HyperionDev

Relational Databases

Task

Visit our website

Introduction

Relational databases are the backbone of modern data management systems, powering everything from small business applications to large-scale enterprise solutions. At their core, these databases organise information into tables with rows and columns, allowing for efficient storage and retrieval of data. However, as databases grow in size and complexity, they can become prone to issues like data redundancy – the unnecessary duplication of information across multiple tables. This redundancy not only wastes storage space but can also lead to data inconsistencies and anomalies. To address these challenges, database designers employ a process called normalisation, which involves structuring the database to minimise redundancy and dependency. In this lesson, we will explore the concepts of relational databases, examine the problems caused by data redundancy, and learn how normalisation techniques can help create more efficient and reliable database systems.

Relational databases

A relational database organises data into formally described tables, linked by defined relationships. This structure allows for flexible data access and retrieval without needing to reorganise the database. Users can combine information from multiple tables using a single query, enabling efficient data management and analysis.

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. Each column represents a specific characteristic or attribute of the data and each row contains a unique instance of data for the attributes defined by the columns. For example, a table that describes a customer can have columns for name, address, and phone number and each row represents a particular customer. Each column is also associated with a data type and may have defined constraints. For instance, phone numbers may be limited to 10 characters or grades could be defined as a numerical data type between 0 and 100. Defining data types and constraints helps to keep data in a uniform format to minimise data inconsistencies.

There are several synonymous terms used to describe relational databases. In most instances, we will use the commonly used terms, but you should be aware of the other terminology in the table below (Watt and Eng, 2014).

Formal terms	**Commonly used**	**Alternative**
Relation	Table	File
Tuple	Row	Record
Attribute	Column	Field

Relationships between tables make it possible to find data in one table that pertains to a specific row in another table. Tables in a relational database 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 data over time and to associate a row with data 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 instance, the Customers table might have a column called `SalesRep` that links to `EmployeeID`, which is the primary key in the `Employees table`. In this case, the `SalesRep` column is a foreign key. It is being used to show that there is a relationship between a specific customer and a specific sales rep. Understanding these relationships is crucial for effectively querying and managing data.

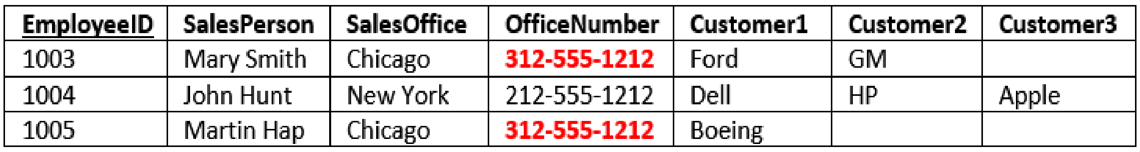
Data redundancy

Understanding common database issues and their solutions is crucial when discussing data organisation. One such issue is data redundancy, which occurs when identical information is stored in multiple locations within a database or data storage system. This duplication can lead to 'islands of information' – scattered data locations that may contain inconsistent versions of the same data. For instance, consider a customer table where a customer's address is stored in both a `'billing_address'` column and a `'shipping_address'` column. If the customer moves house and only one column is updated, the table now contains conflicting information about where the customer lives. This inconsistency can compromise data integrity and lead to practical problems, such as failed deliveries based on the customer's location.

Uncontrolled data redundancy can cause issues such as:

- **Data inconsistency**: This exists when different and conflicting versions of the same data appear in different places.
- **Poor data security**: Having multiple copies of data creates more potential points of vulnerability. Each instance of the data becomes a potential target for unauthorised access.

