertabelo.com/) tool, I can immediately include comments on any item: tables, columns, references, alternate keys, which means that the documentation is stored immediately with my model rather than in some extra document to be maintained separately.

EDIT MODEL IN YOUR BROWSER

Poor or absent documentation is often due to shortsighted thinking, but do not ignore its importance. This is still an issue to be addressed.

### 3. Follow Conventions

Naming conventions might not appear important during the design. In reality, names provide the insight to understanding a model. They are an introduction and should be logical.

Inconsistent naming serves no purpose. It only frustrates developers who must access the data, administrators of the database, and modelers who must make changes in the future.

When “ID” is used for some artificial keys but some tables use a different naming convention (such as Number), developers, analysts, and DBAs may waste time to understand the exceptions. Weak naming conventions also lead to errors in development because the naming is not consistent.

Hand-in-hand with documentation, using a naming convention makes it in the future for someone to understand the model. Do not randomly switch between using “ID” (like **CustomerID**) and “Number” (**AccountNumber**) as the keys for tables. Only make exceptions to the conventions when they are justified. Document what the exception is and why the convention is not respected.

The same applies to cryptic names like “XRT1” – is that the extended reference tables? Your guess is as good as mine. I hope that the designer knew why he chose such a cryptic name, but I doubt that the next person to access the database can guess the reason.

Naming conventions are a matter of personal choice. Make sure decisions are consistent and documented.

If I succeeded to convince you to apply naming convention in your database design, feel free to [**READ MY NEXT ARTICLE**](https://vertabelo.com/blog/technical-articles/naming-conventions-in-database-modeling) entirely devoted to this subject.



### 4. Think Carefully About Keys

Keys often generate controversy: primary keys, foreign keys, and artificial keys. Tables need a primary key that identifies each row. The art is to decide which columns should be part of the primary key and what values to include.

For proper normalization, each table needs an identifying key. Uniqueness must be guaranteed. Yet, natural keys and primary keys don’t have to be the same. In fact, they may not be, as long as the table has a natural key.

Some data modelers prefer an artificial key for uniqueness. Yet some modelers prefer a natural key to ensure data integrity.

So, should we use a natural key as the primary key? One challenge arises if the natural key must be changed. If the natural key consists of many columns, you may need to make changes in many places. Another challenge is using an artificial key as the only key for a table.

As an example, you might have a table storing information about products. The table may be defined with an artificial key such as a sequence, a code for the short alphabetic name for the product and the product definition. If uniqueness is ensured only by the artificial key, there may be two rows with the same product code. Are these the same product that is entered twice? Perhaps a key with the product code is more appropriate.

### 5. Use Integrity Checks Carefully

To ensure data integrity, we need foreign keys and constraints. Be careful not to overuse or underuse these integrity checks.

Domain tables are effective for enforcing integrity. Domain tables work well when there are many values to be checked against, or the values to be checked are frequently changing.

One issue can be that developers decide that the application will check integrity. The issue here is that a central database might be accessed by many applications. Also, you generally want to protect the data where it is: in the database.

If the possible values are limited or in a range, then a check constraint may be preferable. Let’s say that messages are defined as either Incoming or Outgoing, in which case there is no need for a foreign key. But, for something like valid currencies, while these may seem static, they actually change from time-to-time. Countries join a currency union and currencies change.

Applications should also perform integrity checks, but don’t rely only on the application for integrity checking. Defining integrity rules on the database ensures that those rules will never be violated. In this way, the data satisfies the defined integrity rules at all times.

### 6. Don’t Forget Indexes in Your Design

Some indexing design is useful during database modeling, even if indexes may change during actual deployment and usage. Of course, it is possible to have too many indexes, just like it is possible to have too few.

Indexing is an ongoing process. During design, you start the process on your model. Design work is on the primary keys and constraints.

Indexes are important when considering queries on the data. When modeling, you should consider how the data will be queried. Take care not to over-index. Indexing revolves around query optimization.

### 7. Avoid Common Lookup Tables

I have often seen a common lookup table for attribute pairs. Defining a single, generic domain table is perceived to simplify the design. This style of domain table makes an abstract definition for holding text. I have heard it called an “Allowed Value” or “Valid Values” table, but the term “MUCK” table was coined for this anti-pattern in 2006: Massively Unified Code-Key.

MUCK tables contain unrelated data.

For example, you could have a table that defines the domain, entry and a description. Thinking back to the message example above, two entries might be:

|  |  |  |
| --- | --- | --- |
| Domain | Entry | Description |
| 1 | I | Incoming message received by the bank |
| 1 | O | Outgoing message sent by the bank |

Now add in entries for another domain:

|  |  |  |
| --- | --- | --- |
| Domain | Entry | Description |
| 2 | COVER | Cover payment |
| 2 | SERIAL | Serial payment |
| 2 | SSI | Standard Settlement Instructions |

This is just a mess. What does the table mean?

Just for fun, I modeled a simple example of a MUCK table (or OTLT, “One True Lookup Table” if you are a Tolkien fan) and included some comments. Please note that this is an anti-pattern and I am not recommending that you use it in your data model.

EDIT MODEL IN YOUR BROWSER

With MUCK tables, constraints cannot be defined. MUCKs may become many rows without any meaning. When you must query another table to understand the meaning of a field, this is not ideal.

These “anything goes” tables make integrity checks complex or even impossible. One reason that these tables may be created is because several tables within the database have a similar definition. Then data integrity checks become impossible. Also, their size may become quite large.

Normalization should lead away from MUCK tables. A single table should represent a single domain; rather than a single (MUCK) table representing all the domains. Without MUCK tables, we can put in place foreign key constraints.

Use separate tables for domain objects rather than cramming them into a single table. This allows proper column types, constraints and relationships. An “Allowed Values” table is just muck and has no place in a data model.



### 8. Define an Archiving Strategy

All too often, I have seen databases created without a proper strategy of data retention and archiving. How long will data be kept online available in active database tables? Most systems are built to keep data in the database “forever”. For most systems, this is not a reasonable long-term data retention strategy. At some point, active data should be archived.

One approach that I advocate is to include data retention as part of your design considerations. Will you have active and historical tables so that inserts of new rows in the active tables remain fast, while searches on historical data can be optimized?

This avoids having to redesign archiving into your database on top of the original design.

### 9. Test Early, Test Often

To paraphrase Al Capone (or John Van Buren, son of the 8th U.S. President), “test early, test often”. In this way, you follow the path of Continuous Integration. Testing at an early development stage saves time and money.

Testing is always a challenge in the development plan. There is often a test phase at the end of an Agile Sprint and system testing at the end of development. Testing is generally the first thing to be squeezed when time gets short.

In testing the database, the goal should be to simulate a production environment: “A Day in the Life of the Database”. What volumes can be expected? What user interactions are likely? Are the boundary cases being handled?

So the testing plan and proper testing must be an integral part of the data modeling and database development.

### Conclusion

These are the main issues that I have seen when working with data modelers and reviewing data models. By paying attention to these tips, your database will be better designed and more robust. Yet, the payback on some of these investments is not always obvious or visible. Plan, document, use standards, create keys, ensure integrity, perform indexing, avoid MUCK, develop strategies, and TEST!

None of these activities will take an enormous amount of time yet have an enormous impact on the quality of your data model.

What are your views about these tips?

#### Do you have tips of your own?

# ERD Symbols and Meanings

Modeling your data using entity-relationship diagram (ERD) with Crow's Foot notation which is popular in Structured Systems Analysis, Barker's Notations, Design Methods and information engineering you need the software that is easy in use having all necessary elements for creating the needed flowcharts as well as their examples. You can find all essential ERD symbols and its meanings on our site with tutorials of how to use it and you can always use the already existing layouts for making your own diagrams in our Solution Park. Once you download our application with extension of hundreds of design elements you’ll find it simple to start using it straight away and you’ll be pleasantly surprised of how smart this tool for making the charts is.

## Design elements Crow’s Foot notation

Crow's Foot notation is used in Barker's Notation, Structured Systems Analysis and Design Method (SSADM), and information engineering. Crow's Foot diagrams represent entities as boxes, and relationships as lines between the boxes. Different shapes at the ends of these lines represent the cardinality of the relationship.

Entity-Relationship model making possibility to describe a database in which in the tables data can be the point to data in other tables — for instance, your entry in the database could point to several entries.

To create an ERD, software engineers mainly turn to dedicated drawing software, which contain the full notation resources for their specific database design — ERD symbols and meanings. CS Odessa has released an all-inclusive Entity-Relationship Diagram (ERD) solution for their powerful drawing program, ConceptDraw DIAGRAM

**Example 1.** Entity Relationship Diagram symbols and meaning

When you need to describe efficiently a database using the Crow’s Foot notation icons, ConceptDraw DIAGRAM gives you the ability to draw Entity Relationship diagram (ERD) fast and easy.

