Chapter 2

Relational databases

Relational points of view

Relational databases play a critical role in many important (that is, money-related) computer applications. As is the case whenever enormous amounts of money are at stake, people have spent a huge amount of time and effort building, studying, and refining relational databases. Database researchers usually approach relational databases from one of three points of view.

Relational points of view

The first group approaches the problem from a database-theoretical point of view. These people tend to think in terms of probability, mathematical set theory, and propositional logic.

Relational points of view

The second group approaches the matter from a less formal "just build the database and get it done" point of view. Their terminology tends to be less precise and rigorous but more intuitive.

Relational points of view

The third group tends to think in terms of flat files and the underlying disk structure used to hold data. Though these people are probably in the minority these days, their terms file, record, and field snuck into database nomenclature and stuck.

Tables, rows and columns

Informally you can think of a relational database as a collection of tables, each containing rows and columns.

You can put just about anything in any cell in a worksheet. In contrast, every entry in a particular column of a table is expected to contain the same kind of data. For example, all of the cells in a particular column might contain phone numbers or last names.

Tables, rows and columns

The set of the values that are allowed for a column is called the column’s domain. For example, a column’s domain might be telephone numbers, bank account numbers, snowshoe sizes, or hang glider colors.

Domain is closely related to data type but it’s not quite the same. A column’s data type is the kind of data that the column can hold. The data types that you can use for a column depend on the particular database you are using but typical data types include integer, floating point number (a number with a decimal point), string, and date.

Tables, rows and columns

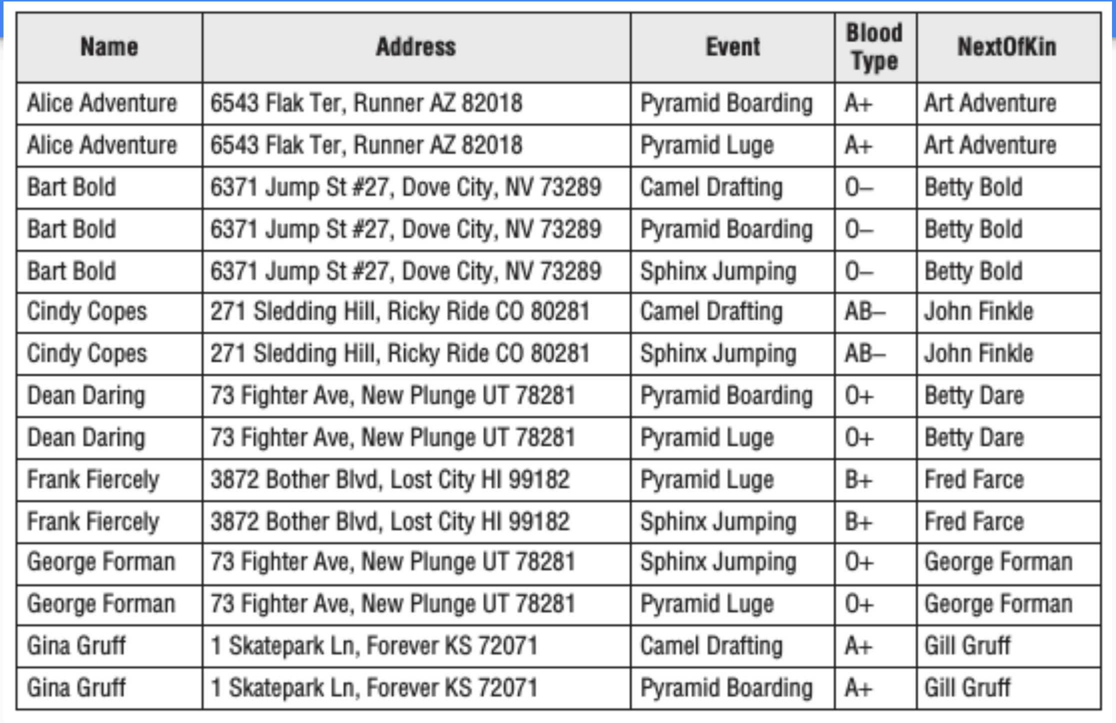
To see the difference between domain and data type, note that street address (323 Relational Rd) and jersey color (red) are both strings. However, the domain for the street address column is valid street addresses, whereas the domain for the jersey color column is colors.

