DBS211

Week 2 - Relational Database Design

Table of Contents

Reading Materials

- Text - Chapter 4 (Types of Database Models)
- Text - Chapter 7 (The Relational Data Model)
- Text - Chapter 8 (Entity Relationship Data Model)

Welcome to Week 2

Welcome to Week 2 of DBS211. This week continues where we left off last week with respect to defining what a database is, the benefits of databases and continues to incestigate best design practices. Through the learning of several key concepts, the learner will be able to:

- describe what a relational model is and how it applies to databases,
- list and define the 3 major types of relationships in relational database design,
- explain the concept of referential integrity and elaborate on why it is a crucial component of relational databases
- list and describe the 3 basic table types in simplistic databases
- define the 5 major types of keys in relational databases and when it is appropriate to use each one.

Continuing from last week where we learned a database is:

- a centralized repository of data,
- maintained in real-time
- allows multiple userrs to work concurrently.
- and allows for ad-hoc queries for data extraction for informed decision making

we will dive into the area of data modelling.

Relational Models

Data Modeling is simple a method of describing data or information which includes:

- the structure and format of data storage
- operations performed on data through selection (queries) and modification (insertion, updating and deleting)
- the limitations place on the data stored through a series of defined constraints or enforced rules.

A relational data model further extends the data model by defining relationships between various data points and separates data to eliminate inefficiencies and non-functional dependencies. By structuring the data, efficient data access and manipulation can be achieved while simultaneously working towards minimizing errors in the data due to anomalies and human error.

The Basics of the Relational Model

In the relational model, data is represented by a series of two dimensional tables (also called relations). Every **row** in a table represents a single instance of the entity in the table and every **column** represents an attribute (or property) of that entity. Each column, for each row, can only have one value in it (i.e. the values must be **atomic**).

For Example: Each row in the following table represents a department and every column represents an attribute of that department.

Departments			
Department_ID	**Department_Name**	**Manager_ID**	**Location_ID**
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	Information Technology	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting	(null)	1700

Therefore, department_id, department_name, manager_id, and location_id are **attributes** in the table of departments.

Schema Notation

A Schema represents the name of a relation (table) and its' attributes and can be written as follows:

```
DEPARTMENT ( department_id, department_name, manager_id, location_id )
```

Keys

One of the most important functions of a database is the process of querying and manipulating data. In order to perform this function, it is extremely important that the system knows exactly on what data it is to perform an action. In the departments table above, if we were to change the name or location of a department, we would update the appropriate column in the one and only one row that needs to be updated. Therefore, it is crucial that there is a way to uniquely identify each and every row, or instance, in a table.

Employees				
First Name	**Last Name**	**Job Title**	**Date of Birth**	**SIN**
Steven	King	President	May 16, 1972	495777777
Trenna	Rajs	Sales Manager	Oct 10, 1983	478333333
Ellen	Abel	Clerk	Feb 21, 1995	123444444
John	Smith	Mechanic	Mar 8, 1992	456555555
Jane	Smith	Receptionist	Aug 17, 1998	788666666

In an employees table, first and last name may be a good way to find the person you are looking for in small or medium size businesses. But think about a business with 10,000 employees, or a college with 50,000 students, and the potential that two individuals would have the same first and last name. The important thing is not whether there are two people with the same name, but is it possible. If the answer is "yes", then that attribute is not good enough to uniquely identify the exact employee you want to.

Therefore, there must be **a single attribute or a combination of attributes that will uniquely identify a specific instance, or row**, in a table.

So let us then consider using first name, last name and date of birth. This surely will identify a single employee, and in 99.999% of cases it will. BUT, there is still a chance that it might not, and therefore this is not good enough. At college, each student is issued a unique studentID value, not to downplay the importance of their names, but to guarantee a unique way to identify each and every student.

In the employees table above, the SIN (Social Insurance Number) is absolutely unique per person and therefore would be a good candidate for uniquely identifying instances of employees. Because of privacy issues, it is unlikely to be the final decision, but for now it will be considered.

When you go home after school or work, you use your keys to open your house door. You expect that your key is unique for your door and that no-one unauthorized will have a key to get in. These unique features are also called keys in database.