## [ERD Symbols and Meanings](https://www.conceptdraw.com/How-To-Guide/picture/Entity-Relationship-Diagram-Symbols-and-Meaning.png)Design elements Chen's notation

The Chen's ERD notation is still used and is considered to present a more detailed way of representing entities and relationships.

Use also ConceptDraw DIAGRAM enhanced with powerful Entity-Relationship Diagram (ERD) solution to draw your own ER diagrams using Chen's notation icons.

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Shape Name** | **Symbol Description** |
| **Entities** | | |
| ERD Symbols and Meaning - Entity | Entity | An entity is represented by a rectangle which contains the entity’s name. |
| ERD Symbols and Meaning - Weak Entity | Weak Entity | An entity that cannot be uniquely identified by its attributes alone. The existence of a weak entity is dependent upon another entity called the owner entity. The weak entity’s identifier is a combination of the identifier of the owner entity and the partial key of the weak entity. |
| ERD Symbols and Meaning - Associative Entity | Associative Entity | An entity used in a many-to-many relationship (represents an extra table). All relationships for the associative entity should be many |
| **Attributes** | | |
| ERD Symbols and Meaning - Attribute | Attribute | In the Chen notation, each attribute is represented by an oval containing atributte’s name |
| ERD Symbols and Meaning - Key attribute | Key attribute | An attribute that uniquely identifies a particular entity. The name of a key attribute is underscored. |
| ERD Symbols and Meaning - Multivalue attribute | Multivalued attribute | An attribute that can have many values (there are many distinct values entered for it in the same column of the table). Multivalued attribute is depicted by a dual oval. |
| ERD Symbols and Meaning - Derived attribute | Derived attribute | An attribute whose value is calculated (derived) from other attributes. The derived attribute may or may not be physically stored in the database. In the Chen notation, this attribute is represented by dashed oval. |
| **Relationships** | | |
| ERD Symbols and Meaning - Relationship | Strong relationship | A relationship where entity is existence-independent of other entities, and PK of Child doesn’t contain PK component of Parent Entity. A strong relationship is represented by a single rhombus |
| ERD Symbols and Meaning - Identifying Relationship | Weak (identifying) relationship | A relationship where Child entity is existence-dependent on parent, and PK of Child Entity contains PK component of Parent Entity. This relationship is represented by a double rhombus. |

In addition to the vector stencil image libraries with full standardized Chen and Crow’s foot notations included, the [Entity-Relationship Diagram (ERD) solution](https://www.conceptdraw.com/solution-park/software-erd) contains a number of ERD templates and samples added to the solution pack. They will help users quick start representing their own ideas.

**Example 2.** Entity Relationship Diagram samples and templates in ConceptDraw STORE

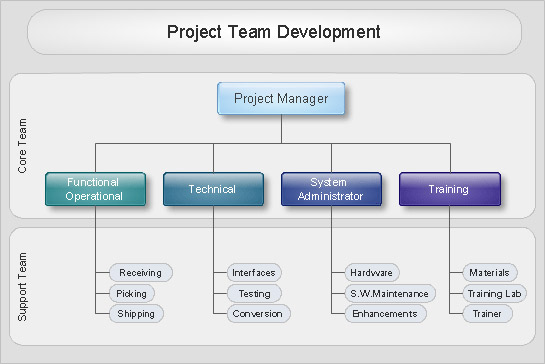
Try it today for saving your time sake not to look for something else once you have great chance to use this excellent product already now!

**The following features make ConceptDraw DIAGRAM the best Entity Relationship Diagram ERD Software:**

1. You don't need to be an artist to draw professional looking diagrams in a few minutes.
2. Large quantity of ready-to-use vector objects (entity relationship diagram symbols and meaning) makes your drawing diagrams quick and easy.
3. Great number of predesigned templates and samples give you the good start for your own diagrams.
4. ConceptDraw DIAGRAM provides you the possibility to use the grid, rules and guides. You can easily rotate, group, align, arrange the objects, use different fonts and colors to make your diagram exceptionally looking.
5. All ConceptDraw DIAGRAM documents are vector graphic files and are available for reviewing, modifying, and converting to a variety of formats: image, HTML, PDF file, MS PowerPoint Presentation, Adobe Flash, MS Visio.
6. Using ConceptDraw STORE you can navigate through ConceptDraw Solution Park, managing downloads and updates. You can access libraries, templates and samples directly from the ConceptDraw STORE.
7. If you have any questions, our free of charge support is always ready to come to your aid.

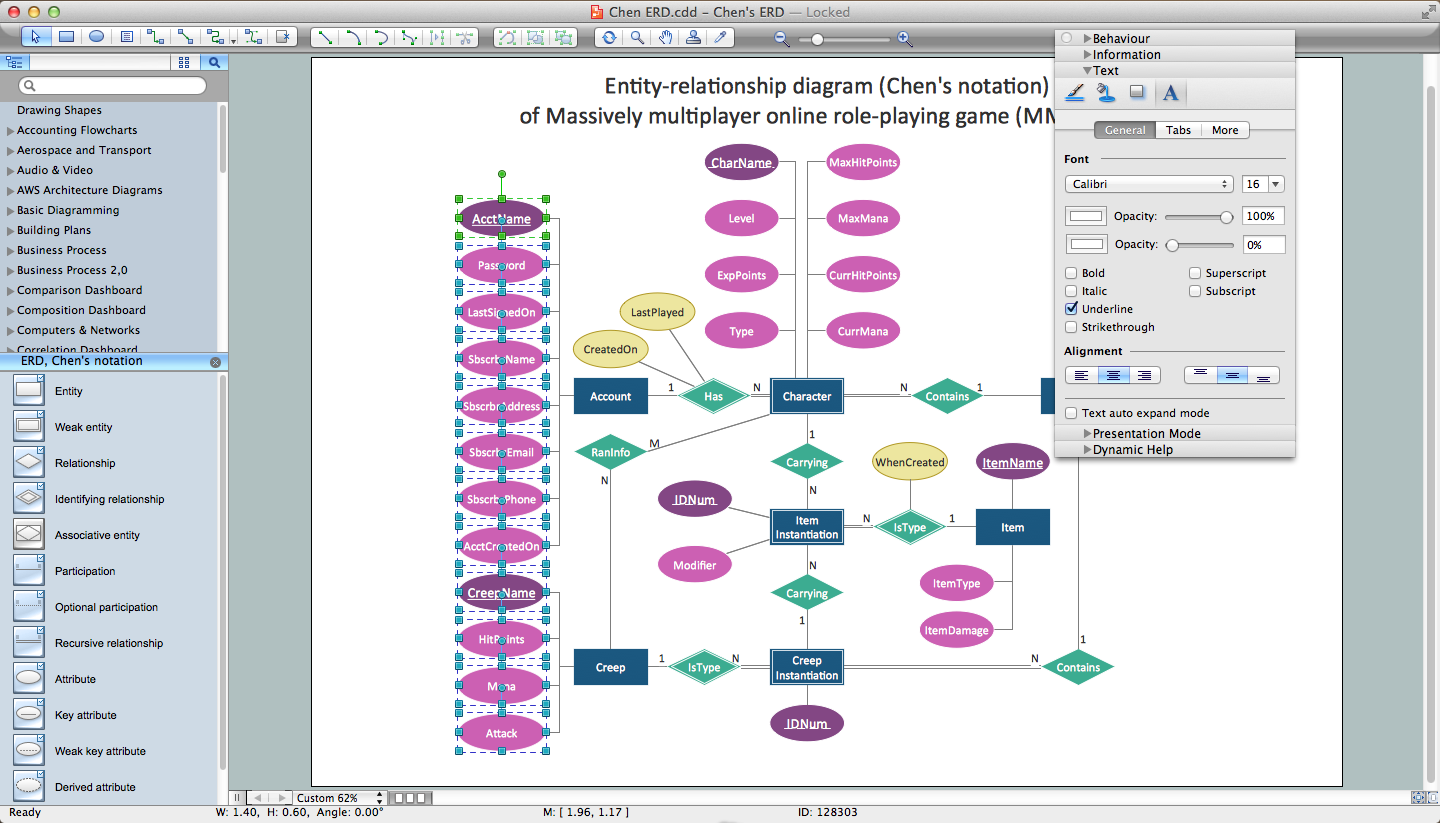
#### [Entity Relationship Diagram samples and templates in ConceptDraw STORE](https://www.conceptdraw.com/How-To-Guide/picture/erd-symbols-and-meanings/Software-Development-Entity-Relationship-Diagram-(ERD).png)NINE RELATED HOW TO's:

### [**Organizational Charts with ConceptDraw DIAGRAM**](https://www.conceptdraw.com/How-To-Guide/orgchart-software)**→**

To your mind, deliberation about the profits of creating organizational charts takes more time than actually creating itself. If lack of software is the only thing that stops you, you should know that organizational charts with ConceptDraw DIAGRAM are created within a few minutes. Owing to the great number of ready-to-use templates, you’ll get your chart easily. Chart is a very handy format for visually depicting official relations of an organization. Usually, an Orgchart (organizational chart) consists from simple geometrical shapes, containing an information on a position, personal data, and the lines that means responsibilities and accountabilities associated with position. ConceptDraw solution designed for making organizational charts enables one to create various org charts effortlessly using special templates and vector libraries.