Tables, rows and columns

The rows in a table correspond to column values that are related to each other according to the purpose of the table. For example, suppose you have a Competitors table that contains typical contact information for participants in your First (and probably Last) Annual Extreme Pyramid Sports Championship.

Tables, rows and columns

This table includes columns to hold competitor name, address, event, blood type, and next of kin as shown next.



Name	Address	Event	Blood Type	NextOfKin
Alice Adventure	6543 Flak Ter, Runner AZ 82018	Pyramid Boarding	A+	Art Adventure
Alice Adventure	6543 Flak Ter, Runner AZ 82018	Pyramid Luge	A+	Art Adventure
Bart Bold	6371 Jump St #27, Dove City, NV 73289	Camel Drafting	O–	Betty Bold
Bart Bold	6371 Jump St #27, Dove City, NV 73289	Pyramid Boarding	O–	Betty Bold
Bart Bold	6371 Jump St #27, Dove City, NV 73289	Sphinx Jumping	O–	Betty Bold
Cindy Copes	271 Sledding Hill, Ricky Ride CO 80281	Camel Drafting	AB–	John Finkle
Cindy Copes	271 Sledding Hill, Ricky Ride CO 80281	Sphinx Jumping	AB–	John Finkle
Dean Daring	73 Fighter Ave, New Plunge UT 78281	Pyramid Boarding	O+	Betty Dare
Dean Daring	73 Fighter Ave, New Plunge UT 78281	Pyramid Luge	O+	Betty Dare
Frank Fiercely	3872 Bother Blvd, Lost City HI 99182	Pyramid Luge	B+	Fred Farce
Frank Fiercely	3872 Bother Blvd, Lost City HI 99182	Sphinx Jumping	B+	Fred Farce
George Forman	73 Fighter Ave, New Plunge UT 78281	Sphinx Jumping	O+	George Forman
George Forman	73 Fighter Ave, New Plunge UT 78281	Pyramid Luge	O+	George Forman
Gina Gruff	1 Skatepark Ln, Forever KS 72071	Camel Drafting	A+	Gill Gruff
Gina Gruff	1 Skatepark Ln, Forever KS 72071	Pyramid Boarding	A+	Gill Gruff

Relations, Attributes, and Tuples

The values in a row are related by the fact that they apply to a particular person. Because of this fact, the formal term for a table is a relation. This can cause some confusion because the word “relation” is also used informally to describe a relationship between two tables.

The formal term for a column is an attribute or data element.

The formal term for a row is a tuple (rhymes with “scruple”). This almost makes sense if you think of a two-attribute relation as holding data pairs, a three-attribute relation as holding value triples, and a four-attribute relation as holding data quadruples. Beyond four items, mathematicians would say 5-tuple, 6-tuple, and so forth, hence the name tuple.

Relations, Attributes, and Tuples

Don’t confuse the formal term relation (meaning table) with the more general and less precise use of the term that means “related to” as in “these fields form a relation between these two tables” (or “that psycho is no relation of mine”). Similarly, don’t confuse the formal term attribute with the less precise use that means “feature of” as in “this field has the ‘required’ attribute” (or “don’t attribute that comment to me!”). I doubt you’ll confuse the term tuple with anything — it’s probably confusing enough all by itself.

Relations, Attributes, and Tuples

Theoretically a relation does not impose any ordering on the tuples that it contains nor does it give an ordering to its attributes. Generally the orderings don’t matter to mathematical database theory. In practice, however, database applications usually sort the records selected from a table in some manner to make it easier for the user to understand the results.

