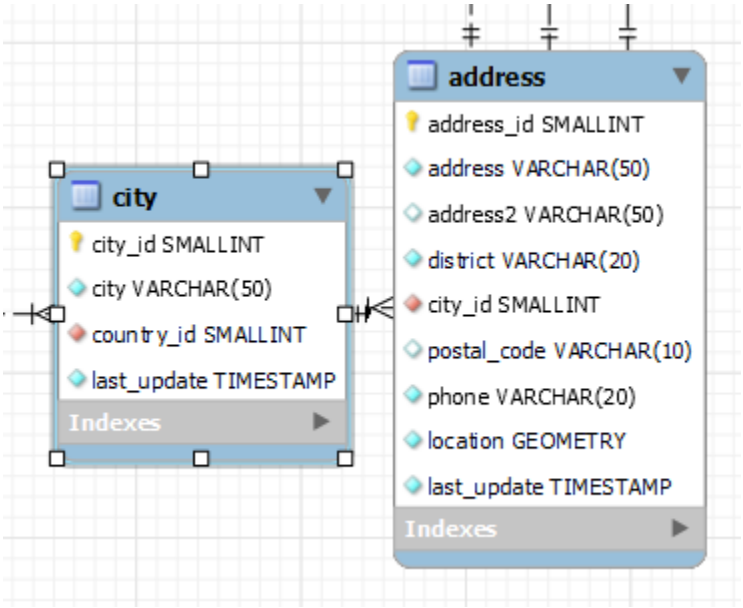
The Function of SQL Joins to Avoid Redundant Data

Ben Jacobs
10/12/24

One of the biggest advantages of a database is the ability to store data in a meaningful format. Reducing redundancy helps the data be more human-understandable, prevents errors storing and retrieving data, and helps us avoid ambiguity through clearly defining what information each table in a database should hold. The way we do this is through putting information in separate tables, or entities. Storing information this way does raise the question though- if we want to find information based on certain parameters stored in tables other than the parameter in question, how can we use SQL to access that data for that filter?

This is one of the core uses of SQL Joins. SQL Joins allow us to specify how and where certain tables should interact when writing a query to draw information in a manner that might pertain to more than one table. Let's use some tables from the sakila database for an example:



Suppose we want to see cities and their addresses in one query. We can use a basic inner join to select shared data from each of the tables. We will join on the primary key city_id in the city table, and the foreign key of city_id in the address table.

```
SELECT c.city_id, a.address
FROM city c
JOIN address a ON c.city_id = a.city_id
```

Other kinds of joins, like left, right, full, and cross joins all allow queries to work with data from more than one table, but select different amounts of the data. For example, a left join will

select all of the first specified table's information, but only shared rows from the table specified second. Cross joins are useful for getting all combinations of two tables, but can return a high number of rows.

It is noteworthy that aliases help make the queries more readable and compact. We can specify a letter or several letters as a shorthand for a given table, and use that alias in other places the table name would normally go. They also allow us to refer to the same table by different names, which is very useful when doing self joins. Here is a query to demonstrate a self join, a type of query that allows us to compare information on the same table as if it were two separate tables. Here's an example using our above tables:

```
SELECT a1.address_id AS address1_id, a1.address AS address1,
     a2.address_id AS address2_id, a2.address AS address2,
     a1.postal_code
FROM address a1
JOIN address a2 ON a1.postal_code = a2.postal_code
                AND a1.address_id <> a2.address_id
ORDER BY a1.postal_code, a1.address_id, a2.address_id;
```

This code uses a self join to find addresses in the address table that share a postal code. A potential use of this technology would be finding all addresses in a certain postal code to send promotions to.

Using these tools, we can effectively query and work with data from multiple tables, avoiding the problems that redundancy can cause.