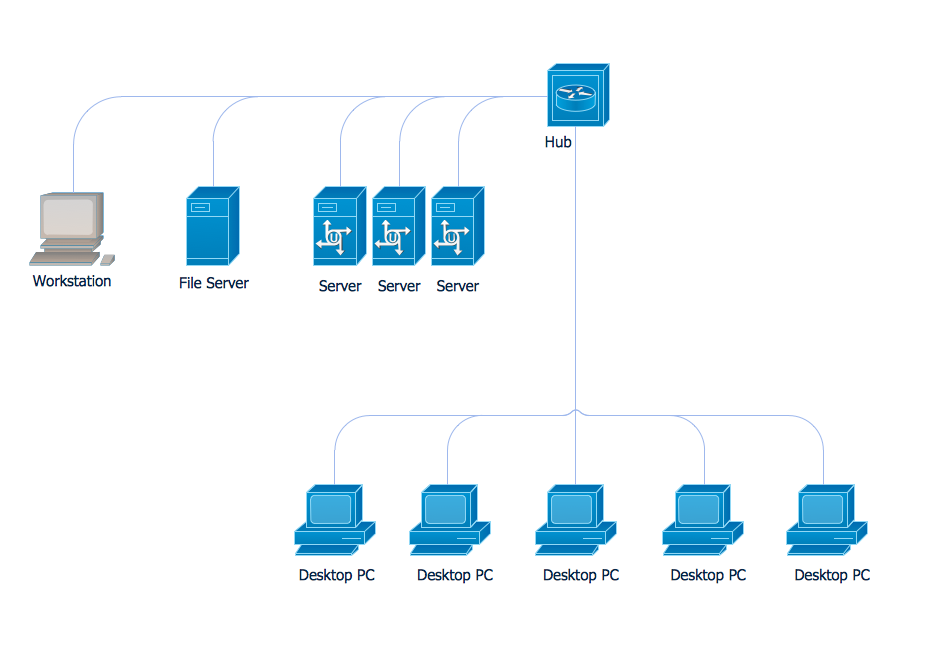
Picture: Organizational Charts with ConceptDraw DIAGRAM

### [**How to Draw ER Diagrams**](https://www.conceptdraw.com/How-To-Guide/erd-how-to-draw-er-diagrams)**→**

How to Draw ER Diagrams? With Entity-Relationship Diagram (ERD) solution from the Software Development Area for ConceptDraw Solution Park you are able to draw ER diagram with no problem. It includes 45 predesigned icons advocated by popular Chen's and Crow’s Foot notations that can be used when describing an enterprise database.

Picture: How to Draw ER Diagrams

### [**Bus Network Topology**](https://www.conceptdraw.com/How-To-Guide/bus-network-topology)**→**

If you have a small budget to design a computer network, you have to be very careful. One of the most cheap technologies to implement is a bus network topology, however it has many disadvantages. For instance, if the network cable is somehow damaged, the entire network won't work. This diagram illustrates a so-called "Bus" network topology. This type of network arrangement means that each computer or other device is linked to a main link (bus). The end nodes are shown as a circle. The links to the bus are depicted as solid vertical lines. The bus is shown as a bold horizontal line. This diagram can serve as a template for creating logical or physical network diagrams. The set of vector libraries supplied with ConceptDraw Computer and Networks solution contains the symbols of all LAN and WLAN elements required for creating network diagrams of any configuration.

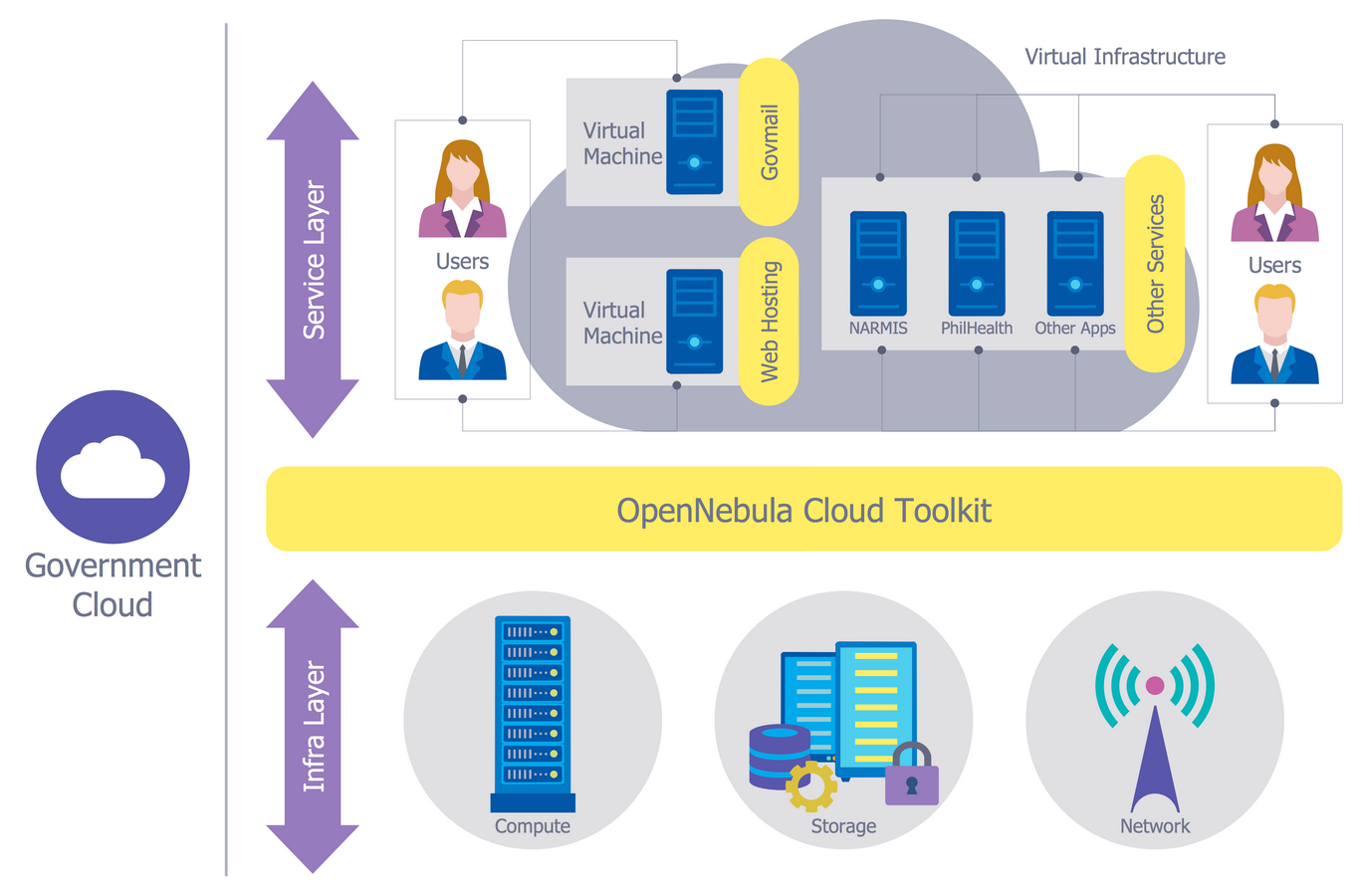
Picture: Bus Network Topology

### [**What Is a Concept Map**](https://www.conceptdraw.com/How-To-Guide/what-is-a-concept-map)**→**

What is a Concept Map and what software is effective for its design? A concept map or conceptual diagram is a diagram that depicts suggested relationships between concepts. As for effective software - we suggest you to pay attention for the ConceptDraw DIAGRAM diagramming and vector drawing software. Extended with Concept Maps Solution from the "Diagrams" Area it is a real godsend for you.