Keys

Relational database terminology includes an abundance of different flavors of keys. In the loosest sense, a key is a combination of one or more columns that you use to find rows in a table. For example, a Customers table might use CustomerID to find customers. If you know a customer’s ID, you can quickly find that customer’s record in the table. (In fact, many ID numbers, such as employee IDs, student IDs, driver’s licenses, and so forth, are invented just to make searching in database tables easier. My library card certainly doesn’t include a 10-character ID number for my convenience.)

The more formal relational vocabulary includes several other more precise definitions of keys.

Keys

In general, a key is a set of one or more columns in the table that have certain properties. A compound key or composite key is a key that includes more than one column. For example, you might use the combination of FirstName and LastName to look up customers.

A superkey is a set of one or more columns in a table for which no two rows can have the exact same values.

Because no two rows in the table have the same values for a superkey, a superkey can uniquely identify a particular row in the table. In other words, a program could use a superkey to find any particular record.

Keys

A candidate key is a minimal superkey. That means if you remove any of the columns from the superkey, it won’t be a superkey anymore.

For example, you already know that Name/Address/Event is a superkey for the Competitors table. If you remove Event from the superkey, Name/Address is not a superkey because everyone in the table is participating in multiple events so they have more than one record with the same name and address.

Keys

If you remove Name, Address/Event is not a superkey because Dean Daring and his roommate George Foreman share the same address and are both signed up for Pyramid Luge. (They also have the same blood type. They became friends and decided to become roommates when Dean donated blood for George after a particularly flamboyant skateboarding accident.)

Finally if you remove Address, Name/Event is still a superkey. That means Name/Address/Event is not a candidate key because it is not minimal. However, Name/Event is a candidate key because no two rows have the same Name/Event values and you can easily see neither Name nor Event is a superkey, so the pair is minimal.

Keys

A unique key is a superkey that is used to uniquely identify the rows in a table. The difference between a unique key and any other candidate key is in how it is used. A candidate key could be used to identify rows if you wanted it to, but a unique key is used to constrain the data. In this example, if you make Name/Event be a unique key, the database will not allow you to add two rows with the same Name and Event values. A unique key is an implementation issue, not a more theoretical concept like a candidate key is.

Keys

A primary key is a superkey that is actually used to uniquely identify or find the rows in a table. A table can have only one primary key (hence the name “primary”). Again, this is more of an implementation issue than a theoretical concern. Database products generally take special action to make finding records based on their primary keys faster than finding records based on other keys.

An alternate key is a candidate key that is not the primary key. Some also call this a secondary key, although others use the term secondary key to mean any set of fields used to locate records even if the fields don’t define unique values.

Keys

The following list briefly summarizes the different flavors:

- ❑ Compound key or composite key: A key that includes more than one field.
- ❑ Superkey: A set of columns for which no two rows can have the exact same values.
- ❑ Candidate key: A minimal superkey.
- ❑ Unique key: A superkey used to require uniqueness by the database.
- ❑ Primary key: A unique key that is used to quickly locate records by the database.
- ❑ Alternate key: A candidate key that is not the primary key.
- ❑ Secondary key: A key used to look up records but that may not guarantee uniqueness.

Indexes

An index is a database structure that makes it quicker and easier to find records based on the values in one or more fields. Indexes are not the same as keys, although the two are related closely enough that many developers confuse the two and use the terms interchangeably.

For example, suppose you have a Customers table that holds customer information: name, address, phone number, Swiss bank account number, and so forth. The table also contains a CustomerId field that it uses as its primary key.

Indexes

Unfortunately customers usually don’t remember their customer IDs, so you need to be able to look them up by name or phone number. If you make Name and PhoneNumber be two different keys, you can quickly locate a customer’s record in three ways: by customer ID, by name, and by phone number.

Constraints

As you might guess from the name, a constraint places restrictions on the data allowed in a table. In formal database theory, constraints are not considered part of the database. However, in practice constraints play such a critical role in managing the data properly that they are informally considered part of the database. (Besides, the database product enforces them!)

The following slides describe some of the kinds of constraints that you can place on the fields in a table.

Constraints: basic constraints

These types of constraints restrict the values that you can enter into a field. They help define the field’s domain so they are called domain constraints. Some database products allow you to define more complex domain constraints, often by using check constraints.

