

Return to [Database Design and Modeling Fundamentals](#)



Database Design and Modeling Fundamentals

Written by **Brent Huscher** on **30 June 2000**

This article covers the basics of database design including normalization, relationships and cardinality. A great tutorial on database design.

Database design and the creation of an entity relationship diagram (also known as an "ERD" or data model) is an important yet sometimes overlooked part of the application development lifecycle. An accurate and up-to-date data model can serve as an important reference tool for DBAs, developers, and other members of a JAD (joint application development) team. The process of creating a data model helps the team uncover additional questions to ask of end users. Effective database design also allows the team to develop applications that perform well from the beginning. By building quality into the project, the team reduces the overall time it takes to complete the project, which in turn reduces project development costs. The central theme behind database design is to "measure twice, cut once".

Effective database designers will keep in mind the principles of normalization while they design a database. Normalization is a database design approach that seeks the following four objectives:

1. minimization of data redundancy,
2. minimization of data restructuring,
3. minimization of I/O by reduction of transaction sizes, and
4. enforcement of referential integrity.

The following concepts and techniques are important to keep in mind when designing an effective database:

1. An entity is a logical collection of things that are relevant to your database. The physical counterpart of an entity is a database table. Name your entities in singular form and in ALL CAPS. For example, an entity that contains data about your company's employees would be named EMPLOYEE.
2. An attribute is a descriptive or quantitative characteristic of an entity. The physical counterpart of an attribute is a database column (or field). Name your attributes in singular form with either Initial Capital Letters or in all lower case. For example, some attribute names for your EMPLOYEE entity might be: EmployeeId (or employee_id) and BirthDate (or birthdate).
3. A primary key is an attribute (or combination of attributes) that uniquely identify each instance of an entity. A primary key cannot be null and the value assigned to a primary key should not change over time. A primary key also needs to be efficient. For example, a primary key that is associated with an INTEGER datatype will be more efficient than one that is associated with a CHAR datatype. Primary keys should also be non-intelligent; that is, their values should be assigned arbitrarily without any hidden meaning. Sometimes none of the attributes of an entity are sufficient to meet the criteria of an effective primary key. In this case the database designer is best served by creating an "artificial" primary key.
4. A relationship is a logical link between two entities. A relationship represents a business rule and can be expressed as a verb phrase. Most relationships between entities are of the "one-to-many" type in which one instance of the parent entity relates to many instances of the child entity. For example, the relationship between EMPLOYEE and STORE_LOCATION would be represented as: one STORE_LOCATION (parent entity) employs many EMPLOYEEs (child entity).
5. The second type of relationship is the "many-to-many" relationship. In a "many-to-many" relationship, many instances of one entity relate to many instances of the other entity. "Many-to-many" relationships need to be resolved in order to avoid data redundancy. "Many-to-many" relationships may be resolved by creating an intermediate entity known as a cross-reference (or XREF) entity. The XREF entity is made up of the primary keys from both of the two original entities. Both of the two original entities become parent entities of the XREF entity. Thus, the "many-to-many" relationship becomes resolved as two "one-to-many" relationships. For example, the "many-to-many" relationship of (many) EMPLOYEEs are assigned (many) TASKs can be resolved by creating a new entity named EMPLOYEE_TASK. This resolves the "many-to-many" relationship by creating two separate "one-to-many" relationships. The two "one-to-many" relationships are EMPLOYEE (parent entity) is assigned EMPLOYEE_TASK (child entity) and TASK (parent entity) is assigned to EMPLOYEE_TASK (child entity).
6. A "foreign key" exists when the primary key of a parent entity exists in a child entity. A foreign key requires that values must be present in the parent entity before like values may be inserted in the child entity. The concept of maintaining foreign keys is known as "referential integrity".
7. Relationships between two entities may be classified as being either "identifying" or "non-identifying". Identifying relationships exist when the primary key of the parent entity is included in the primary key of the child entity. On the other hand, a non-identifying relationship exists when the primary key of the parent entity is included in the child entity but not as part of the child entity's primary

key. In addition, non-identifying relationships may be further classified as being either "mandatory" or "non-mandatory". A mandatory non-identifying relationship exists when the value in the child table cannot be null. On the other hand, a non-mandatory non-identifying relationship exists when the value in the child table can be null.

8. Cardinality helps us further understand the nature of the relationship between the child entity and the parent entity. The cardinality of a relationship may be determined by asking the following question: "How many instances of the child entity relate to each instance of the parent entity?". There are four types of cardinality: (1.) One to zero or more (common cardinality), (2.) One to one or more (P cardinality), (3.) One to zero or one (Z cardinality), and (4.) One to exactly N (N cardinality).

In conclusion, effective database design can help the development team reduce overall development time and costs. Undertaking the process of database design and creating a data model helps the team better understand the user's requirements and thus enables them to build a system that is more reflective of the user's requirements and business rules. The act of performing database design is platform-independent so persons who use database systems other than SQL Server should also be able to benefit from these concepts.