

THE MANGA GUIDE™ TO

DATABASES

MANA TAKAHASHI
SHOKO AZUMA
TREND-PRO CO., LTD.



THE MANGA GUIDE™ TO DATABASES



THE MANGA GUIDE™ TO DATABASES

MANA TAKAHASHI
SHOKO AZUMA
TREND-PRO CO., LTD.



THE MANGA GUIDE TO DATABASES. Copyright © 2009 by Mana Takahashi and TREND-PRO Co., Ltd.

The Manga Guide to Databases is a translation of the Japanese original, *Manga de Wakaru Database*, published by Ohmsha, Ltd. of Tokyo, Japan, © 2004 by Mana Takahashi and TREND-PRO Co., Ltd.

This English edition is co-published by No Starch Press, Inc. and Ohmsha, Ltd.

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

11 10 09 08 1 2 3 4 5 6 7 8 9

ISBN-10: 1-59327-190-5

ISBN-13: 978-1-59327-190-9

Publisher: William Pollock

Author: Mana Takahashi

Illustrator: Shoko Azuma

Producer: TREND-PRO Co., Ltd.

Production Editor: Megan Dunchak

Developmental Editor: Tyler Ortman

Technical Reviewers: Baron Schwartz and Peter MacIntyre

Composer: Riley Hoffman

Proofreader: Cristina Chan

Indexer: Sarah Schott

For information on book distributors or translations from the English edition, please contact No Starch Press, Inc.
No Starch Press, Inc.

555 De Haro Street, Suite 250, San Francisco, CA 94107

phone: 415.863.9900; fax: 415.863.9950; info@nostarch.com; <http://www.nostarch.com/>

Library of Congress Cataloging-in-Publication Data

Takahashi, Mana.

The Manga guide to databases / Mana Takahashi, Shoko Azuma, and Trend-pro Co. -- 1st ed.

p. cm.

Includes index.

ISBN-13: 978-1-59327-190-9

ISBN-10: 1-59327-190-5

1. Database management--Comic books, strips, etc. 2. Database management--Caricatures and cartoons. 3. SQL (Computer program language)--Comic books, strips, etc. 4. SQL (Computer program language)--Caricatures and cartoons. I. Azuma, Shoko, 1974-. II. Trend-pro Co. III. Title.

QA76.9.D3T34 2009

005.75'65--dc22

2008046159

No Starch Press and the No Starch Press logo are registered trademarks of No Starch Press, Inc. Other product and company names mentioned herein may be the trademarks of their respective owners.

The information in this book is distributed on an "As Is" basis, without warranty. While every precaution has been taken in the preparation of this work, neither the author nor No Starch Press, Inc. shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in it.

All characters in this publication are fictitious, and any resemblance to real persons, living or dead, is purely coincidental.

CONTENTS

PREFACE	ix
1	
What Is a Database?	1
Why Do We Need Databases?	2
What's Up in the Kingdom?	16
Data Is Duplicated	16
Data Can Conflict	17
Data Is Difficult to Update	18
A Database—That's Our Solution!	19
How to Use a Database	19
Summary	21
2	
What Is a Relational Database?	23
Database Terms	24
Relational Databases	34
Types of Data Models	39
Data Extraction Operations	39
Set Operations	39
Relational Operations	43
Questions	45
The Relational Database Prevails!	47
Summary	48
Answers	48
3	
LET'S DESIGN a DATABASE!	49
The E-R Model	50
Normalizing a Table	56
What Is the E-R Model?	74
How to Analyze the E-R Model	74
Case 1: One-to-One Relationship	74
Case 2: One-to-Many Relationship	75
Case 3: Many-to-Many Relationship	75
Questions	76
Normalizing a Table	78
Questions	79
Steps for Designing a Database	81
Summary	81
Answers	82

4

<i>LET'S LEARN ABOUT SQL!</i>	85
Using SQL	86
Searching for Data Using a SELECT Statement	93
Using Aggregate Functions	98
Joining Tables	101
Creating a Table	103
SQL Overview	106
Searching for Data Using a SELECT Statement	106
Creating Conditions	107
Comparison Operators	107
Logical Operators	107
Patterns	108
Searches	108
Questions	109
Aggregate Functions	110
Aggregating Data by Grouping	110
Questions	111
Searching for Data	112
Using a Subquery	112
Using a Correlated Subquery	113
Questions	114
Joining Tables	114
Creating a Table	115
Inserting, Updating, or Deleting Rows	116
Creating a View	117
Questions	118
Summary	119
Answers	119