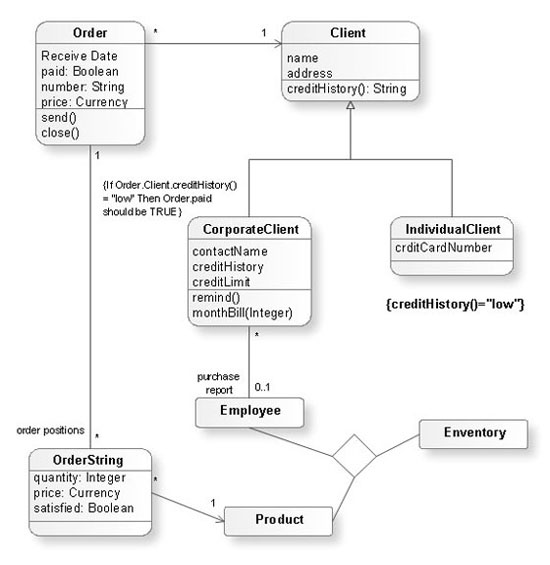
Picture: What Is a Concept Map

### [**Network Security Model**](https://www.conceptdraw.com/How-To-Guide/network-security-model)**→**

Creation of well thought-out network security model will effectively help you in realization your network's security. ConceptDraw DIAGRAM diagramming and vector drawing software extended with Network Security Diagrams Solution from the Computer and Networks Area of ConceptDraw Solution Park lets fast and easy design all variety of network security models: Open network security model, Closed network security model, Restrictive network access model.

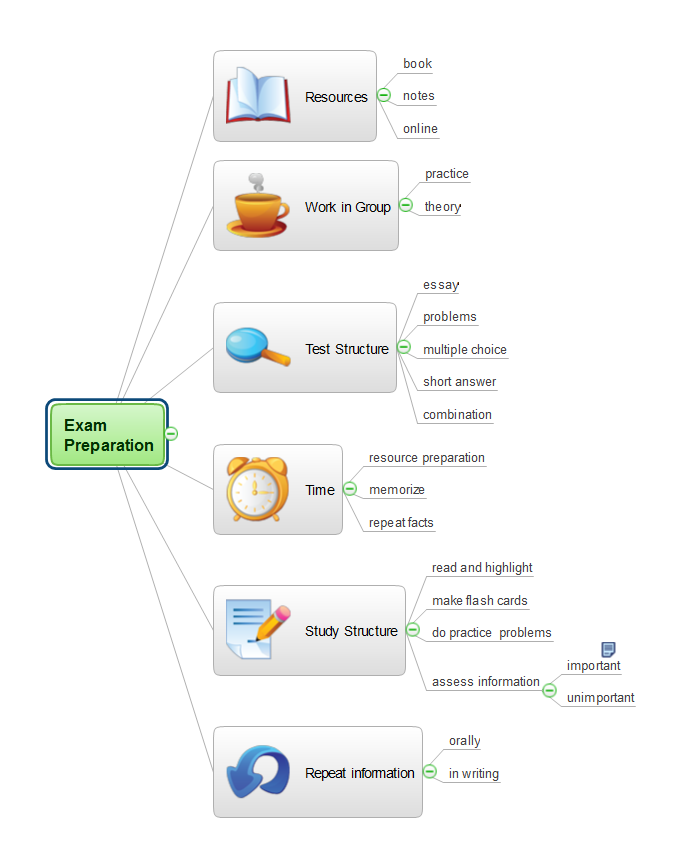
Picture: Network Security Model

### [**Data Flow Diagrams**](https://www.conceptdraw.com/How-To-Guide/data-flow)**→**

Software development flourishes nowadays. Students have tons of educational information and courses that would fit every taste. And it's no secret that knowing how to draw data flow diagrams is a useful skill for a future software architect. The diagrams help a lot in understanding how to storage data effectively and how data processes within a system. This data flow diagram represents the model of small traditional production enterprise. It was created using Yourdon and Coad notation. The data flow diagram include four main objects: entity, process, data store and data flow. Yourdon and Coad notation uses circles to depict processes and parallel lines to represent data stores. Dataflows are shown as arrowed lines. They are labeled with the description of the data that move through them. Dataflow's role is to deliver the pieces of information. A process's function is to transform. the input data flow into output data flow. ConceptDraw Data Flow Diagrams solution contains the data flow diagram symbols of both Yourdon-Coad and Gane-Sarson notations and a set of handy templates to get started with DFD.

Picture: Data Flow Diagrams

### [**Exam Preparation Mind Map**](https://www.conceptdraw.com/How-To-Guide/mindmap-exam-preparation)**→**



Picture: Exam Preparation Mind Map

Related Solution:

Related Solution:

Related Solution:

Related Solution:

Related Solution:

Related Solution:

Related Solution:

[Organizational Charts](https://www.conceptdraw.com/solution-park/management-org-charts)

[Entity-Relationship Diagram (ERD)](https://www.conceptdraw.com/solution-park/software-erd)

[Computer Network Diagrams](https://www.conceptdraw.com/solution-park/computer-and-networks)

[Concept Maps](https://www.conceptdraw.com/solution-park/diagram-concept-maps)

[Network Security Diagrams](https://www.conceptdraw.com/solution-park/computer-networks-network-security-diagrams)

[Data Flow Diagrams (DFD)](https://www.conceptdraw.com/solution-park/software-DFD)

[Mind Map Exchange](https://www.conceptdraw.com/solution-park/collaboration-mind-map-exchange)

# What Are the Symbols Used in an ER Diagram?

[Katarzyna Piskała](https://vertabelo.com/authors/katarzyna-piskala/)

[Katarzyna has a Master’s degree in Mathematics. At Vertabelo Academy, LearnSQL.com, and LearnPython.com, she creates and improves programming courses. She also helps users of these sites to improve themselves. In her free time, she loves traveling, dancing, and teaching.](https://vertabelo.com/authors/katarzyna-piskala/)

Tags:

* [ER diagram](https://vertabelo.com/tags/er-diagram)
* [ERD diagram](https://vertabelo.com/tags/erd-diagram)
* [Entity-Relationship Diagram](https://vertabelo.com/tags/entity-relationship-diagram)

Find out what symbols are used in the Entity-Relationship Diagram (ERD) and what they mean.

The most popular notation in ER diagrams is the Information Engineering (IE) notation, also called [**CROW’S FOOT NOTATION**](https://vertabelo.com/blog/crow-s-foot-notation/). This is the default ER diagram notation used in Vertabelo.

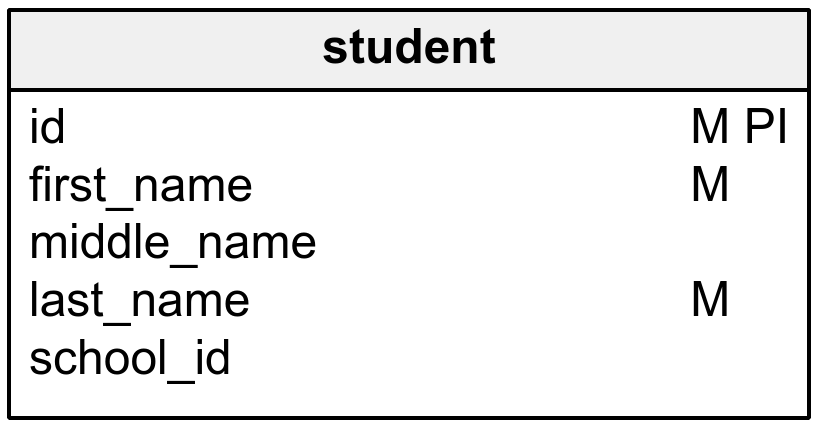
There are a few standard symbols used in logical and physical ER diagrams, and some useful additional non-standard symbols that you can use in [**VERTABELO**](http://vertabelo.com/). We’ll discuss them in this article.

## Standard Symbols Used in Logical ER Diagrams

When creating a logical ER diagram, you typically use the following symbols:

### Entity

The entity is denoted by a rectangle. This rectangle is divided into two parts: the entity name at the top and the entity attributes in the lower part. You can list the attributes with or without their types. Here's an example of an entity in Vertabelo:



The M annotation next to the attribute stands for "mandatory" – you use it for attributes that can't be left empty. In other words, M attributes must have a value.

The PI annotation stands for "primary identifier". You use this for the attribute(s) that uniquely identify the whole entity.

You can also specify attributes’ data types, as shown below:

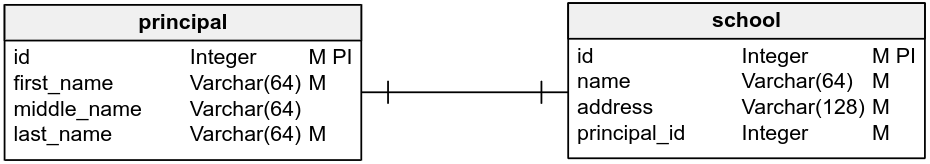
### Relationships

After creating entities, you may need to specify the relationships between them. These relationships are denoted in diagrams by lines. Each relationship is one line and has cardinality and mandatory attributes.