At this time, we will consider the following 5 different types of keys:

Types of Keys	
Candidate Key	an attribute, or combination of attributes, that could potentially be used to uniquely identify an single instance, or row, in a table
Primary Key	an attribute, or combination of attributes, that has been chosen to uniquely identify a single instance, or row, in a table.
Composite Key	A composite key is the case where multiple attributes make up a candidate or primary key. The uniqueness comes from the unique combination of values. Each single attribute can have repeat values, but there can be no repeats in the combination of values from all attributes that are part of the composite key.
Surrogate Key	Is an artificially added field, or attribute, that is added to replace the existing fields from being the primary key. This occurs most often in 2 cases: 1) when multiple composite fields are chosen, in order to simplify the interaction with the database and 2) when there is not a field, or attribute, available to be a Primary Key
Foreign Key	A foreign key is a constraint applied to a table defining the relationship between two tables (see next section).

This brings us to one of our fundamental rules of relational database design:

> Every table must have one, and only one, primary key. Remember that a composite key is a single key made up of more than one field.

Relationships

When multiple tables, or entities, are being used, there is often attributes in each table the relates the two tables together.

Players			
PlayerID	**FirstName**	**LastName**	**TeamID**
123	Billl	Marlow	**22**
456	Robert	MacDonald	**22**
78	John	Smith	24

Teams		
TeamID	**TeamName**	**ShortColour**
22	Hornets	Yellow
23	Shooters	Grey
24	Slowpokes	Blue

Looking at the above two tables, it is clear that Bill and Robert both play on the Hornets team. This information os obtained by seeing that there is a relationship between Players and Teams defined by the teamID attribute. This becomes the relationship between the two tables and is further defined by a foreign key constraint. The playerID field in the Players table and the TeamID field in the Teams table are underlined to indicate that they are defined as the primary Key for their respective tables.

Relationship Types

In the above tables, the way they are designed, and knowing that each row is uniquely identified by its' primary key and that values must be atomic, each player can only play on one team. This can be determined just because of the fact that the teamID field is in the player table and must be atomic. If you visit the relationship in the other direction, it is also clear that each team can be referenced multiple times, in other words, teams can have no players, one player, or more than one player. This is the most common type of relationship in relational databases and is called a 1-to-many relationship. In this relationship, the Teams table is known as the Parent Table and the Players Table is

the Child Table. A foreign key constraint would be created on the child players table referencing the the relationship to the parent tables teamID field.

Another way to look at this would be to reword the consideration like the following:

- FOR EACH row in the players table, how many teams can there be? (answer: none or one)
- FOR EACH row in the teams table, how many players can there be? (answer: none, one or more than one)

This clearly defines a 1-to-many relationship.

There are three types of relationships:

1-to-many (1-∞)	These relationships are the most common type of relationship and means that in one direction, there can be only one value per row of onethe child table, but in the other direction, there can many rows in the child table that reference a single row in the parent table.
1-to-1 (1-1)	These relationships are used to reference multiple attributes of a row, determined by the primary key, where different attributes have varying levels of completeness (i..e they have a value or not). When investigating the relationship from both directions, there can not be more than one related row in either table. An example follows.
many-to-many (∞-∞)	Many-to-Many, also noted M:M or M:N, relationships are a special circumstance where there could be more than one reference to each row of the other table, in both directions. If the league was to decide that a player could play on more than one team, then the design of the tables has to change, but then we would have the scenario where each team could have more than one player and each player could play on more than one team.

Additional Examples

1-to-Many Relationships

- At college, each section of a course is taught by a single professor, but a professor can teach multiple course sections (from the same course or different courses (ignore the summer semester split)

- Each office can only be located inside a single building, but each building can have many offices
- Each gas station can only be affiliated with one oil company (example: Shell, Petro Canada, etc) but each oil company is affiliated with many gas stations

1-to-1 Relationships

- Each employee is a single person and each person can only be a single employee, but might not be an employee
- Each person may or may not be a customer, and each custome can only be a single person
- Each restaurant in Toronto can only be a single licensed business, where each licensed business in Toronto may be a single restaurant, or may not be a restaurant, but can not more than one restaurant.

Many-to-Many Relationships

- Each student at college can take more than one course, and each course at a college contains more than one student
- Each player may play on more than one team, and each team has more than one player
- Each shopper at a store can buy more than one product, and each product can be sold to more than one shopp.

a further, and very important, note regarding many-to-many relationships is that they can not physically be created in a database directly between 2 tables. Once a field is placed in a table, the value must be atomic and the primary key prevents duplicates. Therefore, in order to "simulate" the required relationship, a third table, called or bridge or junction table, is added between the two original tables with opposite 1-to-many relationships between each original table and the new table. (more on this later).