5

<i>LET'S OPERATE A DATABASE!</i>	125
What Is a Transaction?	126
What Is a Lock?	131
Database Security	138
Speeding Things Up with Indexing	143
Disaster Recovery	148
Properties of Transactions	153
Atomicity	153
Consistency	154
Isolation	155
Durability	159
When Disaster Strikes	161
Types of Failures	161
Checkpoints	161
Questions	162

Indexes	162
Questions	164
Optimizing a Query	164
Nested Loop	165
Sort Merge	166
Hash	166
Optimizer	167
Summary	167
Answers	167
6 Databases are Everywhere!	169
Databases in Use	175
Databases and the Web	177
Distributed Databases	183
Stored Procedures and Triggers	185
Databases on the Web	194
Using Stored Procedures	196
Questions	196
What Is a Distributed Database?	197
Horizontal Distribution	197
Vertical Distribution	198
Partitioning Data	198
Horizontal Partitioning	198
Vertical Partitioning	199
Preventing Inconsistencies with a Two-Phase Commit	199
Questions	201
Database Replication	201
Read-Only	201
Replication Enabled for All Servers	202
Further Application of Databases	202
XML	202
Object-Oriented Databases	203
Summary	205
Answers	205
Closing Remarks	205
APPENDIX	
FREQUENTLY USED SQL STATEMENTS	207
REFERENCES	209
INDEX	211

PREFACE

Databases are a crucial part of nearly all computer-based business systems. Some readers of this book may be considering introducing databases into their routine work. Others may have to actually develop real database-based business systems. The database is the technology that supports these systems behind the scenes, and its true nature is difficult to understand.

This book is designed so that readers will be able to learn the basics about databases through a manga story. At the end of each chapter, practice exercises are provided for confirmation and expanding the knowledge you've obtained. Each chapter is designed so that readers can gain an understanding of database technology while confirming how much they understand the contents.

The structure of this book is as follows.

Chapter 1 describes why we use databases. Why is a database necessary? What kind of difficulties will you have if you do not use a database? You will learn the background information that using a database requires.

Chapter 2 provides basic terminology. You'll learn about various database models and other terms relating to databases.

Chapter 3 explains how to design a database, specifically, a relational database, the most common kind.

Chapter 4 covers SQL, a language used to manage relational databases. Using SQL allows you to easily manage your data.

Chapter 5 explains the structure of the database system. Since a database is a system through which many people share data, you will learn how it can do so safely.

Chapter 6 provides descriptions of database applications. You'll learn how Web-based and other types of database systems are used.

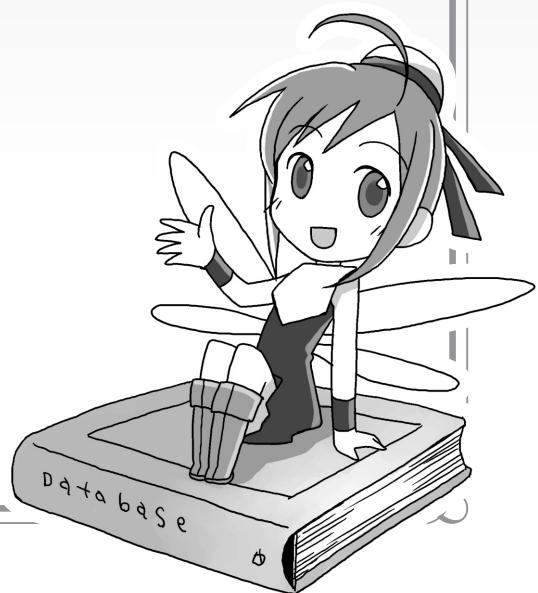
This book was published thanks to the joint efforts of many people: Shoko Azuma for cartoons, TREND-PRO for production, and Ohmsha for planning, editing, and marketing. I extend my deep gratitude to all those concerned.

I hope that this book is helpful to all readers.