Relationships can be one-to-one, one-to-many, or many-to-many. The many side is denoted by the crow’s foot symbol; one is denoted by a single line.

Each entity in a relationship can be mandatory or optional. If the entity is mandatory, we denote it by a vertical segment. If it's not mandatory, we denote it as an open circle.

Let’s consider some examples:



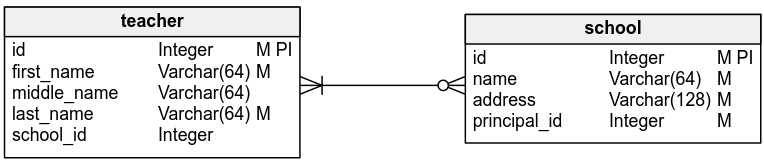
This is a one-to-one relationship. Each principal is a principal in only one school and each school has only one principal, so there are no crow’s feet.

Each entity is mandatory – each school needs to have a principal and there is no principal without the school. This is shown by the vertical bars near each entity.



Above, we see a many-to-one relationship. Each school has many students (hence the crow’s foot near the students entity) but each student is only in one school.

Each entity is mandatory (hence the vertical line). Each student must be assigned to a school and each school must have some students.



This is a many-to-many relationship. Each school has many teachers and the teachers can teach in a few different schools. In this kind of relationship, there are crow’s feet at both ends.

The teacher entity is mandatory – each school must have some teachers. On the other hand, a teacher can be currently unemployed but still in the database. So the school entity isn't mandatory, which we denote by an empty circle.

You can read more about many-to-many relationships [**HERE**](https://vertabelo.com/blog/many-to-many-relationship/).

## Nonstandard Logical ER Diagram  Symbols Used in Vertabelo

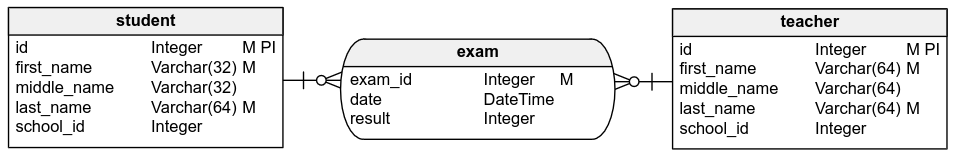
If you’re dealing with complicated databases, the above notations may not be enough. That's why Vertabelo also supports some nonstandard functions (e.g. association and inheritance) you can use when creating your logical ER diagram. Using these nonstandard symbols can make a programmer’s life a lot easier.

### Association

An association is a special representation of a relationship between two entities – specifically, a relationship with additional attributes. In this way, you can realize a many-to-many relationship with attributes. Or you can implement a ternary relationship, which cannot be achieved through a simple relationship between entities.

Associations are represented in a diagram by a rectangle with rounded corners. Like entities, associations have names (displayed at the top) and attributes (displayed in the lower part of the rounded rectangle). You can (but don't have to) specify the attributes’ types. You can also leave comments about the association or edit its appearance.

Here’s an example of an association in a logical diagram:



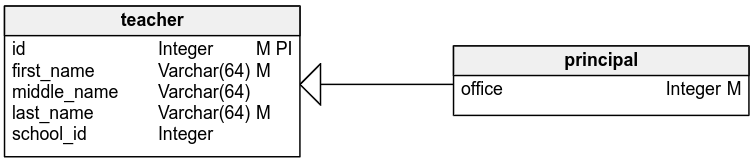
This is an association between students and teachers. The relationship is represented by a rectangle with rounded corners. Inside we have the association name and its attributes. You can choose their data types and specify whether they are mandatory or optional. On the other hand, since it's an association and not the entity, there is no such thing as a primary identifier. As with basic relationships, you can also specify cardinality for the entity-association relationship.

### Inheritance

The other nonstandard feature Vertabelo allows you to use in logical ER diagrams is inheritance. Traditional ER diagrams don't have this concept, but it has become a standard modeling technique.

Inheritance is denoted by an arrow that points from the child entity to the parent entity. It's a UML notation. The child entity inherits all parent's attributes, their types, and whether they're mandatory and are primary identifiers. You can also add some other columns if you'd like.

Take a look at the following example of inheritance:



A principal is a specific type of teacher, so the principal entity inherits from the teacher entity; you want the principal to have all the attributes a teacher has. Here, principal is the child entity and teacher is the parent entity; thus, the arrow points to the teacher entity. If you want the principal to have some additional attributes (e.g. an office) just list these attributes in the lower part of the principal entity.

Now that you know the symbols, feel free to [**CREATE YOUR OWN LOGICAL ER DIAGRAM**](https://vertabelo.com/blog/logical-diagrams/). Not sure if you understand everything mentioned above? Or do you want to know more? There's no better place than the Vertabelo Data Modeler [**DOCUMENTATION**](https://www.vertabelo.com/documentation/) to learn answers to your questions.

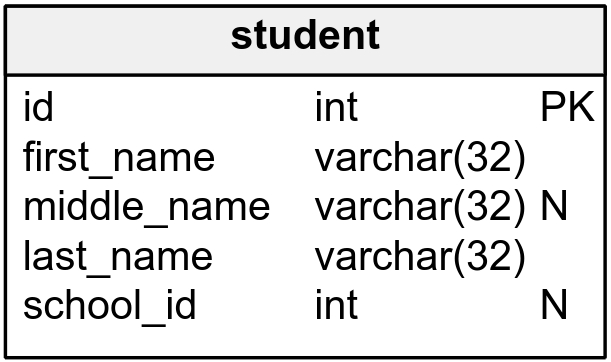
Whether you already have a logical ER diagram or you feel like you don't need it, you can create a physical ER diagram in Vertabelo. This kind of diagram is much more specific than the logical ER diagram. As a matter of fact, it's an evolution of the logical diagram. It has some other symbols and other functions, but you may find it quite similar to the logical diagram.

## Symbols Used in Physical ER Diagrams

When creating a physical ER diagram, you typically use the following symbols:

### Table

The table is similar to the logical diagram's entity. It's denoted by a rectangle divided into two parts: the table name at the top and the table columns lower down. You list the columns with their data types; you can choose them from a list or simply type them. Here's an example of a table in Vertabelo:



N stands for "nullable" – you use it when there can be NULL values in a column. Otherwise, the NULL values won't be accepted. Here, the **middle\_name** column is not mandatory, so this column is nullable.

PK stands for "primary key", which is a unique key used to identify the whole table. Here, the student should be identified by their **id** – a column with unique, non-repeating values. It may happen that two students in the same school have exactly the same name, so you must be able to distinguish them using another unique identifier – in this case, the **id** column.

Note that a primary key column can't have NULL values. Vertabelo makes sure you don't make such mistakes by automatically blocking the NULL option when you indicate a column is the primary key.

Tables also have one or more foreign keys when they are in a relationship with other tables. Foreign keys are denoted by the  FK notation.

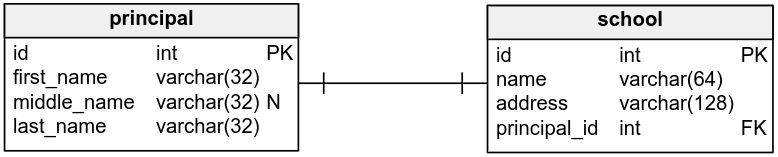
### References

Just like the relationships between entities in the logical ER diagram, references between tables are denoted by lines. Each reference is one line and has cardinality and mandatory attributes.

References in the physical ER diagram can be one-to-one or one-to-many. Many is denoted by the crow’s foot; one is denoted by a vertical line. There are no many-to-many references in the physical diagrams.

Each table in a reference can be mandatory or optional. If the table is mandatory, we denote it by a vertical segment. If it's not mandatory, we denote it by an open circle.

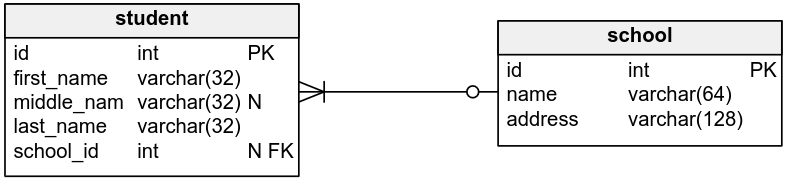
Here are some examples:



This is a one-to-one reference. Each principal is the principal in only one school and each school has only one principal, so there are no crow’s feet.

Each table is mandatory – each school needs to have a principal and there is no principal without the school. This is shown by the vertical bars.

