TASK

Design and Build Relational Database

Visit our website

Introduction

WELCOME TO THE DESIGN AND BUILD RELATIONAL DATABASE TASK!

In this task, we introduce the relational database. We will also discuss how to evaluate and design good table structures to control data redundancies, and therefore avoid data anomalies.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with a code reviewer. You can also schedule a call or get support via email.

Our expert code reviewers are happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

WHAT IS A RELATIONAL DATABASE?

As mentioned in the previous task, a relational database is a database that organises data as a set of formally described tables. Data can then be accessed or reassembled from these tables in many different ways without having to reorganise the database tables.

Each table in a relational database has a unique name and may relate to one or more other tables in the database through common values. These tables contain rows (records) and columns (fields). Each row contains a unique instance of data for the categories defined by the columns. For example, a table that describes a *Customer* can have columns for *Name, Address, Phone Number,* and so on. Rows are sometimes referred to as tuples and columns are sometimes referred to as attributes. A relationship is a link between two tables. Relationships make it possible to find data in one table that pertains to a specific record in another table.

Tables in a relational database often contain a primary key, which is a column or group of columns used as a unique identifier for each row in the table. For example, a *Customer* table might have a column called *CustomerID* that is unique for every row. This makes it easy to keep track of a record over time and to associate a record with records in other tables.

Tables may also contain foreign keys, which are columns that link to primary key columns in other tables, thereby creating a relationship. For example, the *Customers* table might have a foreign key column called *SalesRep* that links to *EmployeeID*, which is the primary key in the *Employees* table.

A Relational Database Management System (RDBMS) is the software used for creating, manipulating, and administering a database. Most commercial RDBMSes use the Structured Query Language (SQL). SQL queries are the standard way to access data from a relational database. SQL queries can be used to create, modify, and delete tables as well as select, insert, and delete data from existing tables. You will learn more about SQL in a later task.

DATA REDUNDANCY

When the same data is stored in different places for no reason, this is called data redundancy. This could cause issues when the same versions of data are not updated consistently. This could cause different versions of the same information to be kept in a database, which could lead to misinformation when trying to

retrieve the most recent data. This is known as **data inconsistency**. Redundancy could also negatively impact the security of the data because having multiple versions of something increases the risk of someone being able to access it without authorisation.

Redundancies can also lead to abnormalities in the data, known as anomalies. **Data anomalies** can occur for when changes to the data are unsuccessful. For example:

- *Update anomalies:* Occur when the same information is recorded in multiple rows. For example, in an Employee table, if the office number changes, then there are multiple updates that need to be made. If these updates are not successfully completed across all rows, then an inconsistency occurs.

EmployeeID	SalesPerson	SalesOffice	OfficeNumber	Customer1	Customer2	Customer3
1003	Mary Smith	Chicago	**312-555-1212**	Ford	GM	
1004	John Hunt	New York	212-555-1212	Dell	HP	Apple
1005	Martin Hap	Chicago	**312-555-1212**	Boeing		

- *Insertion anomalies:* When there is data we cannot record until we know information for the entire row. For example, we cannot record a new sales office until we also know the salesperson because, in order to create the record, we need to provide a primary key. In our case, this is the EmployeeID.

EmployeeID	SalesPerson	SalesOffice	OfficeNumber	Customer1	Customer2	Customer3
1003	Mary Smith	Chicago	312-555-1212	Ford	GM	
1004	John Hunt	New York	212-555-1212	Dell	HP	Apple
1005	Martin Hap	Chicago	312-555-1212	Boeing		
???	**???**	**Atlanta**	**312-555-1212**			

- *Deletion anomalies:* When deletion of a row can cause more than one set of facts to be removed. For example, if John Hunt retires, deleting that row causes us to lose information about the New York office.

EmployeeID	SalesPerson	SalesOffice	OfficeNumber	Customer1	Customer2	Customer3
1003	Mary Smith	Chicago	312-555-1212	Ford	GM	
~~1004~~	~~John Hunt~~	~~New York~~	~~212-555-1212~~	~~Dell~~	~~HP~~	~~Apple~~
1005	Martin Hap	Chicago	312-555-1212	Boeing		

NORMALISATION

The aim of normalisation is to correct the tables in a relational database to reduce anomalies by eliminating redundancies. To achieve normalisation, we work through up to six stages called **normal forms**: 1NF (first normal form), 2NF, 3NF, etc. up to 6NF. However, it is rarely necessary to go through all six stages. It is most common to go through the first three (1NF to 3NF). Therefore, in the example below, we will be working through the first three stages. By working through these stages, each version of the tables becomes structurally better than the previous stage.