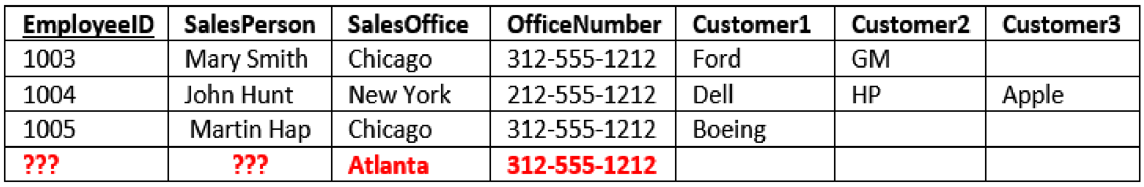
- **Data anomalies**: This occurs when changes to redundant data are not consistently applied across all instances, leading to inconsistencies or errors in the stored information. Data anomalies are defined by Coronel and Morris (2014) as:
    - **Update anomalies**: These occur when the same information is recorded in multiple rows. For instance, in the `Employee` table below, 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		

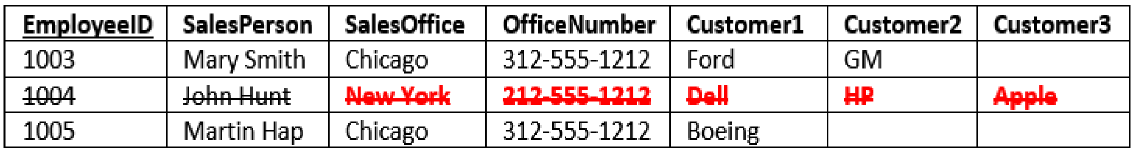
Table (Sharma and Kaushik, 2015)

-
    - **Insertion anomalies**: There is data we cannot record until we know information for the entire row. For example, we cannot capture a new sales office until we also know the salesperson because, in order to create the entry, 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**: The deletion of a row can cause more than one set of facts to be removed. For example, if John Hunt retires, then 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		

To address the challenges posed by data redundancy, database designers employ a structured technique known as normalisation, which aims to optimise table structures and improve data integrity.

Normalisation

Normalisation is a process of restructuring database tables to minimise redundancies and reduce data anomalies, ultimately bringing the database to a consistent state. It is used as part of the design process or to modify existing data structures. It works through a series of stages called **normal forms.** The first three stages are described as the:

- First normal form (1NF),
- Second normal form (2NF), and
- Third normal form (3NF).

2NF is structurally better than 1NF, and 3NF is structurally better than 2NF. Normally, 3NF is as high as you need to go in the normalisation process for most business database design purposes. The objective of normalisation, according to Coronel and Morris (2014), is to create tables that have the following characteristics:

- Each table represents a single subject. For instance, an employee table will only contain data that directly pertains to employees.
- No data item will be unnecessarily stored in more than one table, so that data is only updated in one place.
- All attributes in a table are functionally dependent on the primary key.

In a relational database, saying that "all attributes in a table are dependent on the primary key" means that each attribute (or column) in the table is related to and can be uniquely identified by the primary key of the table. In simpler terms:

- **Primary key**: This is a unique identifier for each row in the table. For example, in a `Customers` table, the `CustomerID` might be the primary key, with each `CustomerID` being unique to each customer.
- **Attribute dependency**: If an attribute is dependent on the primary key, it means that the value of that attribute is determined by the primary key. In other words, knowing the primary key value allows you to **uniquely** identify the values of all other attributes in that row.

For example, consider a `Customers` table with attributes `CustomerID`, `Name`, and `Email`:

- `CustomerID` is the primary key.
- `Name` and `Email` are attributes dependent on `CustomerID`.

If you know a specific `CustomerID`, you can find the corresponding `Name` and `Email` for that customer. Conversely, if you have a `CustomerID`, you can be sure that the `Name` and `Email` values you retrieve are unique to that `CustomerID`.

Extra resource

If you'd like to know more about functional dependencies and normalisation, we recommend reading the book ***Database Design – 2nd Edition*** by Adrienne Watt. **Chapters 11** and **12** of this book are a great supplement to this task.

Now that you understand the benefits and characteristics of a normalised database let's work through the steps to normalise a database to the 3NF.

Conversion to first normal form (1NF)

According to Coronel and Morris (2014), the normalisation starts with a simple three-step procedure. Let's walk through the steps with the help of an example. Imagine a project manager comes to you to design a database. Part of their job is to keep track of expenses based on each person's hourly rate and their time contribution to each project. They provide you with a table with some data for you to design and create a relational database. Let's get started!

