### Chapter 3

### RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS)

### Introduction

### RDBMS Concepts

1. **Database Keys**

### 3.1 Introduction

### A relational database management system (RDBMS) is a database engine/system based on the relational model specified by Edgar F. Codd--the father of modern relational database design--in 1970.

Most modern commercial and open-source database applications are relational in nature. The most important relational database features include an ability to use tables for data storage while maintaining and enforcing certain data relationships.

In 1970, Edgar F. Codd, a British computer scientist with IBM, published “A Relational Model of Data for Large Shared Data Banks.” At the time, the renowned paper attracted little interest, and few understood how Codd’s groundbreaking work would define the basic rules for relational data storage, which can be simplified as:

1. Data must be stored and presented as relations, i.e., tables that have relationships with each other, e.g., primary/foreign keys.
2. To manipulate the data stored in tables, a system should provide relational operators - code that enables the relationship to be tested between two entities.

Codd later published another paper that outlined the 12 rules that all databases must follow to qualify as relational. Many modern database systems do not follow all 12 rules, but these systems are considered relational because they conform to at least two of the 12rules.

Most modern commercial and open-source database systems are relational in nature and include well-known applications, e.g., Oracle DB (Oracle Corporation); SQL Server (Microsoft) and MySQL and Postgres (open source).

### 3.2 RDBMS Concepts

 **RDBMS** is used to manage Relational database. **Relational database** is a collection of organized set of tables from which data can be accessed easily. Relational Database is most commonly used database. It consists of number of tables and each table has its own primary key.

#### 3.2.1 What is Table ?

In Relational database, a **table** is a collection of data elements organized in terms of rows and columns. A table is also considered as convenient representation of **relations**. But a table can have duplicate tuples while a true **relation** cannot have duplicate tuples. Table is the most simplest form of data storage. Below is an example of Employee table.

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Name** | **Age** | **Salary** |
| 1 | Adam | 34 | 13000 |
| 2 | Alex | 28 | 15000 |
| 3 | Stuart | 20 | 18000 |
| 4 | Ross | 42 | 19020 |

#### 3.2.2 What is a Record ?

A single entry in a table is called a **Record** or **Row**. A **Record** in a table represents set of related data. For example, the above **Employee** table has 4 records. Following is an example of single record.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | Adam | 34 | 13000 |

#### 3.2.3 What is Field ?

A table consists of several records(row), each record can be broken into several smaller entities known as **Fields**. The above **Employee** table consist of four fields, **ID**, **Name**, **Age** and **Salary**.

#### 3.2.4 What is a Column ?

In **Relational** table, a column is a set of value of a particular type. The term **Attribute** is also used to represent a column. For example, in Employee table, Name is a column that represent names of employee.

|  |
| --- |
| **Name** |
| Adam |
| Alex |
| Stuart |
| Ross |

**3.3 Database Keys**

**3.3.1 Introduction**

For the purposes of clarity we will refer to keys in terms of RDBMS tables but the same definition, principle and naming applies equally to Entity Modeling and Normalization.

Keys are, as their name suggests, a key part of a relational database and a vital part of the structure of a table. They ensure each record within a table can be uniquely identified by one or a combination of fields within the table. They help enforce integrity and help identify the relationship between tables. There are three main types of keys, candidate keys, primary keys and foreign keys. There is also an alternative key or secondary key that can be used, as the name suggests, as a secondary or alternative key to the primary key

**3.3.2 Super Key**

A Super key is any combination of fields within a table that uniquely identifies each record within that table.

**3.3.3 Candidate Key**

A candidate is a subset of a super key. A candidate key is a single field or the least combination of fields that uniquely identifies each record in the table. The least combination of fields distinguishes a candidate key from a super key. Every table must have at least one candidate key but at the same time can have several.

