Table of Contents

This document discusses database terms and concepts from a design point of view. This is not a database design class (that is CS 159A); but we cannot talk about tables without talking about design.

We need to talk about collections of tables/relations and the associations/relationships between these tables and part of this is terminology.

1. Attributes and domains

An attribute is a characteristic of an entity that we need to store; an attribute is represented as a column in a table. We will limit the values that can be stored in an attribute. The first way is simply the data type that we use when we define the table. In the vt_staff table, the stf_id is defined as an integer; so we can use a staff id of 1567, but we cannot used a staff id value of 45.67 nor can we store a staff id value of 'Sect'. Look at the vt_staff.stf_job_title attribute; it is limited to the four values that are listed in the definition. (MYSQL does not yet enforce those check rules in the table.) You can find other rules about the allowed data values as you read the Create Table statements. The term domain is sometimes used to describe the legitimate values for an attributes; some people use the term 'type' for this concept.

2. Relations

A **RELATION** is a collection of tuples (the theoretical term for what we commonly call a table row). A relation is represented as a two-dimensional table. A relation has the following characteristics:

- Each relation has a name that is unique within the schema. We cannot create two tables with the same name in the same schema.
- Each row contains information about a single entity instance. We should not have a row in a table that includes information about an client and also information about their animals. These are two separate entities and the data is stored in two tables.
- No two rows in a table are identical. It would not be helpful to have two rows in the client table for the same client. We use a **primary key** to avoid duplicate rows.
- Each column contains attribute data values. A column is defined with a name that represents an attribute we want to store and we should use that column for that purpose.
- All of the values in a column are from the same domain.
- Each column has a name that is unique within the relation. We cannot create two columns in the same table with the same name. We can use the same name for columns in different tables and we often do this for the columns that form the relationships between tables.
- A table has to have at least one column.
- Each cell contains a single data value. If we want to keep salary history for our staff, we would have to set up a new table for this. We should not try to keep salary history as a list of values in the salary attribute.
- The relation may not have repeating columns. Suppose we decided to keep track of our staff phone numbers and we wanted to store multiple phone numbers for each staff person. It would not be appropriate to create columns such as phone_1, phone_2, phone_3 etc.
- The order of the rows has no logical significance. We cannot have any logic in our table concepts that tries to reflect that a row in a table has some meaning because it follows another row. We can display

the rows in a certain order, but we do not store the rows in any particular order. Tables reflect sets which are not ordered.

- The order of the attributes has no logical significance. We are not required to have the attribute for the primary key be the first column in the table, although it often is the first. In fact we cannot meaningfully speak of the first attribute in a relation. We speak of attribute by the name of the attribute. (Tables actually have an ordering to their columns and we do use that in SQL sometimes. An SQL table is not strictly speaking a relation.)

3. Keys

We talk a lot about keys in databases; there are several types of keys.

3.1. Candidate, Primary Keys

A candidate key is an attribute or a collection of attributes which is never null and is always unique within its table. In the vt_staff_pay table, we have two candidate keys, the stf_id and the stf_ssn. The primary key is the key that we chose as the identifier and define as such when we create the table. A table has only one primary key.

A composite key is made up of more than one attribute. The vt_exam_details table has a primary key made up of two attributes (ex_id and line_item.).

The Entity Integrity Rule says that each table must have a primary key, which cannot be null. If this is a composite primary key, then no component of the key can be null.

3.2. Foreign Key

Foreign keys are defined with respect to two tables. The foreign key is the attribute in the child table that has values matching a candidate key's values (normally the primary key) in the parent table. The foreign key must have the same underlying domain as the associated candidate key. The foreign key values must either be null or have values that match existing values in the parent table.

The foreign key and the referenced candidate key must have the same data type. A foreign key can be nullable.