Constraints: primary constraints

By definition, no two records can have identical values for the fields that define the table's primary key.That greatly constrains the data.

In more formal terms, this type of constraint is called entity integrity. It simply means that no two records are exact duplicates (which is true if the fields in their primary keys are not duplicates) and that all of the fields that make up the primary key have non-null values.

Constraints: unique constraints

A unique constraint requires that the values in one or more fields be unique. Note that it only makes sense to place a uniqueness constraint on a superkey. Recall that a superkey is a group of one or more fields that cannot contain duplicate values. It wouldn't make sense to place a uniqueness constraint on fields that can validly contain duplicated values.

For example, it would be silly to place a uniqueness constraint on a Gender field.

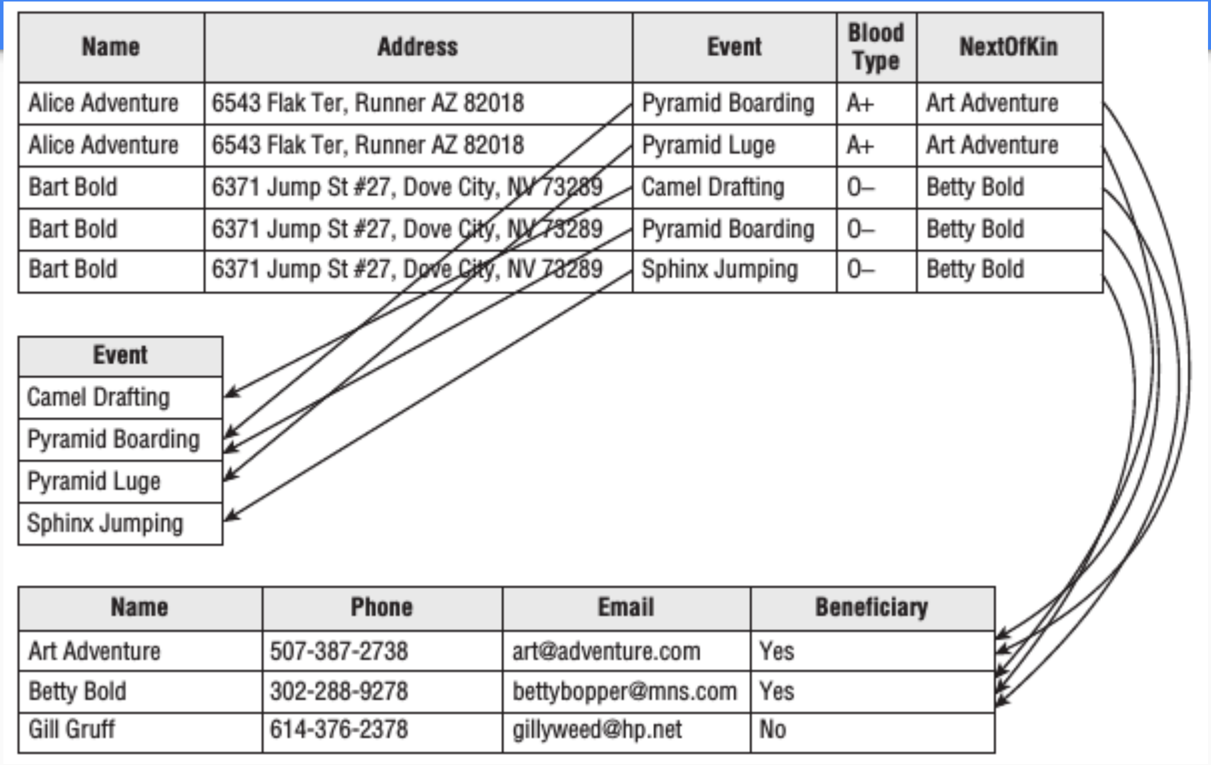
Constraints: foreign key constraints

A foreign key is not quite the same kind of key defined previously. Instead of defining fields that you use to locate records, a foreign key refers to a key in another (foreign) table. The database uses it to locate records in the other table but you don't. Because it defines a reference from one table to another, this kind of constraint is also called a referential integrity constraint.