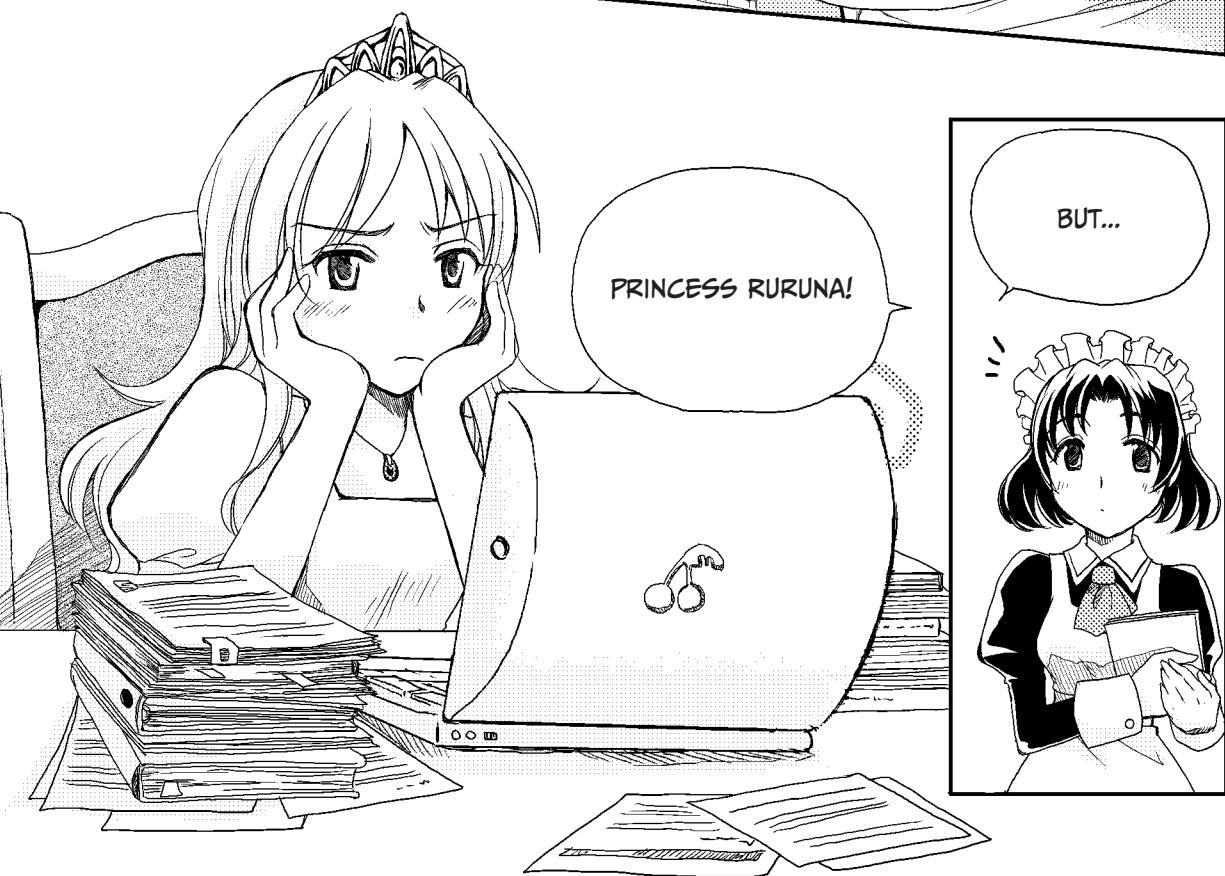
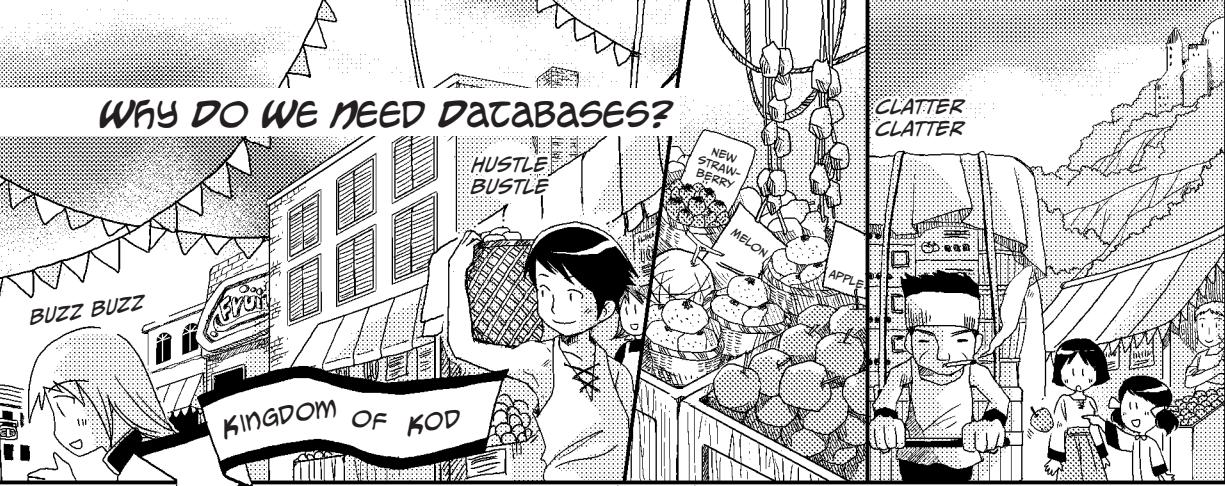
MANA TAKAHASHI