By the end of normalisation, the final tables should show the following:

- Each table represents a single subject. For example, an Employee table will only contain data that directly pertains to employees.
- No data is unnecessarily stored in more than one table.
- All attributes in a table are dependent on a primary key.

CONVERSION TO FIRST NORMAL FORM

Normalisation starts with a simple three-step procedure.

Step 1: Eliminate the repeating groups

To start, the data must be formatted to a table. It is important here that there are no repeating groups of data. Each cell must contain a maximum of one value.

Table name: MUSIC

SONG_NAME	SONG_ARTIST	SONG_ALBUM	SONG_GENRE	REC_LBL	SONG_PRDR
Hello	Adele	25	Soul	XL Recordings	Greg Kurstin
Sunflower	Post Malone	Spiderman: Into the Spider-Verse	Pop	Republic	Carter Lang
Wow.	Post Malone	Hollywood's Bleeding	Hip-Hop	Republic	Louis Bel

One	Ed Sheeran	x	Folk-pop	Asylum	Jake Gosling
Bad Guy	Billie Eilish	When We Fall Asleep, Where Do We Go?	Alternative	Dark Room	Finneas O'Connell
One	Metallica	...And Justice for All	Metal	One on One Studios	Metallica
Hello	Eminem	Relapse	Hip-Hop	Aftermath Shady Interscope	Dr Dre
7 Rings	Ariana Grande	Thank U, Next	Pop	Republic	Tommy Brown
Thank U, Next	Ariana Grande	Thank U, Next	Pop	Republic	Tommy Brown

Take a look at the table above. Note that some songs could reference more than one data entry. For example, a song named "Hello" could refer to one of two songs, each with a different artist. To make sure there is no repeated data, we need to fill in the blanks to make sure there are no null values in the table. Looking at the last row, the empty cells contain the same information as the row above because the two songs come from the same album. Therefore, the filled-in table will look like this:

Table name: 1NF_MUSIC

SONG_NAME	SONG_ARTIST	SONG_ALBUM	SONG_GENRE	REC_LBL	SONG_PRDR
Hello	Adele	25	Soul	XL Recordings	Greg Kurstin
Sunflower	Post Malone	Spiderman: Into the Spider-Verse	Pop	Republic	Carter Lang
Wow.	Post Malone	Hollywood's Bleeding	Hip-hop	Republic	Louis Bel
One	Ed Sheeran	x	Folk-pop	Asylum	Jake Gosling
Bad Guy	Billie Eilish	When We Fall Asleep, Where Do We Go?	Alternative	Dark Room	Finneas O'Connell
One	Metallica	...And Justice for All	Metal	One on One Studios	Metallica

Hello	Eminem	Relapse	Hip-Hop	Aftermath Shady Interscope	Dr. Dre
7 Rings	Ariana Grande	Thank U, Next	Pop	Republic	Tommy Brown
Thank U, Next	Ariana Grande	Thank U, Next	Pop	Republic	Tommy Brown

Step 2: Identify the primary key

Next, we need to identify the primary key. The primary key needs to be able to identify one row of the table, thereby identifying all the remaining rows, or attribute, when needed. For example, SONG_NAME is not a good primary key because “One” and “Hello” each identify 2 different rows. Similarly, SONG_ARTIST would not be a good primary key because “Ariana Grande” and “Post Malone” each identify 2 rows. Therefore, we could use a combination of SONG_NAME and SONG_ARTIST to be able to narrow down our search to a single row. Now, if you know that SONG_NAME = “Hello and SONG_ARTIST = “Adele”, the entries for the attributes can only be “25”, “Soul”, “XL Recordings”, and “Greg Kurstin”