Step one: Eliminate the repeating groups

The first condition that needs to be met is that data must be presented in a tabular format, where each ‘cell’ contains a single value. When a cell contains more than one value of the same type, it is known as a repeating group. Take a look at the table provided by your project manager below. Note that each project number can reference a group of related data entries. The Evergreen project, for instance, is associated with five entries in each column, one for each person working on the project. To normalise the data, each employee number, employee name, job class, charge per hour and hours worked should be separated into individual rows.

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
15	Evergreen	103, 101, 105, 106, 102	June Arbaugh, John News, Alice Johnson, William Smithfield, David Senior	Elect. Engineer, Database Designer, Database Designer, Programmer, System Analyst	$67.55, $82.00, $82.00, $26.66, $76.43	23, 19, 35, 12, 12
18	Amberwave	114, 118, 104, 112	Ann Jones, James Frommer , Anne Remoras, Darlene Smithson	Applications Designer, General Support, System Analyst, DSS Analyst	$38.00, $14.50, $76.43, $36.30	24, 45, 32, 44
22	Rolling Tide	105, 104, 113, 111, 106	Alice K. Johnson, Ann K Ramoras, Delbert K. Joenbroek, Geoff B. Wabash, William Smithfield	DB Designer, Systems Analyst, Applications Designer, Clerical Support, Programmer	$105, $96, $75, $48.1, $26.87, $35.75	65.7, 48.4, 23.6, 22., 12.8
25	Starflight	107, 115, 101, 114, 108, 118, 112	Maria D. Alonzo, Travis B. Bawangi, John G. News, Annelise Jones, Ralph B. Washington, James J. Frommer, Darlene M. Smithson	Programer, Systems Analyst, Database Design, Applications Designer, Systems Analyst, General Support, DSS Analyst	$35, 75, $96.75, $105., $48.1, $96.75, $18.36, $45.95	25.6, 45.8, 56.3, 33.1, 23.6, 30.5, 41.4

Table (Coronel and Morris, 2014)

Eliminating repeating groups converts the table above to 1NF as shown below.

`PROJ_NUM`	`PROJ_NAME`	`EMP_NUM`	`EMP_NAME`	`JOB_CLASS`	`CHG_HOUR`	`HOURS`
15	Evergreen	103	June Arbaugh	Elect. Engineer	$67.55	23
15	Evergreen	101	John News	Database Designer	$82.00	19
15	Evergreen	105	Alice Johnson	Database Designer	$82.00	35
15	Evergreen	106	William Smithfield	Programmer	$26.66	12
15	Evergreen	102	David Senior	System Analyst	$76.43	12
18	Amberwave	114	Ann Jones	Applications Designer	$38.00	24
18	Amberwave	118	James Frommer	General Support	$14.50	45
18	Amberwave	104	Anne Remoras	System Analyst	$76.43	32
18	Amberwave	112	Darlene Smithson	DSS Analyst	$36.30	44

Step two: Identify the primary key

Recall that the primary key is a unique identifier for each row in the table. `PROJ_NUM` cannot be used as the primary key because there are multiple rows with the same project number. To create a primary key that will **uniquely** identify a row in our table, we can combine `PROJ_NUM` and `EMP_NUM`. So, if you know that `PROJ_NUM = 15` and `EMP_NUM = 103`, the entries for the attributes can only be Evergreen, June Arbaugh, Elect. Engineer, $67.55, and 23.