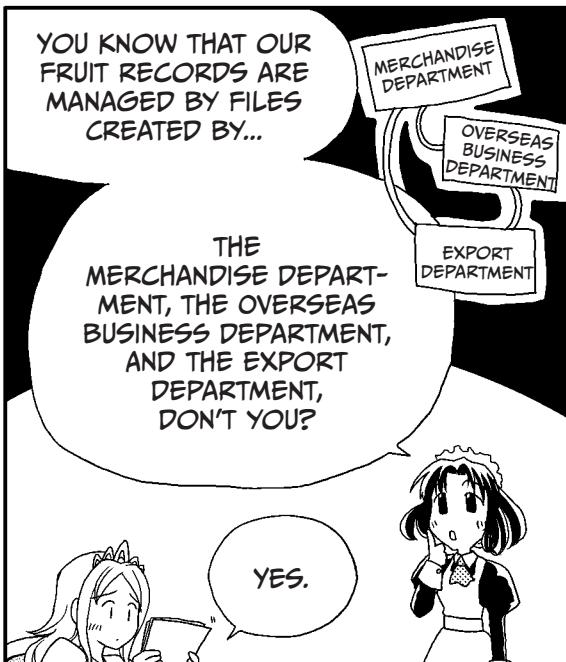
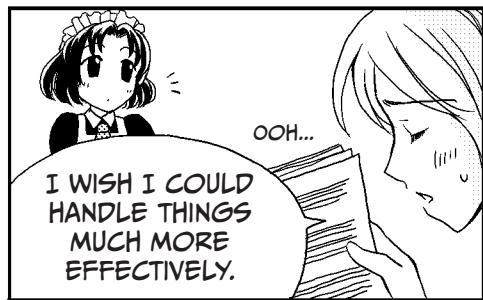
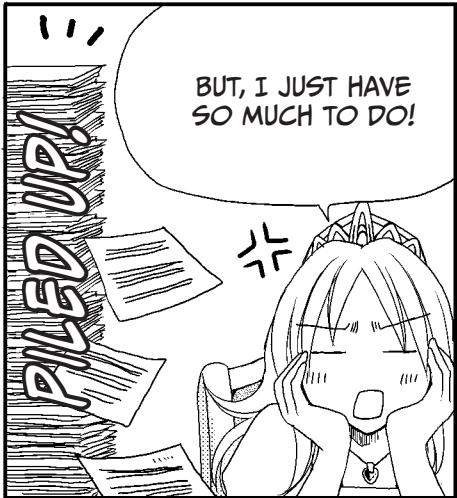
1

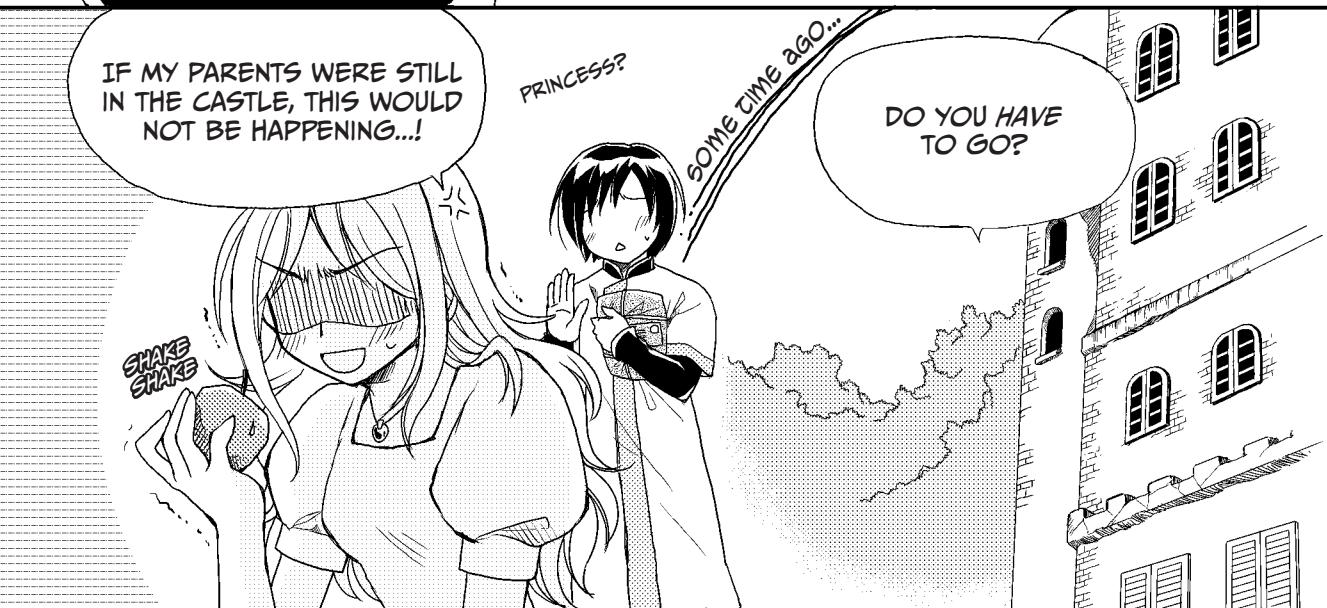
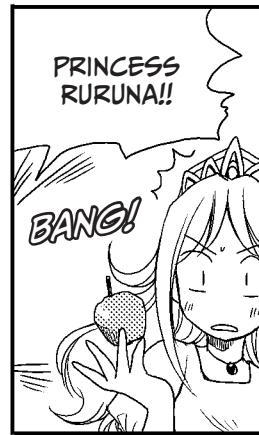
WHAT IS A DATABASE?