The **principal\_id** column in the **school** table is created automatically in Vertabelo, along with the foreign key symbol. It's also automatically removed when you remove the reference between the tables.



Above we have a many-to-one reference. Each school has many students (hence the crow’s foot near the **students** table) but each student is only in one school.

In case you decide that you still want to keep the students who graduated or dropped out of school in the database, we’ve made the **school** table optional;  this is denoted by an open circle. But each school still must have some students. Thus, the **student** table is mandatory (which is denoted by the vertical line).

Again, the **student\_id** column in the **school** table is created automatically and it's a foreign key. Also, because the **school** table isn't mandatory, the foreign key can be NULL – hence the nullable symbol.

## Other ER Diagram Notations

These are the basics you need to create an ER diagram in Vertabelo. Of course, there's more to it when it comes to physical ERDs. You can create constraints, default values, unique keys, etc. Those objects are denoted in the diagram, but the information about their definition is kept in the physical data model in Vertabelo. You can explore them using the Table properties panel or Model properties panel.

Now that you know the default notation, you should be aware that Vertabelo also supports other notations: [**UML**](https://vertabelo.com/blog/uml-notation/)and [**IDEF1X**](https://vertabelo.com/blog/idef1x-notation/) for logical diagrams; UML, [**BARKER’S NOTATION**](https://vertabelo.com/blog/barkers-erd-notation/), and IDEF1X for physical diagrams.

You’ll be pleased to know that Vertabelo also lets you [**CREATE PHYSICAL DIAGRAMS FROM LOGICAL DIAGRAMS AUTOMATICALLY**](https://vertabelo.com/blog/logical-diagrams/). You can even generate the SQL script automatically from the diagram. It simplifies everything!

Thanks for reading this article. If you have questions or comments on ERDs or diagram notations, tell us about them in the comments section!

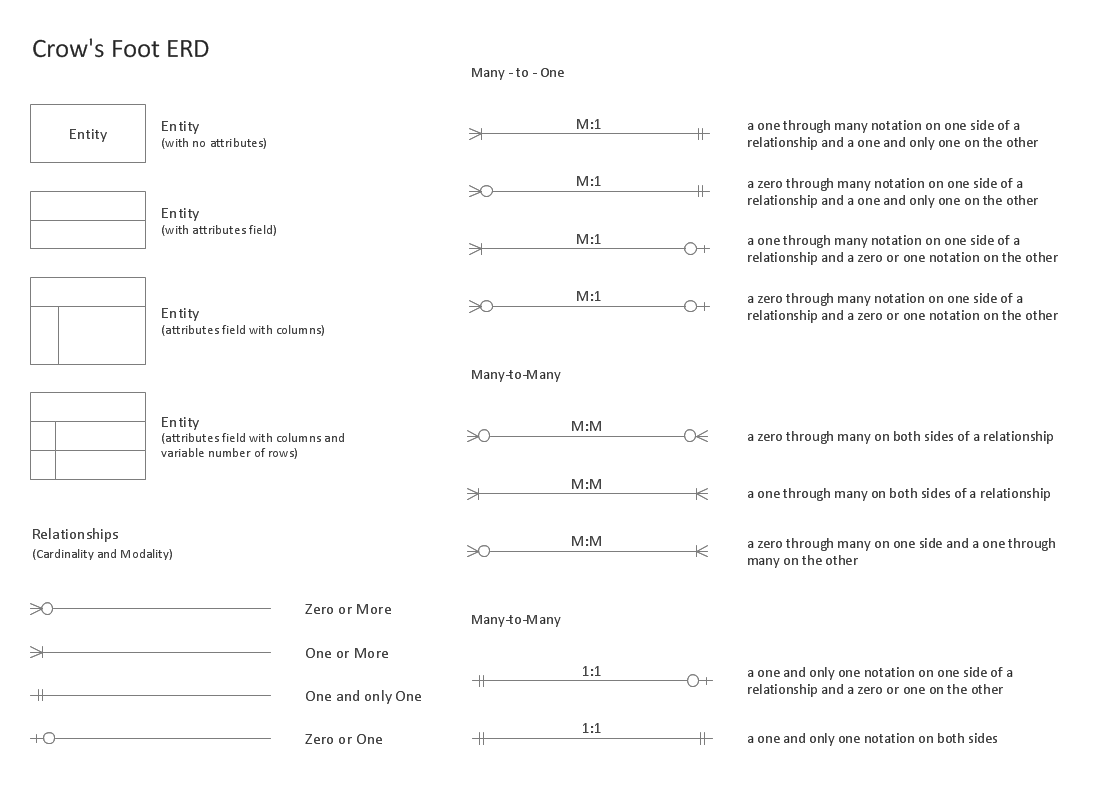
* Install this solution Free
* [What I need to get started](https://www.conceptdraw.com/solution-park/diagramming-ERD" \l "!howto)
* Solution Requirements

This solution requires the following products to be installed:  
[ConceptDraw DIAGRAM v16](https://www.conceptdraw.com/products/drawing-tool)

* Compatibility

macOS 10.15, 11.0  
MS Windows 8.1, 10

* [Support for this Solution](https://helpdesk.conceptdraw.com/ticket.php)
* [Helpdesk](https://www.conceptdraw.com/solution-park/diagramming-ERD" \l "helpdesk)



This solution extends ConceptDraw DIAGRAM software with the ability to describe a database using the Entity-Relationship model.

This solution includes icons advocated by Chen's and Crow’s Foot notation that can be used when describing a database.

The vector graphic diagrams produced when using this solution can be used in your whitepapers, presentations, datasheets, posters or any technical material.

* [Main](https://www.conceptdraw.com/solution-park/diagramming-ERD)
* [How-to](https://www.conceptdraw.com/solution-park/diagramming-ERD" \l "!howto)

# **Entity-Relationship Diagram (ERD)**

## What I Need to Get Started

Both ConceptDraw DIAGRAM diagramming and drawing software and the Entity-Relationship Diagram (ERD) solution can help creating the you need. The Entity-Relationship Diagram (ERD) solution can be found in the Universal Diagramming area of ConceptDraw STORE application that can be downloaded from this site. Make sure that both ConceptDraw DIAGRAM and ConceptDraw STORE applications are installed on your computer before you get started.

#### How to install

After ConceptDraw STORE and ConceptDraw DIAGRAM are downloaded and installed, you can install the Entity-Relationship Diagram (ERD) solution from the ConceptDraw STORE.

#### Start Using

To make sure that you are doing it all right, use the pre-designed symbols from the stencil libraries from the solution to make your drawings look smart and professional. Also, the pre-made examples from this solution can be used as drafts so your own drawings can be based on them. Using the samples, you can always change their structures, colors and data.

## Inside

## HelpDesk

[How to Use an ERD Solution](https://www.conceptdraw.com/helpdesk/how-to-use-erd-solution)

## Example 2: Entity-Relationship Diagram Using Crow's Foot Notation

This diagram was created in ConceptDraw DIAGRAM using the Crows Foot ERD library from the Entity-Relationship Diagram (ERD) solution. An experienced user spent 20 minutes creating this sample.

This sample diagram using the Entity-Relationship Diagram (ERD) Solution and shows the type of icons and graphics you can use to develop a model of a database of arbitrary complexity using Crow's Foot notation.

This diagram example was redrawn from [*https://www2.cs.uregina.ca/~bernatja/crowsfoot.html*](https://www2.cs.uregina.ca/~bernatja/crowsfoot.html) using ConceptDraw DIAGRAM software enhanced with ConceptDraw ERD solution.