Step three: Identify all dependencies

We have already identified the following dependency by finding the primary key in step two:

```
PROJ_NUM, EMP_NUM -> PROJ_NAME, EMP_NAME, JOB_CLASS, CHG_HOUR, HOURS
```

This means that `PROJ_NAME`, `EMP_NAME`, `JOB_CLASS`, `CHG_HOUR`, and `HOURS` are determined by the combination of `PROJ_NUM` and `EMP_NUM`. However, there are other dependencies to consider. For example, the project number determines the project name. You can write this dependency as:

```
PROJ_NUM -> PROJ_NAME
```

You can also determine an employee's name, job classification, and charge per hour if you know an employee number. You can write this dependency as:

```
EMP_NUM -> EMP_NAME, JOB_CLASS, CHG_HOUR
```

Lastly, knowing the job classification means knowing the charge per hour for that job classification:

```
JOB_CLASS -> CHG_HOUR
```

A dependency diagram can help depict dependencies. They are helpful in getting a bird's-eye view of all the relationships among a table's attributes.

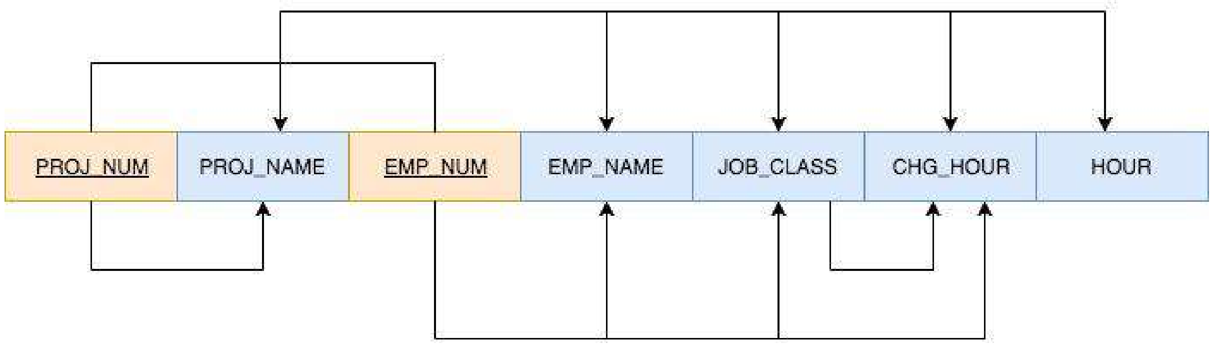


Diagram (Coronel and Morris, 2014)

Look at the dependency diagram above:

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 a row's attributes are dependent on the combination of `PROJ_NUM` and `EMP_NUM`.

3. The arrows below the diagram indicate less desirable dependencies. There are two types of less desirable dependencies:

    a. **Partial dependencies**: Dependencies based on only a part of the composite primary key. For example, you only need to know `PROJ_NUM` to determine `PROJ_NAME`. Similarly, `EMP_NAME`, `JOB_CLASS`, and `CHG_HOUR` are only dependent on `EMP_NUM`.

    b. **Transitive dependencies**: Dependencies of one non-prime attribute (an attribute that is not part of a primary key) on another non-prime attribute. For example, `CHG_HOUR` is dependent on `JOB_CLASS`.

A table is in 1NF when:

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

Spot check 1

Let's see what you can remember from this section.

1. Examine the information provided in the table below. Then, identify the attribute that does not depend on the primary key and would be more appropriately placed in a different table for the following data tables:

    a. Order table

    b. Employee table

Table	Primary Key	Other attributes
`Order`	`OrderID`	`CustomerName, OrderDate, TotalAmount, ProductPrice`
`Employee`	`Employee ID`	`Name, DepartmentName, HireDate, Position, Salary`

Conversion to second normal form (2NF)

If the primary key identified in 1NF is not a composite key of two attributes (columns) then it is already in 2NF. However, in our example the primary key is a combination of `PROD_NUM` and `EMP_NUM`. According to Coronel and Morris (2014), the following steps are required to convert a database to 2NF:

Step one: Write each key component on a separate line

Write each part of the primary key on a separate line, then write the composite key on the last line:

```
PROJ_NUM

EMP_NUM

PROJ_NUM EMP_NUM
```

Each part of the primary key will become the primary key of a new table, namely the `PROJECT` and `EMPLOYEE` tables. However, not all the attributes are uniquely determined by the project number or employee number alone. We will see in the next step that the composite primary key uniquely determines the assigned hours (`HOURS`) for a project. Let's call this the `ASSIGNMENT` table. Note in other scenarios you may not need to retain the composite key.