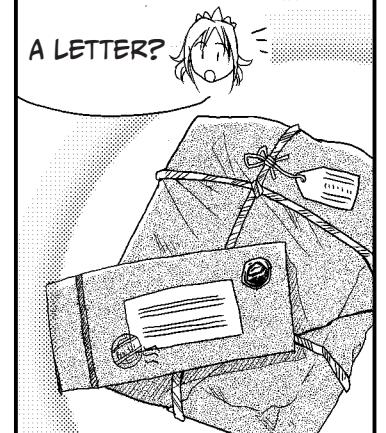
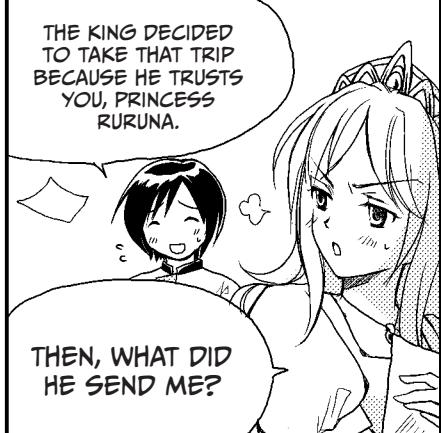
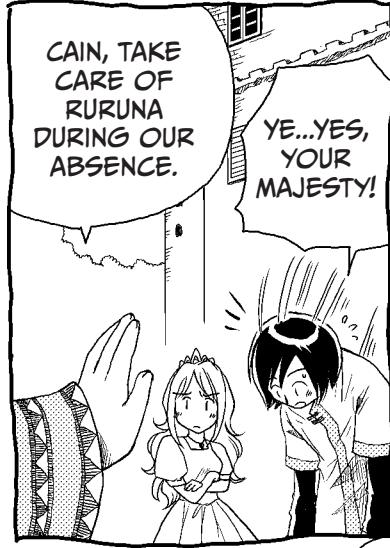
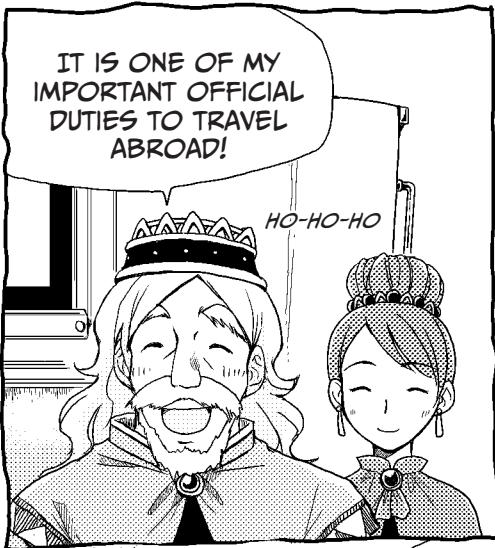