## Example 1: Entity-Relationship Diagram Using Chen's Notation

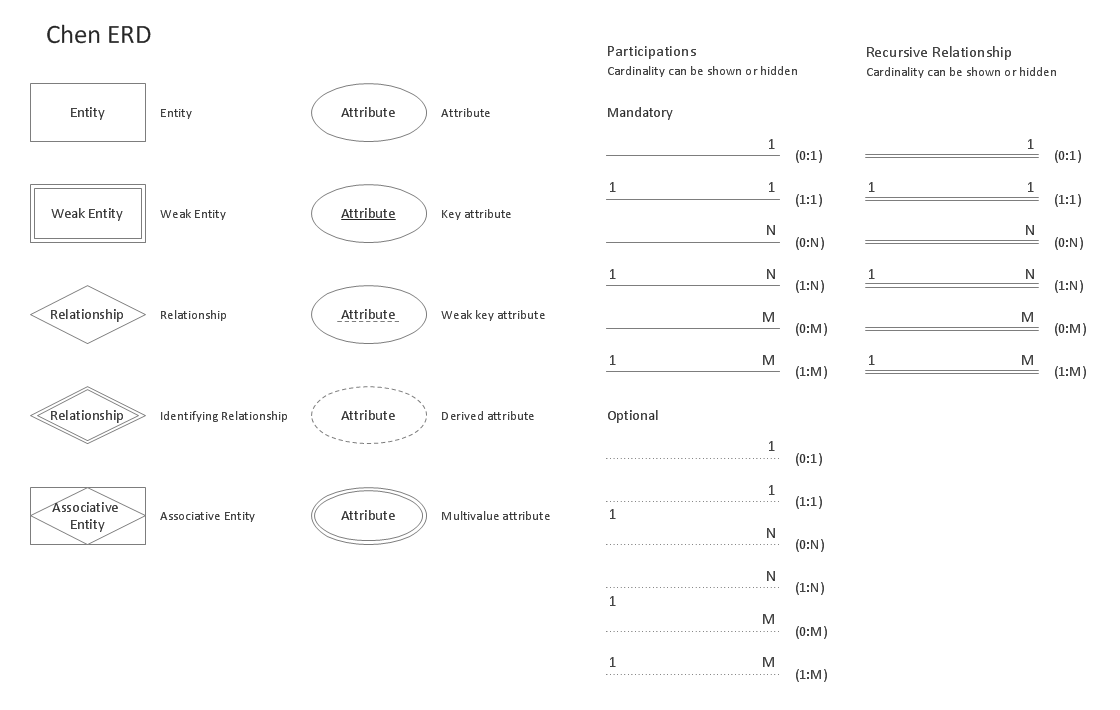
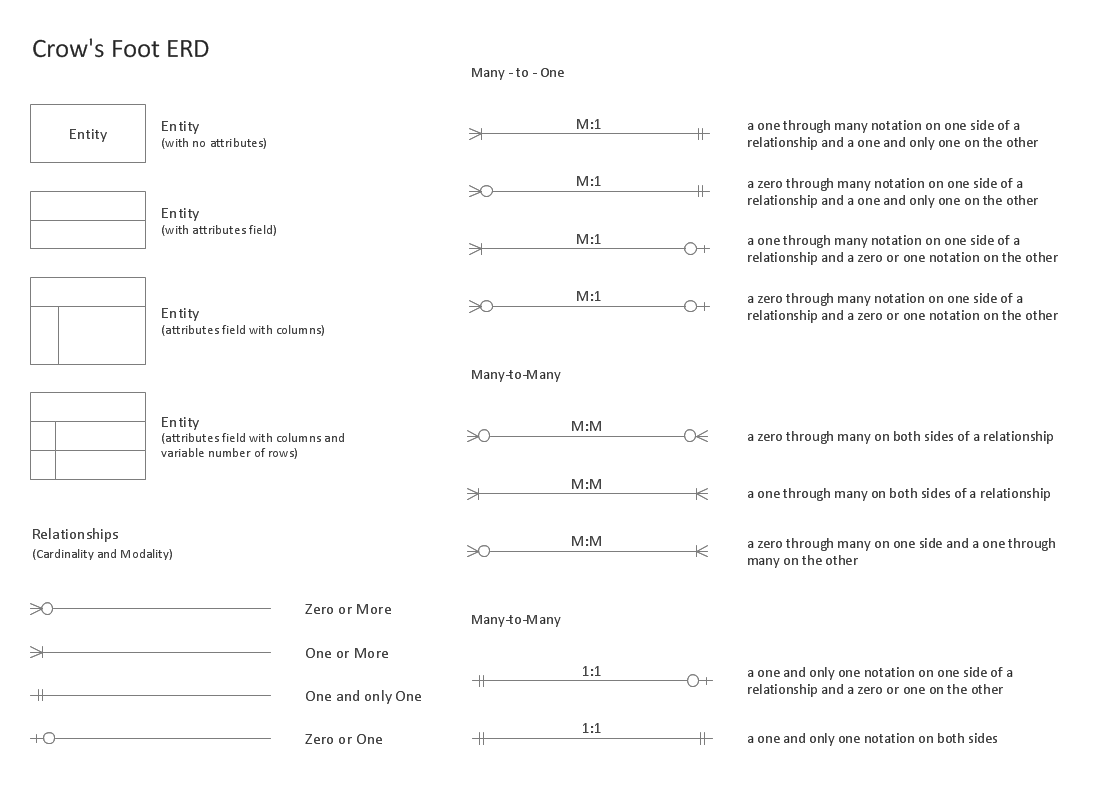
This diagram was created in ConceptDraw DIAGRAM using the Chen ERD library from the Entity-Relationship Diagram (ERD) solution. An experienced user spent 20 minutes creating this sample.

This sample diagram using the Entity-Relationship Diagram (ERD) Solution shows the type of icons and graphics you can use to develop a model of a database of arbitrary complexity according to Chen's notation.

This diagram example was redrawn from [*https://en.wikipedia.org/wiki/File:ER\_Diagram\_MMORPG.png*](https://en.wikipedia.org/wiki/File:ER_Diagram_MMORPG.png)  
using ConceptDraw DIAGRAM software enhanced with ConceptDraw ERD solution.

## Design Elements

2 libraries, 45 objects included in Entity-Relationship Diagram (ERD).



## More Examples and Templates

## Examples

There are a few samples that you see on this page which were created in the ConceptDraw DIAGRAM application by using the Entity-Relationship Diagram (ERD) solution. Some of the solution's capabilities as well as the professional results which you can achieve are all demonstrated here on this page.

All source documents are vector graphic documents which are always available for modifying, reviewing and/or converting to many different formats, such as MS PowerPoint, PDF file, MS Visio, and many other graphic ones from the ConceptDraw Solution Park or ConceptDraw STORE. The Entity-Relationship Diagram (ERD) solution is available to all ConceptDraw DIAGRAM users to get installed and used while working in the ConceptDraw DIAGRAM diagramming and drawing software.

## Related News:

[New Entity-Relationship Diagram (ERD) Solution Announced by CS Odessa for ConceptDraw DIAGRAM](https://www.conceptdraw.com/news/article.php?nid=NID-7537)



[Entity-Relationship Diagram (ERD) Sample — Students and Teachers](https://www.conceptdraw.com/solution-park/resource/images/solutions/entity-relationship-diagram-(erd)/Diagramming-Students-and-Teachers-ERD-Sample48.png)

# 11 important database designing rules which I follow



[**Shivprasad koirala**](https://www.codeproject.com/script/Membership/View.aspx?mid=1335831)

**Rate me:**













4.80/5 (112 votes)

25 Feb 2014[CPOL](http://www.codeproject.com/info/cpol10.aspx)9 min read

This article will discuss about 11 important database designing rules.

## Table of Contents

* [Introduction](https://www.codeproject.com/articles/359654/11-important-database-designing-rules-which-i-fo-2" \l "Introduction)
* [Rule 1: What is the nature of the application (OLTP or OLAP)?](https://www.codeproject.com/articles/359654/11-important-database-designing-rules-which-i-fo-2" \l "Rule1:-WhatistheNatureoftheapplication(OLTPorOLAP))
* [Rule 2: Break your data in to logical pieces, make life simpler](https://www.codeproject.com/articles/359654/11-important-database-designing-rules-which-i-fo-2" \l "Rule2:-Breakyourdataintologicalpieces,makelifesimpler)
* [Rule 3: Do not get overdosed with rule 2](https://www.codeproject.com/articles/359654/11-important-database-designing-rules-which-i-fo-2" \l "Rule3:-Donotgetoverdosedwithrule2)
* [Rule 4: Treat duplicate non-uniform data as your biggest enemy](https://www.codeproject.com/articles/359654/11-important-database-designing-rules-which-i-fo-2" \l "Rule4:-Treatduplicatenonuniformdataasyourbiggestenemy)
* [Rule 5: Watch for data separated by separators](https://www.codeproject.com/articles/359654/11-important-database-designing-rules-which-i-fo-2" \l "Rule5:-Watchfordataseparatedbyseparators.)
* [Rule 6: Watch for partial dependencies](https://www.codeproject.com/articles/359654/11-important-database-designing-rules-which-i-fo-2" \l "Rule6:-Watchforpartialdependencies.)
* [Rule 7: Choose derived columns preciously](https://www.codeproject.com/articles/359654/11-important-database-designing-rules-which-i-fo-2" \l "Rule7:-Choosederivedcolumnspreciously)
* [Rule 8: Do not be hard on avoiding redundancy, if performance is the key](https://www.codeproject.com/articles/359654/11-important-database-designing-rules-which-i-fo-2" \l "Rule8:-Donotbehardonavoidingredundancy,ifperformanceisthekey)
* [Rule 9: Multidimensional data is a different beast altogether](https://www.codeproject.com/articles/359654/11-important-database-designing-rules-which-i-fo-2" \l "Rule9:-Multidimensionaldataisadifferentbeastaltogether)
* [Rule 10: Centralize name value table design](https://www.codeproject.com/articles/359654/11-important-database-designing-rules-which-i-fo-2" \l "Rule10:-Centralizenamevaluetabledesign)
* [Rule 11: For unlimited hierarchical data self-reference PK and FK](https://www.codeproject.com/articles/359654/11-important-database-designing-rules-which-i-fo-2" \l "Rule11:-Forunlimitedhierarchicaldataself-referencePKandFK)



***Courtesy: Image from Motion pictures***

## Introduction

Before you start reading this article let me confirm to you I am not a guru in database designing. The below 11 points are what I have learnt via projects, my own experiences, and my own reading. I personally think it has helped me a lot when it comes to DB designing. Any criticism is welcome.