Step two: Assign corresponding dependent attributes

Use the dependency diagram we created in step three of the 1NF conversion to determine the attributes that are dependent on the three primary keys identified in step one. The dependencies for the new primary keys are found by examining the arrows below the dependency diagram. The three new tables are described by the following relational schemas (a blueprint that defines the structure of a database, including its tables, columns, and relationships):

PROJECT (**PROJ_NUM**, PROJ_NAME)

EMPLOYEE (**EMP_NUM**, EMP_NAME, JOB_CLASS, CHG_HOUR)

ASSIGNMENT (**PROJ_NUM**, **EMP_NUM**, ASSIGN_HOURS)

The number of hours spent on each project by each employee is dependent on both `PROJ_NUM` and `EMP_NUM`; you, therefore, place those hours in the `ASSIGNMENT` table and rename the attribute `ASSIGN_HOURS`.

The dependency diagram below shows the results of steps one and two.

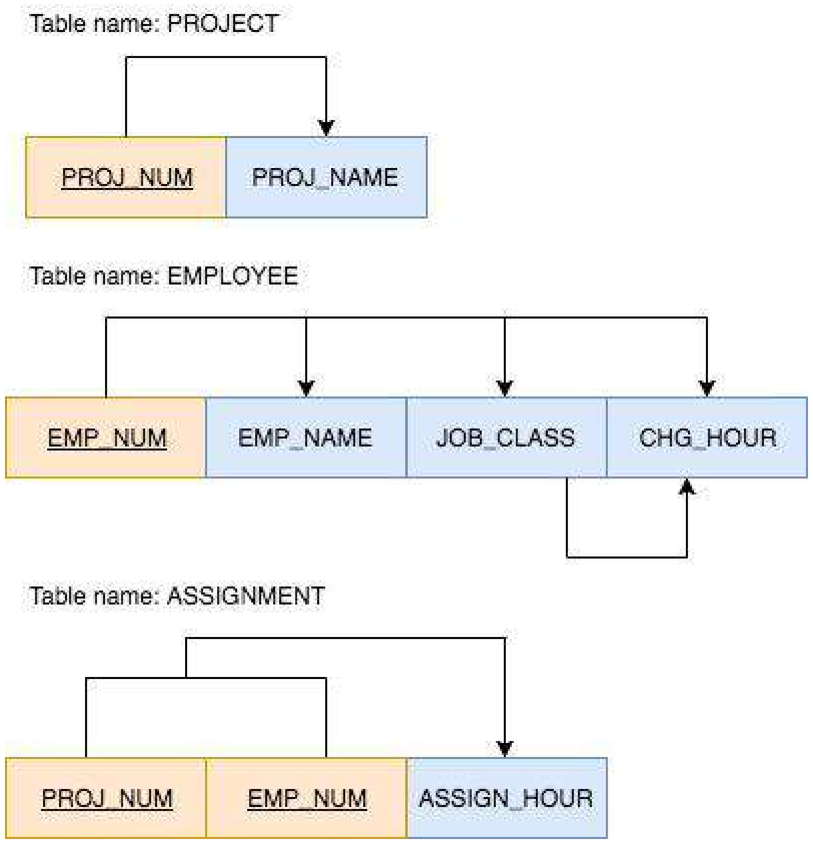


Diagram (Coronel and Morris, 2014)

At this point, most of the anomalies discussed earlier have been eliminated. For instance, if you now want to add, change, or delete a `PROJECT` record, you need to go only to the `PROJECT` table and add, change, or delete only one row.

A table is in 2NF when:

- It is in 1NF, and
- It includes no partial dependencies; that is, no attribute is dependent on only a portion of the primary key.

It is still possible for a table in 2NF to exhibit transitive dependency; that is, one or more attributes may be functionally dependent on non-key attributes (Coronel and Morris, 2014). In this example, charge per hour `(CHG_HOUR)` is dependent on the non-key attribute `JOB_CLASS`. We will tackle transitive dependencies in the conversion to 3NF.