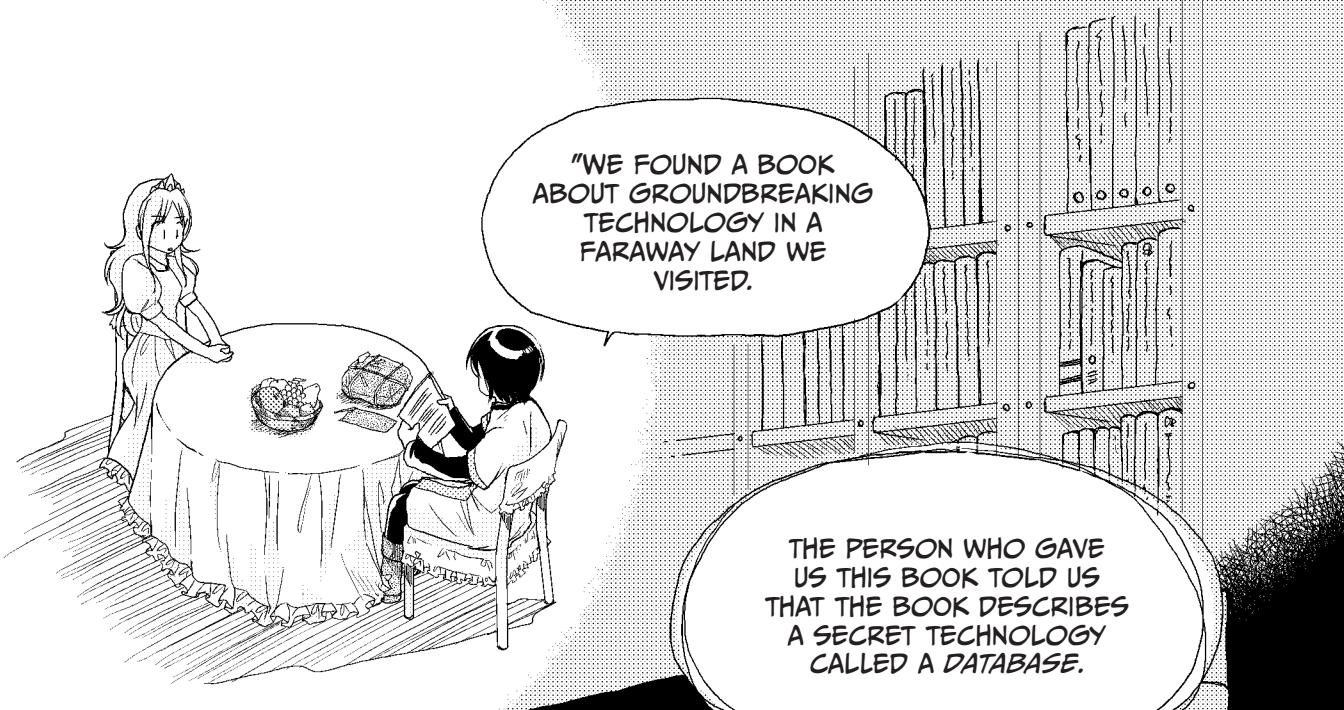
WHY DO WE NEED DATABASES?





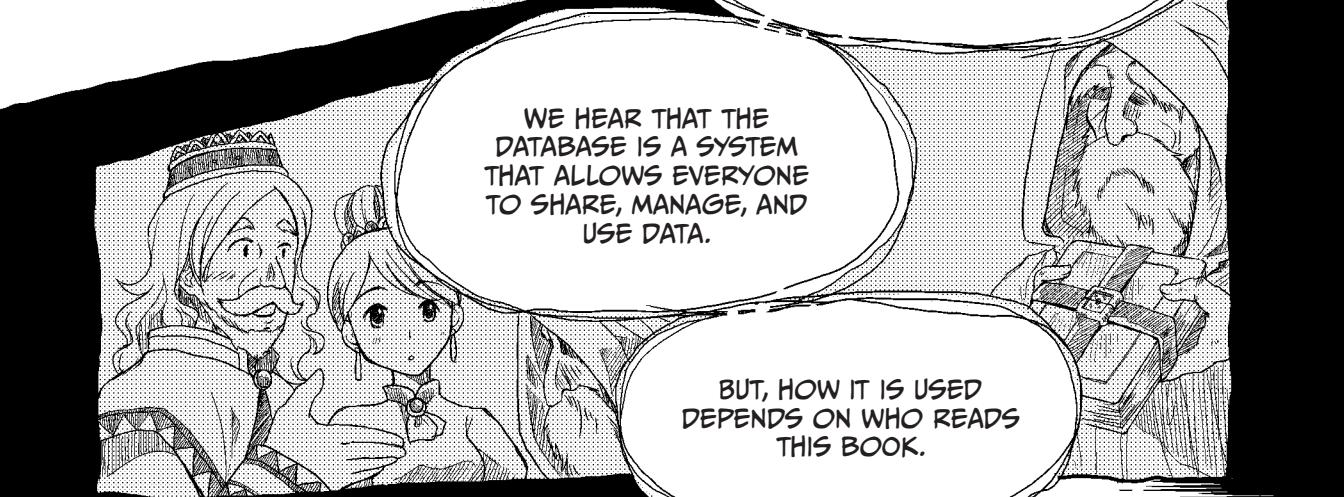






"WE FOUND A BOOK
ABOUT GROUNDBREAKING
TECHNOLOGY IN A
FARAWAY LAND WE
VISITED.