Constraints: foreign key constraints

A foreign key constraint requires that a record’s values in one or more fields in one table (the referencing table) must match the values in another table (the foreign or referenced table). The fields in the referenced table must form a candidate key in that table. Usually they are that table’s primary key, and most database products try to use the foreign table’s primary key by default when you make a foreign key constraint.

Constraints: foreign key constraints



Name	Address	Event	Blood Type	NextOfKin
Alice Adventure	6543 Flak Ter, Runner AZ 82018	Pyramid Boarding	A+	Art Adventure
Alice Adventure	6543 Flak Ter, Runner AZ 82018	Pyramid Luge	A+	Art Adventure
Bart Bold	6371 Jump St #27, Dove City, NV 73289	Camel Drafting	O–	Betty Bold
Bart Bold	6371 Jump St #27, Dove City, NV 73289	Pyramid Boarding	O–	Betty Bold
Bart Bold	6371 Jump St #27, Dove City, NV 73289	Sphinx Jumping	O–	Betty Bold

Event
Camel Drafting
Pyramid Boarding
Pyramid Luge
Sphinx Jumping

Name	Phone	Email	Beneficiary
Art Adventure	507-387-2738	art@adventure.com	Yes
Betty Bold	302-288-9278	bettybopper@mns.com	Yes
Gill Gruff	614-376-2378	gillyweed@hp.net	No

Constraints: foreign key constraints

Foreign keys define associations between tables that are sometimes called relations, relationships, or links between the tables. The fact that the formal database vocabulary uses the word relation to mean table sometimes leads to confusion. Fortunately, the formal and informal database people usually get invited to different parties so the terms usually don’t collide in the same conversation.

Database Operations

Eight operations were originally defined for relational databases and they form the core of modern database operations. The following list describes those original operations:

1. Selection: This selects some or all of the records in a table. For example, you might want to select only the Competitors records where Event is Pyramid Luge so you can know who to expect for that event (and how many ambulances to have standing by).

Database Operations

2. Projection: This drops columns from a table (or selection). For example, when you make your list of Pyramid Luge competitors you may only want to list their names and not their addresses, blood types, events (which you know is Pyramid Luge anyway), or next of kin.
3. Union: This combines tables with similar columns and removes duplicates. For example, suppose you have another table named FormerCompetitors that contains data for people who participated in previous years' competitions. Some of these people are competing this year and some are not. You could use the union operator to build a list of everyone in either table.

Database Operations

4. Intersection: This finds the records that are the same in two tables. The intersection of the FormerCompetitors and Competitors tables would list those few who competed in previous years and who survived to compete again this year (the slow learners).
5. Difference: This selects the records in one table that are not in a second table. For example, the difference between FormerCompetitors and Competitors would give you a list of those who competed in previous years but who are not competing this year (so you can email them and ask them what the problem is).

Database Operations

6. Cartesian Product: This creates a new table containing every record in a first table combined with every record in a second table. For example, if one table contains values 1, 2, 3 and a second table contains values A, B, C, then their Cartesian product contains the values 1/A, 1/B, 1/C, 2/A, 2/B, 2/C, 3/A, 3/B, and 3/C.
7. Join: This is similar to a Cartesian product except records in one table are paired only with those in the second table if they meet some condition. For example, you might join the Competitors records with the NextOfKin records where a Competitors record's NextOfKin value matches the NextOfKin record's Name value. In this example, that gives you a list of the competitors together with their corresponding next of kin data.

Database Operations

8. Divide: This operation is the opposite of the Cartesian product. It uses one table to partition the records in another table. It finds all of the field values in one table that are associated with every value in another table. For example, if the first table contains the values 1/A, 1/B, 1/C, 2/A, 2/B, 2/C, 3/A, 3/B, and 3/C and a second table contains the values 1, 2, 3, then the first divided by the second gives A, B, C. (Don't worry, I think it's pretty weird and confusing, too, so it won't be on the final exam. Probably.)