The reason I am writing a full blown article is, when developers design a database they tend to follow the three normal forms like a silver bullet. They tend to think normalization is the only way of designing. Due this mind set they sometimes hit road blocks as the project moves ahead.

If you are new to normalization, then click and see [3 normal forms](http://youtu.be/wp0N1tYjEWc?hd=1) in action which explains all the three normal forms step by step.

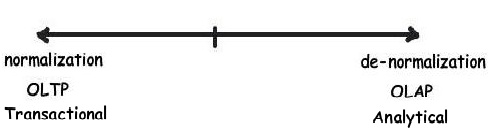
Said and done normalization rules are important guidelines but taking them as a mark on stone is calling for trouble. Below are my own 11 rules which I remember on the top of my head while doing DB design.

## Rule 1: What is the nature of the application (OLTP or OLAP)?

When you start your database design the first thing to analyze is the nature of the application you are designing for, is it Transactional or Analytical. You will find many developers by default applying normalization rules without thinking about the nature of the application and then later getting into performance and customization issues. As said, there are two kinds of applications: transaction based and analytical based, let’s understand what these types are.

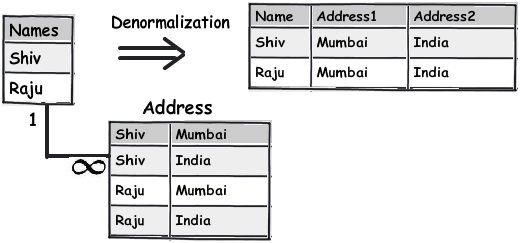
Transactional: In this kind of application, your end user is more interested in CRUD, i.e., creating, reading, updating, and deleting records. The official name for such a kind of database is OLTP.

Analytical: In these kinds of applications your end user is more interested in analysis, reporting, forecasting, etc. These kinds of databases have a less number of inserts and updates. The main intention here is to fetch and analyze data as fast as possible. The official name for such a kind of database is OLAP.



In other words if you think inserts, updates, and deletes are more prominent then go for a normalized table design, else create a flat denormalized database structure.

Below is a simple diagram which shows how the names and address in the left hand side are a simple normalized table and by applying a denormalized structure how we have created a flat table structure.

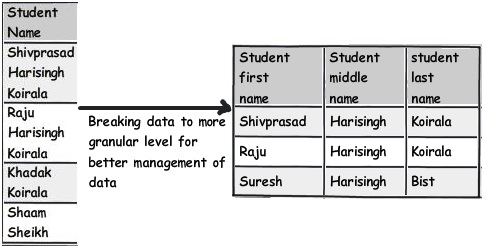


## Rule 2: Break your data into logical pieces, make life simpler

This rule is actually the first rule from 1st normal form. One of the signs of violation of this rule is if your queries are using too many string parsing functions like substring, charindex, etc., then probably this rule needs to be applied.

For instance you can see the below table which has student names; if you ever want to query student names having “Koirala” and not “Harisingh”, you can imagine what kind of a query you will end up with.

So the better approach would be to break this field into further logical pieces so that we can write clean and optimal queries.



## Rule 3: Do not get overdosed with rule 2

Developers are cute creatures. If you tell them this is the way, they keep doing it; well, they overdo it leading to unwanted consequences. This also applies to rule 2 which we just talked above. When you think about decomposing, give a pause and ask yourself, is it needed? As said, the decomposition should be logical.

For instance, you can see the phone number field; it’s rare that you will operate on ISD codes of phone numbers separately (until your application demands it). So it would be a wise decision to just leave it as it can lead to more complications.

## Rule 4: Treat duplicate non-uniform data as your biggest enemy

Focus and refactor duplicate data. My personal worry about duplicate data is not that it takes hard disk space, but the confusion it creates.

For instance, in the below diagram, you can see “5th Standard” and “Fifth standard” means the same. Now you can say the data has come into your system due to bad data entry or poor validation. If you ever want to derive a report, they would show them as different entities, which is very confusing from the end user point of view.

One of the solutions would be to move the data into a different master table altogether and refer them via foreign keys. You can see in the below figure how we have created a new master table called “Standards” and linked the same using a simple foreign key.

## Rule 5: Watch for data separated by separators

The second rule of 1st normal form says avoid repeating groups. One of the examples of repeating groups is explained in the below diagram. If you see the syllabus field closely, in one field we have too much data stuffed. These kinds of fields are termed as “Repeating groups”. If we have to manipulate this data, the query would be complex and also I doubt about the performance of the queries.

These kinds of columns which have data stuffed with separators need special attention and a better approach would be to move those fields to a different table and link them with keys for better management.

So now let’s apply the second rule of 1st normal form: “Avoid repeating groups”. You can see in the above figure I have created a separate syllabus table and then made a many-to-many relationship with the subject table.

With this approach the syllabus field in the main table is no more repeating and has data separators.

## Rule 6: Watch for partial dependencies

Watch for fields which depend partially on primary keys. For instance in the above table we can see the primary key is created on roll number and standard. Now watch the syllabus field closely. The syllabus field is associated with a standard and not with a student directly (roll number).

The syllabus is associated with the standard in which the student is studying and not directly with the student. So if tomorrow we want to update the syllabus we have to update it for each student, which is painstaking and not logical. It makes more sense to move these fields out and associate them with the Standard table.

You can see how we have moved the syllabus field and attached it to the Standards table.

This rule is nothing but the 2nd normal form: “All keys should depend on the full primary key and not partially”.

## Rule 7: Choose derived columns preciously

If you are working on OLTP applications, getting rid of derived columns would be a good thought, unless there is some pressing reason for performance. In case of OLAP where we do a lot of summations, calculations, these kinds of fields are necessary to gain performance.

In the above figure you can see how the average field is dependent on the marks and subject. This is also one form of redundancy. So for such kinds of fields which are derived from other fields, give a thought: are they really necessary?

This rule is also termed as the 3rd normal form: “No column should depend on other non-primary key columns”. My personal thought is do not apply this rule blindly, see the situation; it’s not that redundant data is always bad. If the redundant data is calculative data, see the situation and then decide if you want to implement the 3rd normal form.

## Rule 8: Do not be hard on avoiding redundancy, if performance is the key

Do not make it a strict rule that you will always avoid redundancy. If there is a pressing need for performance think about de-normalization. In normalization, you need to make joins with many tables and in denormalization, the joins reduce and thus increase performance.

## Rule 9: Multidimensional data is a different beast altogether

OLAP projects mostly deal with multidimensional data. For instance you can see the below figure, you would like to get sales per country, customer, and date. In simple words you are looking at sales figures which have three intersections of dimension data.

For such kinds of situations a dimension and fact design is a better approach. In simple words you can create a simple central sales fact table which has the sales amount field and it makes a connection with all dimension tables using a foreign key relationship.

## Rule 10: Centralize name value table design

Many times I have come across name value tables. Name and value tables means it has key and some data associated with the key. For instance in the below figure you can see we have a currency table and a country table. If you watch the data closely they actually only have a key and value.

For such kinds of tables, creating a central table and differentiating the data by using a type field makes more sense.

## Rule 11: For unlimited hierarchical data self-reference PK and FK

Many times we come across data with unlimited parent child hierarchy. For instance consider a multi-level marketing scenario where a sales person can have multiple sales people below them. For such scenarios, using a self-referencing primary key and foreign key will help to achieve the same.

This article is not meant to say that do not follow normal forms, instead do not follow them blindly, look at your project's nature and the type of data you are dealing with first.

Below is a video which explains the three normal forms step by step using a simple school table.

**For further reading do watch the below interview preparation videos and step by step video series.**

[How to Draw an ER Diagram Online | Vertabelo Database Modeler](https://vertabelo.com/blog/draw-er-diagram-online/)

[Top 7 Entity Relationship (ER) Diagram Online Tools | Vertabelo Database Modeler](https://vertabelo.com/blog/er-diagram-tools/)