THE PERSON WHO GAVE
US THIS BOOK TOLD US
THAT THE BOOK DESCRIBES
A SECRET TECHNOLOGY
CALLED A DATABASE.



WE HEAR THAT THE
DATABASE IS A SYSTEM
THAT ALLOWS EVERYONE
TO SHARE, MANAGE, AND
USE DATA.

BUT, HOW IT IS USED
DEPENDS ON WHO READS
THIS BOOK.

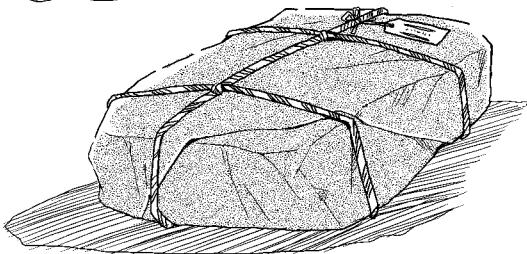


THE PERSON GAVE THIS
BOOK TO US BELIEVING
THAT THE KINGDOM OF
KOD WOULD USE IT IN A
PEACEFUL MANNER.

RURUNA...

OPEN THIS BOOK,
AND USE IT FOR
THE BETTERMENT
OF OUR COUNTRY!"

OH, PLEASE! YOU DON'T
KNOW ANYTHING ABOUT
HOW STRESSED OUT I AM
FEELING!



WHAT IN THE HECK???



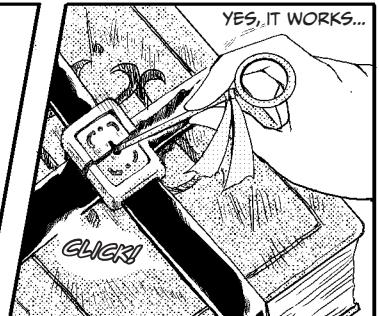
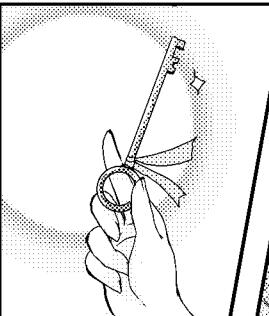
IT'S LOCKED.

I CAN'T OPEN IT.

HMM...

IS THIS THE
KEY FOR IT?

IT WAS IN THE ENVELOPE...



YES, IT WORKS...

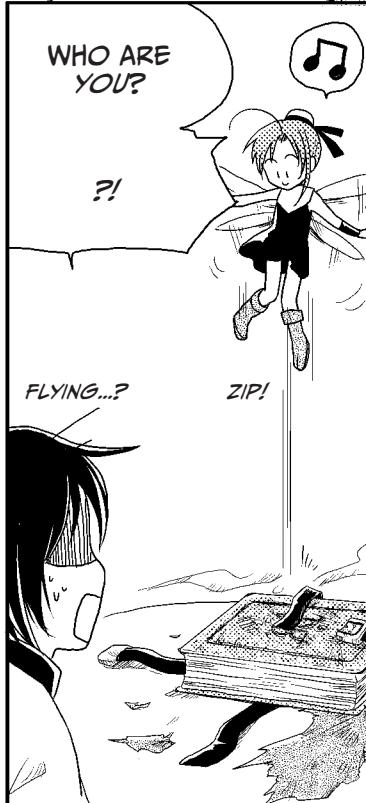
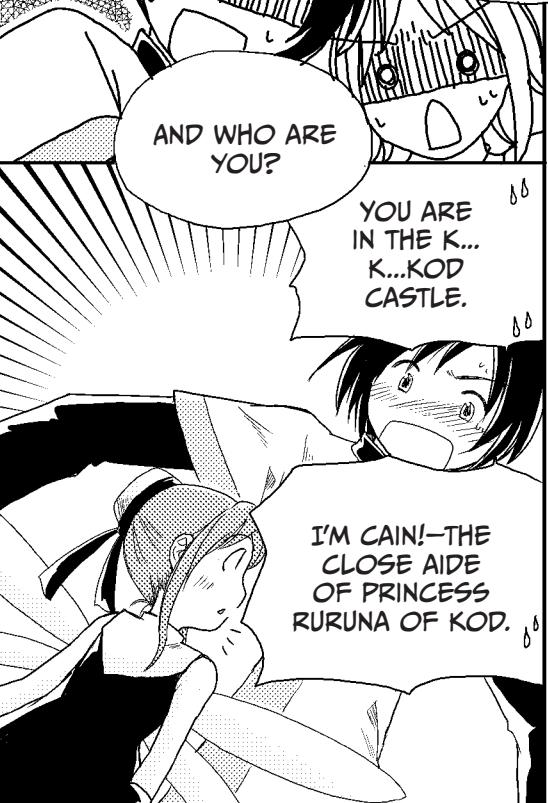
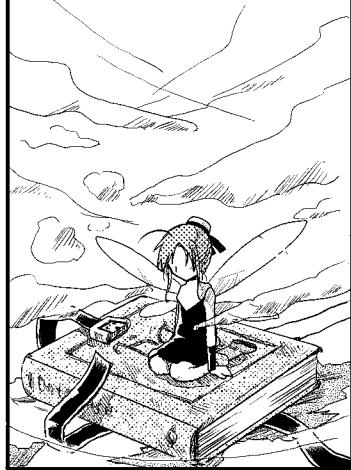
KA-BOOM!