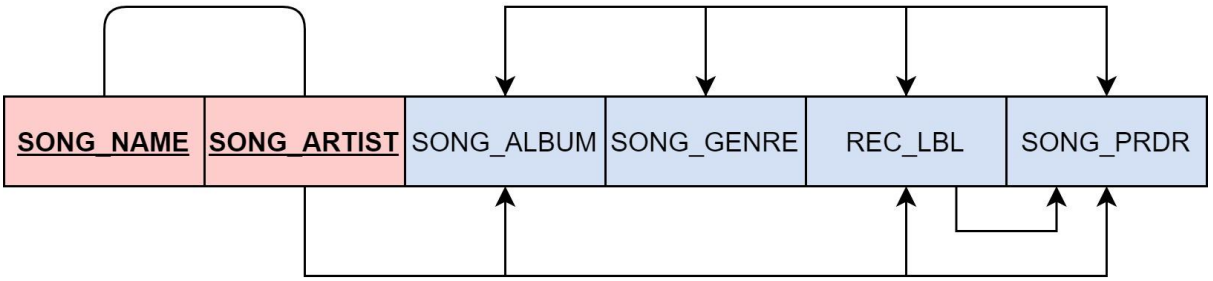
Step 3: Identify all dependencies

The example above, step 1 shows us the following relationship:

SONG_NAME, ARTIST_NAME → SONG_ALBUM, SONG_GENRE, REC_LBL, SONG_PRDR

This means that SONG_ALBUM, SONG_GENRE, REC_LBL and SONG_PRDR are dependent on the combination of SONG_NAME and ARTIST_NAME. However, looking at the table, we could also see that the song artist determines the album name, the record label and the record producer (SONG_ARTIST → SONG_ALBUM, REC_LBL, SONG_PRDR). These are known as partial dependencies because only one of the two primary keys is needed to determine the other attributes.

Finally, knowing the record label means knowing the song’s producer (REC_LBL → SONG_PRDR). This is known as a partial dependency because it is not based on a primary attribute. For a visual representation of this, look at the dependency diagram below:



Note that:

1. The primary key attributes are underlined and shaded in a different colour.

2. The arrows above the attributes indicate all desirable dependencies. Desirable dependencies are those based on the primary key. Note that the entity's attributes are dependent on the combination of SONG_NAME and ARTIST_NAME.

3. The arrows below the diagram indicate partial dependencies and transitive dependencies. These are less desirable than those based on the primary key

We have now put the table in 1NF because:

- All of the key attributes are defined
- There are no repeating groups in the table
- All attributes are dependent on the primary key

CONVERSION TO SECOND NORMAL FORM

The relational database design can be improved by converting the database into a format known as the second normal form. Here, we try to rid the tables of partial dependencies, as they can cause anomalies.

Step 1: Write each key component on a separate line

Write each key component on a separate line, then write the original (composite) key on the last line:

SONG_NAME
SONG_ARTIST
SONG_NAME SONG_ARTIST

You may have noticed from the diagram above that SONG_ARTIST is not a primary key on its own for any attribute, therefore giving it its own new table isn't necessary. Therefore, the keys that will have their own tables are SONG_NAME and SONG_NAME SONG_ARTIST. We will name these two tables ARTIST and GENRE (because the table is used to determine the genre) respectively.

Step 2: Assign corresponding dependent attributes

Then use the dependency diagram above to determine those attributes that are dependent on other attributes by looking at the arrows underneath the diagram. With this, we create three new tables:

ARTIST (SONG_ARTIST, SONG_ALBUM, REC_LBL, SONG_PRDR)
GENRE (SONG_NAME, SONG_ARTIST, SONG_GENRE)

The dependency diagram below shows the result of Steps 1 and 2.

Table name: GENRE

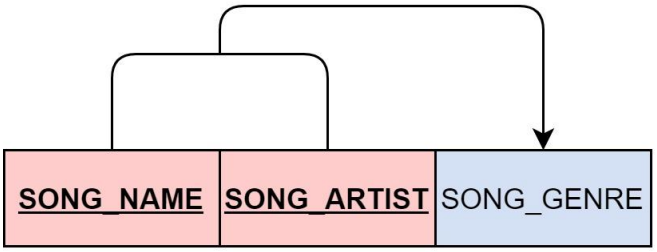
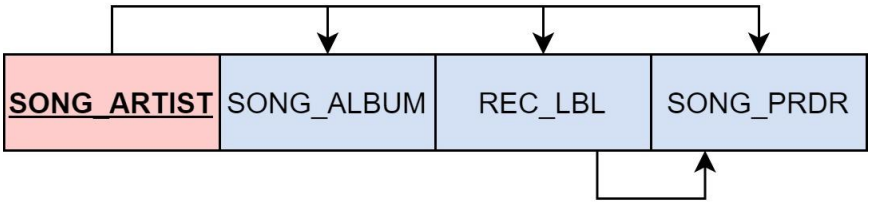


Table name: ARTIST



Note how there are no longer any partial dependencies that could cause anomalies in the tables. However, there is still transitive dependency in the Employee table.