Conversion to third normal form (3NF)

The transitive dependency present in the `EMPLOYEE` table above is easily eliminated by conversion to the 3NF. Coronel and Morris (2014) outline the following steps:

Step one: Identify each new determinant

A determinant is any attribute whose value determines other values within a row. For every transitive dependency, write its determinant as a primary key for a new table. The dependency diagram above shows a table that contains only one transitive dependency. Therefore, we write the determinant for this transitive dependency as:

```
JOB_CLASS
```

Step two: Identify the dependent attributes

Identify the attributes that are dependent on each determinant identified in step one. You will write in this case:

```
JOB_CLASS -> CHG_HOUR
```

Give the table a name that reflects its contents and function. We shall name this table `JOB.`

Step three: 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 instance, we eliminate `CHG_HOUR` from the `EMPLOYEE` table shown in the dependency diagram above to leave the `EMPLOYEE` table dependency definition as:

```
EMP_NUM -> EMP_NAME, JOB_CLASS.
```

Notice that the `JOB_CLASS` remains in the `EMPLOYEE` table 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 three to make sure that each table has a determinant and that no table contains inappropriate dependencies. The new dependency diagram should look as follows:

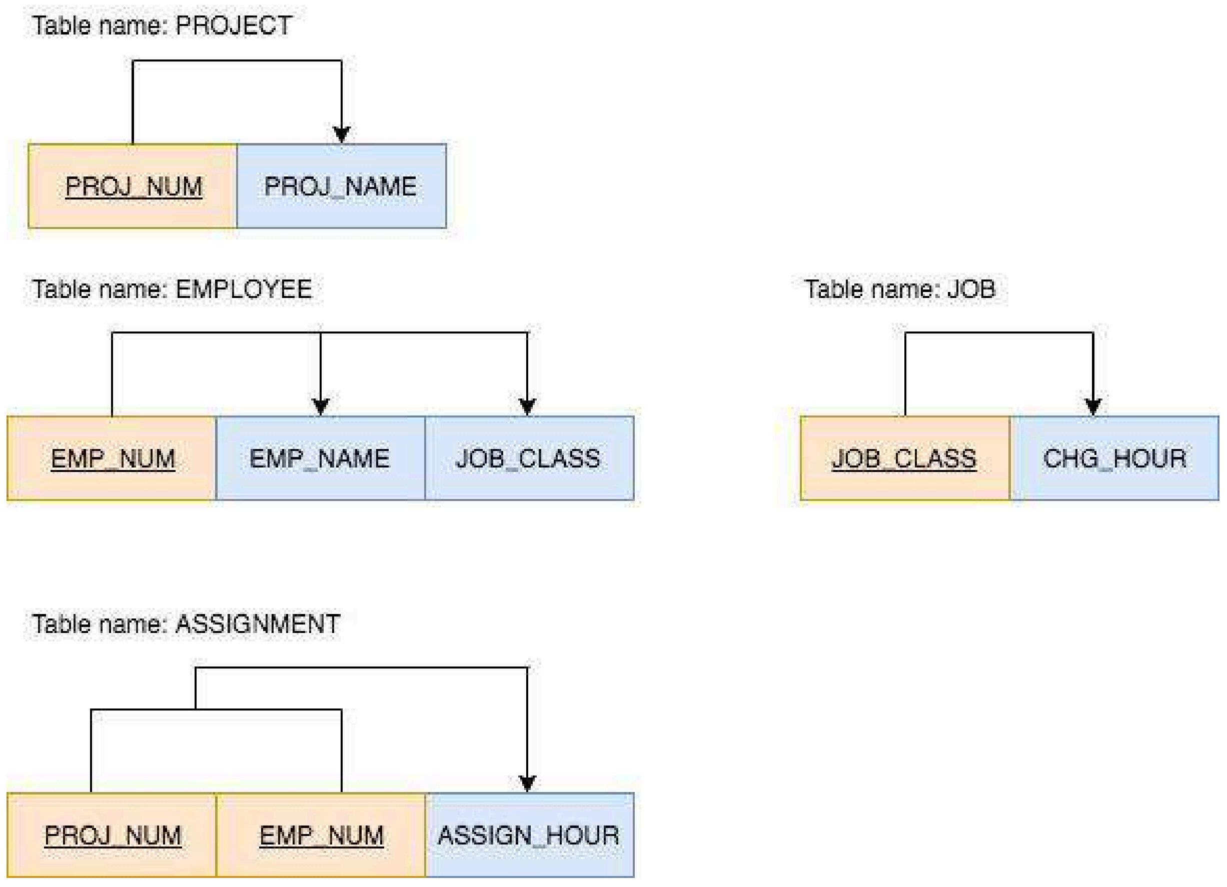


Diagram (Coronel and Morris, 2014)

After the conversion has been completed, your database should contain four tables:

```
PROJECT (PROJ_NUM, PROJ_NAME)

EMPLOYEE (EMP_NUM, EMP_NAME, JOB_CLASS)

JOB (JOB_CLASS, CHG_HOUR)

ASSIGNMENT (PROJ_NUM, EMP_NUM, ASSIGN_HOURS)
```

This conversation has eliminated the original `EMPLOYEE` table’s transitive dependency. The tables are now said to be in the 3NF.

A table is said to be in 3NF according to Coronel and Morris (2014), when:

- It is in 2NF, and
- It contains no transitive dependencies.

Potential pitfalls