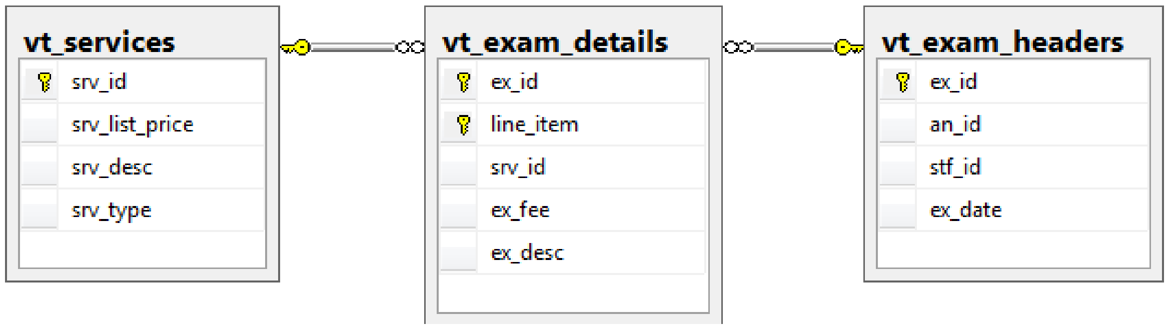
The foreign key and the associated candidate key can have different names.

A row in the parent table can have zero, one, or more associated rows in the child table.

It is possible that the parent table and the child table are actually the same table.

If the primary key of a table is composite, then any related foreign key must also be composite.

Take another look at the table vt_exam_details. The column ex_id refers to the ex_id in the vt_exam_headers table so that we know that this detail row belongs to a specific exam row. The table vt_exam_details also has a column srv_id that refers to the table vt_services; that way we know that the service we have for that detail row is a service the database knows about.



4. Relationships

A Relationship is an association between two or more relations based on data values in common between the tables. Relationships can be classified as One-to-One, One-to-Many, or Many-to-Many.

We have mentioned that we could have a table with data about clients and another table with data about animals. For our database, each client could have multiple animals. The relationship between the clients table and the animals table is One-to-Many. The term "many" does not mean that we must have multiple rows on the many side- only that we could. A one-to-many relationship allows a situation where we could have a row for a client and no animals for that client.

The terms parent and child are often used to describe the tables in a one to many (1:N) relationship. For example, one client may have many animals. In this case, the client table is the parent and the animals table is the child. One animal may be associated with multiple exams. In this relationship, the parent is the vt_animals table and the child is the vt_exam_header table.

We decided to put some the data about our staff in the vt_staff table and part into the vt_staff_pay table for privacy reasons. The relationship between these tables is One-to-One. Each staff person has at one row in the vt_staff table and one row in the vt_staff_pay.

We cannot create a Many-to-Many relationship in our database but logically we do have a Many-to-Many relationship between our entities. Each service that the vet can supply can be used on multiple exams and each exam can include multiple services. (The vt_exam_details table handles that relationship.)

5. Database and table design

Good database design depends on good table design. Good table design reduces redundancy and avoids errors in adding new rows, deleting current rows and updating existing data.

These are some general rules for designing good tables for a relational database. We will formalize these later, when we discuss normalization.

- Store each fact one time only. One purpose of a database is to control redundancy, so you do not want to store facts more than once.
- Each attribute should contain a single value. For example, if you are storing information about exams and an exam includes 10 services, you do not store all the service ids in a single attribute.
- Each attribute should be atomic. Do not have an attribute named Address that contains the street address, city, state, country, and postal code. This should be five separate attributes.
- Each row should have an attribute or combination of attributes that is a unique identifier- a primary key. Naturally occurring attributes, such as an animal name attribute, are not good primary keys because you cannot guarantee that they will be unique.
- The primary key should not be subject to change. For example, a phone number is generally not a good primary key since people change their phone numbers and the phone company can reassign phone numbers to other people.
- Every attribute in a row should contain information about the entity identified by the primary key. For example, a table with information about clients will store the client's name and address values — but not information on the animals that they have. That information will be stored in related tables.
- In general, every table in the database will have a relationship with at least one other table.
- Usually, a well-designed database will have many narrow tightly focused tables- rather than a few tables that try to store a lot of different columns of data.
- Database logic is based on set logic- a set is an unordered collection of items, all of the same type. A well designed table is a set of rows.
- SQL allows you to create tables which have duplicate rows; therefore SQL tables correspond to logical constructs called multi-sets or bags.
- A table is a set of rows; a row is a set of columns; a column is an atomic value which has a data type.
- With object-relational databases, the column value might be an object rather than a simple data value.

The discussion here has focused on the tables for the vets database. It would be a ***Very Good Idea*** if you think about these concepts in terms of the AltgeldMart database tables.