We have now put the table into 2NF because:

- It is in 1NF
- It includes no partial dependencies because no attribute in each table is dependent on only a part of a primary key

CONVERSION TO THIRD NORMAL FORM

We will now eliminate the transitive dependencies.

Step 1: Identify each new determinant

Write the determinant of each transitive dependency as a primary key in its own table. In this example, the determinant will be REC_LBL.

Step 2: Identify the dependent attributes

As mentioned, SONG_PRDR is dependent on REC_LBL (REC_LBL → SONG_PRDR). Therefore, we can name the new table Label.

Step 3: Remove the dependent attributes from transitive dependencies

Eliminate all dependent attributes in the transitive relationship from each of the tables that have such a relationship. In this example, we eliminate SONG_PRDR from the ARTIST table shown in the dependency diagram above to leave the ARTIST table dependency definition as SONG_ARTIST → SONG_ALBUM, REC_LBL.

Notice that the REC_LBL remains in the ARTISTtable to serve as the foreign key. You can now draw a new dependency diagram to show all of the tables you have defined in the steps above. Then check the new tables as well as the tables you modified in Step 3 to make sure that each table has a determinant and that the tables don't contain inappropriate dependencies. The new dependency diagram should look as follows:

Table name: GENRE

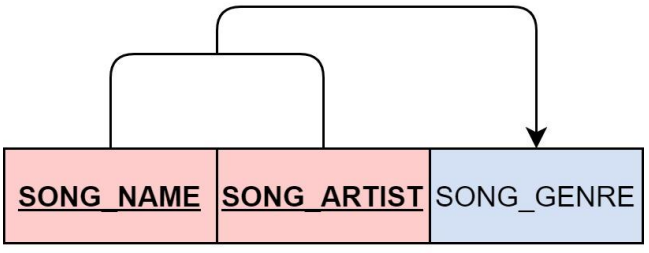


Table name: ARTIST

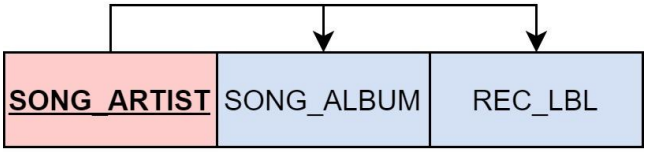
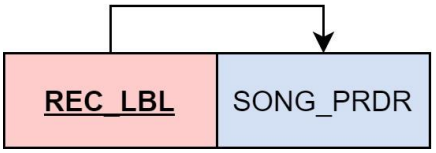


Table name: LABEL



This conversation has eliminated the original ARTIST table's transitive dependency. The tables are now said to be in third normal form (3NF).

We have now put the table into 3NF because:

- It is in 2NF
- It contains no transitive dependencies

Compulsory Task

Answer the following questions:

- Using the INVOICE table given below, draw its dependency diagram and identify all dependencies (including transitive and partial dependencies). You can assume that the table does not contain any repeating groups and that an invoice number references more than one product.
  Hint: This table uses a composite primary key

INV_NUM	PROD_NUM	SALE_DATE	PROD_LABEL	VEND_CODE	VEND_NAME	QUANT_SOLD	PROD_PRICE
211347	AX-P126VBB	15-Jun-2018	Lawnmower	111	Mowers and Bulbs	1	R4099.90
211347	GF-5106D2F	15-Jun-2018	Lightbulb	111	Mowers and Bulbs	8	R24,99
211347	RB-163698G	15-Jun-2018	Blue paint	063	All Things Paint	1	R80,90
211348	AX-P126VBB	15-Jun-2018	Lawnmower	111	Mowers and Bulbs	2	R4099.90
211349	TF-7465023Q	17-Jun-2018	2m ladder	207	Tall Things	1	R699,99

- Draw new dependency diagrams to show the data in 2NF.
- Draw new dependency diagrams to show the data in 3NF.
- Use this table to illustrate one of each kind of anomaly described in the task. In a text file called **anomalies.txt**, how you would change the table and which anomaly that change would create.

Completed the task(s)?

Ask an expert to review your work!

Review work

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.

References:

Coronel, C., Morris, S., & Rob, P. (2011). *Database systems* (9th ed., pp. 4-26). Boston: Cengage Learning.

Coronel, C., Morris, S., & Rob, P. (2011). *Database systems* (9th ed., pp. 174-218). Boston: Cengage Learning.