While normalisation is a critical process in relational database design, it does come with its own set of challenges or pitfalls. Here are some common ones:

1. **Performance issues**: Normalisation often leads to data being spread across multiple tables. This can result in more complex queries and increased join operations, which can negatively impact performance.
2. **Increased complexity:** As normalisation progresses through the various normal forms, the number of tables can increase significantly. This can make the database more complex to understand and manage.
3. **Redundancy elimination**: While normalisation aims to eliminate redundancy, it doesn't always completely eliminate it. Some redundancy may still exist, especially if the database design isn't fully optimised.
4. **Update anomalies**: Despite the goal of normalisation to reduce update anomalies, in some cases, it can introduce new ones. For example, if not all dependencies are properly identified and addressed during the normalisation process.
5. **Loss of business semantics**: Normalisation focuses on the structure of the data, not on the business processes or rules that the data supports. This can sometimes lead to a loss of business semantics or context.
6. **Over-normalisation**: There's a risk of over-normalising data, which can lead to excessive division of tables and an overly complex database structure. This can result in decreased performance and increased difficulty in querying data.

Design implementation

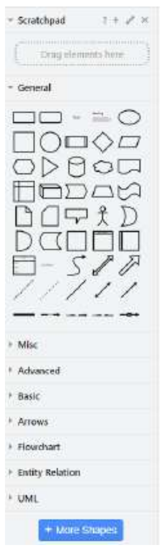
After you have designed or improved on a design for your relational database you will implement it using a relational **database management system (DBMS)**. This is a software used for creating, manipulating, and administering a database. Common DBMSs include MySQL, MariaDB and PostgreSQL. A query language is used to communicate with the DBMS. Most commercial relational DBMSs use the **structured query language (SQL).** SQL queries can be used to create, modify, and delete tables, as well as select, insert, and delete data from existing tables.