Referential Integrity

In order to help prevent errors in the data, we rely on the concept of referential integrity to be enforced in a relational database. Referential Integrity ensures that values entered into child table attributes already exist in the parent table. Additionally, referential integrity prevents records in a parent table to be deleted if it is currently being referenced from a child table.

For example:

Players

PlayerID	FirstName	LastName	TeamID
123	Billl	Marlow	22
456	Robert	MacDonald	22
78	John	Smith	24

Teams

TeamID	TeamName	ShortColour
22	Hornets	Yellow
23	Shooters	Grey
24	Slowpokes	Blue

Without referential integrity, there is nothing stopping a person from entering a 26 under teamID in the players table, even though teamid 26 does not exist in the Teams table. This would be referred to as an orphan record, meaning a child value does not have an associated parent. The same would be true if someone was to delete team 22 from the Teams table. Players 123 and 456 would still reference team 22, even though it no longer exists. Referential Integrity catches these potential errors and prevents the changes from being made. i.e. you will only be able to enter a teamID of 22, 23, or 24 in the teamID attribute of the players table and teams 22 and 24 can not be deleted from the teams table while players are related to those teams. Note: team 23 can be deleted without issue as there are currently no players referring to it.

Cascading

In relational databases, in order to help automate the process and simplify the stpes needed to allow these changes to occur, referential integrity has a feature called cascading. There are several types of cascading, but the two most common ones are `cascade updates` and `cascade deletes`. Cascading is both very powerful, but also very dangerous and should only be implemented by experienced database designers.

Cascade Updates - If the primary key of the parent record changes for any reason, then the child records that reference the parent value would be automatically updated to match the change. For example: if for some reason, team 22 were to change to team 28, then players 123 and 456 would be automatically be updated to now play on team 28, rather than 22. This prevents these records becoming orphaned by the parental changes. This feature is generally fairly safe to include when the parent value is a primary key key made up of a single field, as the uniqueness feature of a primary key would maintain the unique reference between tables, even if the value were to change.

However, in cases of a composite primary key, the uniqueness comes only from the combination of values and therefore it is possible that records could overlap and mixed together where they should not be.

Cascade Deletes - Cascade deletes are the most dangerous version of cascading. If a parent record is deleted, then all child records that refer to the parent record will also be deleted. So if the Hornets team were to be deleted, and cascade deletes were turned on, then both players 123 and 456 would also be deleted resulting in a loss of data (the player names).

If there are multiple relationships in a database, one relationship can not override another. For example: if player 123 was listed in the coaches table as a coach of another team, then there are 2 scenarios.

1. If cascade deletes was allowed for that relationship, then both player 123 and the associated coaching position would also be deleted. i.e. by deleting a team, you deleted all the players and the coach position 2 tables away.
2. If cascade deletes in not allowed for that relationship, then the coaching position would prevent player 123 from being deleted, and because player 123 can not be deleted, team 22 (the Hornets) also can not be deleted, regardless of cascade delets enabled on that relationship. What do you think would happen to player 456 in this case, assuming that player has no further references in any other table?

Table Types

The types of tables in a relational database are not typically formally referenced, but it is very important to understand each tables role in the larger database design. Understanding these table types will often make review a previously created database design and also assist in adding additional tables to a database design for reasons other than data storage.

There are several types of tables, but there are 4 types that cover the majority of tables in common databases.

Data Table	The primary purpose of this table is to store raw data for later querying, analyzing and manipulating. The majority of tables in a database are of this type. *Examples: students, employees, orders, products*
Lookup Table	The primary purpose of a lookup table is to centralize data to avoid repeated data groups. A common use of a lookup table is to populate a dropdown list on a website or user interface., Often lookup tables are parent tables in

	relationships. *Examples: Provinces, Countries, Colours, Manufacturers, Gender, Transmission Type*
Junction/Bridge Table	A third table created between two tables to simulate a many-to-many relationship through two opposite 1-to-many relationships. Junction or Bridge tables can often contain data as well, based on the dependencies between the data and the two parent primary keys.
Temporary Table	A table that has limited lifetime in a database. Often these table lack referential integrity, are not manually manipulated, break several fundamental database design rules, but has a very distinct purpose in a database. These purposes could include: - moving data from one table to another or importing data from an external source - migrating an old database design to a new database design - the static storage of query results to minimize repeated processing required for complex calculations. (Often used for feeding high traffic websites or mobile applications)

Summary

coming soon....