**Candidate Keys**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student Id** | **firstName** | **lastName** | **courseId** |
| L0002345 | Jim  | Black  | C002 |
| L0001254 | James  | Harradine  | A004  |
| L0002349 | Amanda  | Holland  | C002 |
| L0001198 | Simon | McCloud | S042 |
| L0023487 | Peter  | Murray  | P301 |
| L0018453 | Anne | Norris | S042 |

As an example we might have a student\_id that uniquely identifies the students in a student table. This would be a candidate key. But in the same table we might have the student’s first name and last name that also, when combined, uniquely identify the student in a student table. These would both be candidate keys.

In order to be eligible for a candidate key it must pass certain criteria.

* It must contain unique values
* It must not contain null values
* It contains the minimum number of fields to ensure uniqueness
* It must uniquely identify each record in the table

Once your candidate keys have been identified you can now select one to be your primary key

**3.3.4 Primary Key**

A primary key is a candidate key that is most appropriate to be the main reference key for the table. As its name suggests, it is the primary key of reference for the table and is used throughout the database to help establish relationships with other tables. As with any candidate key the primary key must contain unique values, must never be null and uniquely identify each record in the table.

As an example, a student id might be a primary key in a student table, a department code in a table of all departments in an organisation. This module has the code DH3D 35 that is no doubt used in a database somewhere to identify RDBMS as a unit in a table of modules. In the table below we have selected the candidate key student\_id to be our most appropriate primary key.

**primary key**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student Id** | **firstName** | **lastName** | **courseId** |
| L0002345 | Jim  | Black  | C002 |
| L0001254 | James  | Harradine  | A004  |
| L0002349 | Amanda  | Holland  | C002 |
| L0001198 | Simon | McCloud | S042 |
| L0023487 | Peter  | Murray  | P301 |
| L0018453 | Anne | Norris | S042 |

Primary keys are mandatory for every table each record must have a value for its primary key. When choosing a primary key from the pool of candidate keys always choose a single simple key over a composite key.

**3.3.5 Foreign Key**

A foreign key is generally a primary key from one table that appears as a field in another where the first table has a relationship to the second. In other words, if we had a table A with a primary key X that linked to a table B where X was a field in B, then X would be a foreign key in B.

An example might be a student table that contains the course\_id the student is attending. Another table lists the courses on offer with course\_id being the primary key. The 2 tables are linked through course\_id and as such course\_id would be a foreign key in the student table.

|  |  |  |  |
| --- | --- | --- | --- |
| **Student Id** | **firstName** | **lastName** | **courseId** |
| L0002345 | Jim  | Black  | C002 |
| L0001254 | James  | Harradine  | A004  |
| L0002349 | Amanda  | Holland  | C002 |
| L0001198 | Simon | McCloud | S042 |

**foreign keys**

**Relationship**

**Primary keys**

|  |  |
| --- | --- |
| **courseld** | **courseName** |
| A004 | Accounts  |
| C002 | Computing  |
| P301 | History  |
| S042 | Short course  |

 **3.3.6 Secondary Key or Alternative Key**

A table may have one or more choices for the primary key. Collectively these are known as candidate keys as discuss earlier. One is selected as the primary key. Those not selected are known as secondary keys or alternative keys.

For example in the table showing candidate keys above we identified two candidate keys, studentId and firstName + lastName. The studentId would be the most appropriate for a primary key leaving the other candidate key as secondary or alternative key. It should be noted for the other key to be candidate keys, we are assuming you will never have a person with the same first and last name combination. As this is unlikely we might consider fistName+lastName to be a suspect candidate key as it would be restrictive of the data you might enter. It would seem a shame to not allow John Smith onto a course just because there was already another John Smith.

**3.3.7 Simple Key**

Any of the keys described before (ie primary, secondary or foreign) may comprise one or more fields, for example if firstName and lastName was our key this would be a key of two fields where as studentId is only one. A simple key consists of a single field to uniquely identify a record. In addition the field in itself cannot be broken down into other fields, for example, studentId, which uniquely identifies a particular student, is a single field and therefore is a simple key. No two students would have the same student number.

**3.3.8 Compound Key**

A compound key consists of more than one field to uniquely identify a record. A compound key is distinguished from a composite key because each field, which makes up the primary key, is also a simple key in its own right. An example might be a table that represents the modules a student is attending. This table has a studentId and a moduleCode as its primary key. Each of the fields that make up the primary key are simple keys because each represents a unique reference when identifying a student in one instance and a module in the other.