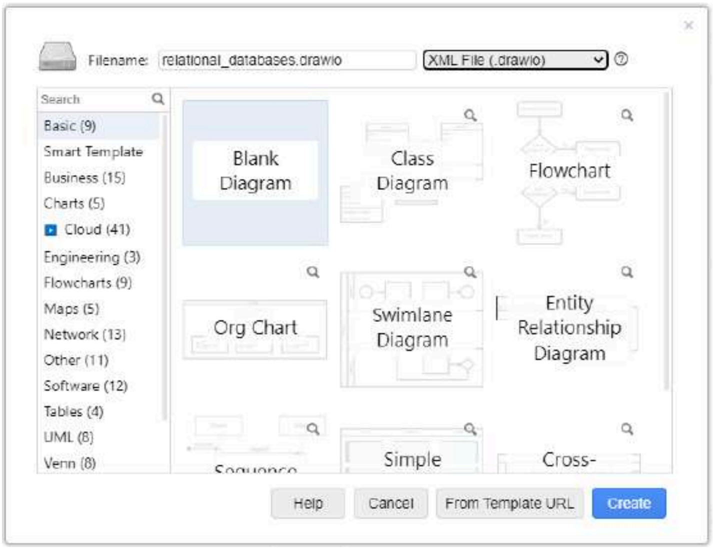
Take note

Various tools can be used to create dependency diagrams. One such tool is **Draw.io,** a free and open-source, cross-platform graph drawing software developed in HTML5 and JavaScript. Its interface can be used to create diagrams such as flowcharts, wireframes, UML diagrams, organisational charts, and network diagrams.

To create a dependency diagram:

- Open **https://www.draw.io/**.
- When creating a new file, we recommend starting with a "Blank Diagram" and using a selection of the basic shapes and arrows, from the "General" tab, as well as the "Table" element to easily map out and connect dependencies as is illustrated in this task.




Filename: relational_databases.drawio
XML File (.drawio)

- If you're interested in trying your hand at a more complex diagram, feel free to explore some of the options available in the "UML" templates.

While Draw.io provides an intuitive interface for designing and creating high-quality diagrams, they also provide interactive tutorials and step-by-step guides on their **Tutorials Page**.

Take note

The task(s) below is/are **auto-graded**. An auto-graded task still counts towards your progression and graduation. Give it your best attempt and submit it when you are ready.

When you select "Request Review", the task is automatically complete, you do not need to wait for it to be reviewed by a mentor.

You will then receive an email with a link to a model answer, as well as an overview of the approach taken to reach this answer.

Take some time to review and compare your work against the model answer. This exercise will help solidify your understanding and provide an opportunity for reflection on how to apply these concepts in future projects.

In the same email, you will also receive a link to a survey, which you can use to self-assess your submission.

Once you've done that, feel free to progress to the next task.

Auto-graded task

Answer the following questions:

1. What is normalisation?
2. When is a table in 1NF?
3. When is a table in 2NF?
4. When is a table in 3NF?

5. 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	AA-E3522QW	15-Jan-2018	Rotary sander	211	NeverFail, Inc.	1	$34.46
211347	QD-300932X	15-Jan-2018	0.25-in. Drill bit	211	NeverFail, Inc.	8	$2.73
211347	RU-995748G	15-Jan-2018	Band saw	309	BeGood, Inc.	1	$31.59
211348	AA-E3522QW	15-Jan-2018	Rotary sander	211	NeverFail, Inc.	2	$34.46
211349	GH-778345P	16-Jan-2018	Power drill	157	ToughGo, Inc.	1	$69.32

6. Using the answer to the above question, remove all partial dependencies and draw the new dependency diagrams.

7. Using the answer to the above question, remove all transitive dependencies and draw the new dependency diagrams.

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Spot check 1 answers

1. Identify the attribute that does not depend on the primary key and would be more appropriately placed in a different table, based on the attributes provided in the table below for the following data tables:

    a. **Order:** the `ProductPrice` would depend on the Product not the order

    b. **Employee**: the `DepartmentName` would depend on the Department, not directly on the employee

Table	**Primary key**	**Other attributes**
`Order`	`OrderID`	`CustomerName, OrderDate, TotalAmount, ProductPrice`
`Employee`	`EmployeeID`	`Name, DepartmentName, HireDate, Position, Salary`

Share your thoughts

Please take some time to complete this short feedback **form** to help us ensure we provide you with the best possible learning experience.

Reference list

Coronel, C., and Morris, S. (2014). *Data Systems: Design, Implementation and Management*. London: Cengage Learning.

Sharma, R., and Kaushik, S. (2015). *Database Management System*. Horizon Books (A Division of Ignited Minds Edutech P Ltd).

Watt, A., and Eng, N. (2014). *Database design*. BCcampus.