**3.3.9 Composite Key**

A composite key consists of more than one field to uniquely identify a record. This differs from a compound key in that one or more of the attributes, which make up the key, are not simple keys in their own right. Taking the example from compound key, imagine we identified a student by their firstName + lastName. In our table representing students on modules our primary key would now be firstName + lastName + moduleCode. Because firstName + lastName represent a unique reference to a student, they are not each simple keys, they have to be combined in order to uniquely identify the student. Therefore the key for this table is a composite key.

* 1. **Database Schema**

When we talk about a database, we must differentiate between the **database schema**, which is the logical design of the database, and a **database instance**, which is a snapshot of the data in the database at a given instant in time.

The concept of a relation corresponds to the programming-language notion of a variable. The concept of a **relation schema** corresponds to the programming-language notion of type definition.

It is convenient to give a name to a relation schema, just as we give names to type definitions in programming languages. We adopt the convention of using lowercase names for relations, and names beginning with an uppercase letter for relation schemas. Following this notation, we use *Account-schema* to denote the relation schema for relation *account*. Thus,

***Account*-*schema* = (*account-number*, *branch-name*, *balance*)**

We denote the fact that *account* is a relation on *Account-schema* by

*account*(*Account*-*schema*)

In general, a relation schema consists of a list of attributes and their corresponding domains.

The concept of a **relation instance** corresponds to the programming language notion of a value of a variable. The value of a given variable may change with time; similarly the contents of a relation instance may change with time as the relation is updated. However, we often simply say “relation” when we actually mean “relation instance.”

As an example of a relation instance, consider the *branch* relation of Figure 3.3. The schema for that relation is

***Branch*-*schema* = (*branch-name*, *branch-city*, *assets*)**

Note that the attribute *branch-name* appears in both *Branch-schema* and *Accountschema*.

This duplication is not a coincidence. Rather, using common attributes in

relation schemas is one way of relating tuples of distinct relations. For example, suppose we wish to find the information about all of the accounts maintained in branches

|  |  |  |
| --- | --- | --- |
| **Branch-name** | **Branch-city** | **Assets**  |
| Brighton  | Brooklyn  | 7100000 |
| Downtown  | Brooklyn  | 9000000 |
| Mianus  | Horseneck  | 400000 |
| Northtown  | Rye  | 3700000 |
| Perryridge  | Horseneck  | 1700000 |
| Powanl  | Bennington  | 300000 |
| Redwood  | Paloalto  | 2100000 |
| Round hill | Horseneck  | 8000000 |

**Figure 3.3** The *branch* relation.

located in Brooklyn. We look first at the *branch* relation to find the names of all the branches located in Brooklyn. Then, for each such branch, we would look in the *account* relation to find the information about the accounts maintained at that branch.

This is not surprising—recall that the primary key attributes of a strong entity set appear in the table created to represent the entity set, as well as in the tables created to represent relationships that the entity set participates in.

Let us continue our banking example. We need a relation to describe information about customers. The relation schema is

***Customer*-*schema* = (*customer-name*, *customer-street*, *customer-city*)**

Figure 3.4 shows a sample relation *customer* (*Customer-schema*). Note that we have omitted the *customer-id* attribute, which we used Chapter 2, because now we want to have smaller relation schemas in our running example of a bank database. We assume that the customer name uniquely identifies a customer—obviously this may not be true in the real world, but the assumption makes our examples much easier to read.

|  |  |  |
| --- | --- | --- |
| **Customer-name** | **Customer-street** | **Customer-city** |
| Adams  | Spring  | Pitsfield  |
| Brooks  | Senator  | Brooklyn  |
| Curry  | North  | Rye  |
| Glenn  | Sandhill  | Woodside  |
| Green  | Walnut  | Stamford  |
| Hayes  | Main  | Harrison  |
| johnson | Alma  | Paloalto  |
| Jones  | Main  | Harrison  |

**Figure 3.4** The *customer* relation.