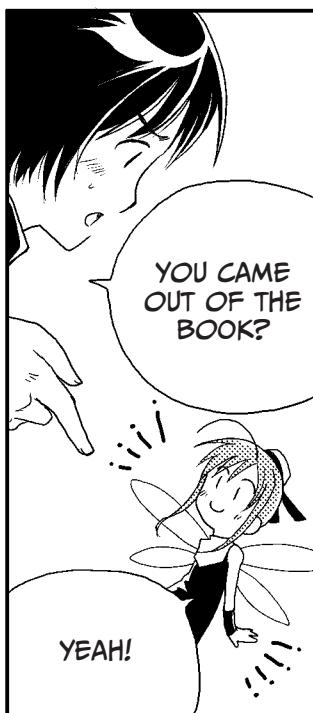
AHHHH!!

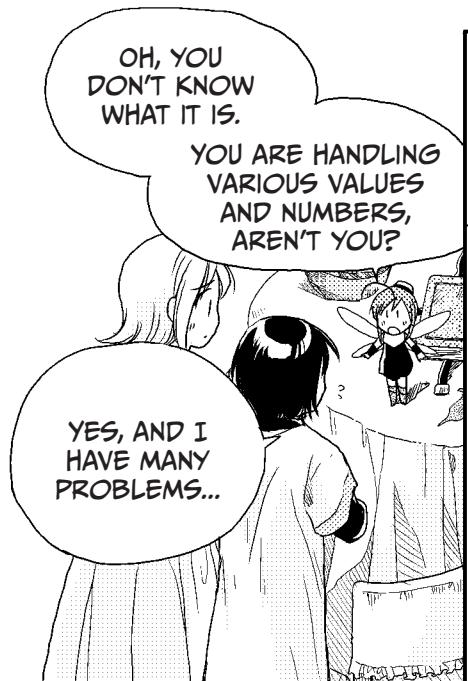
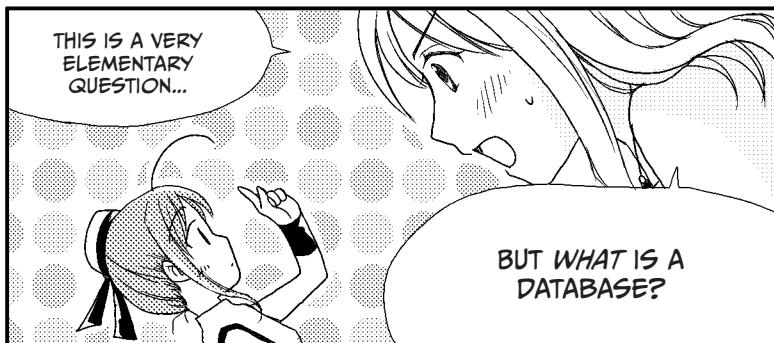
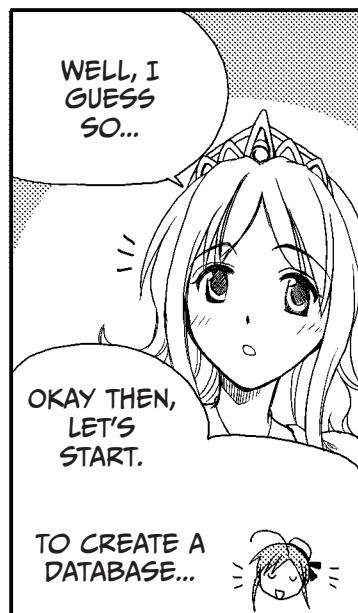
?!
HACK!

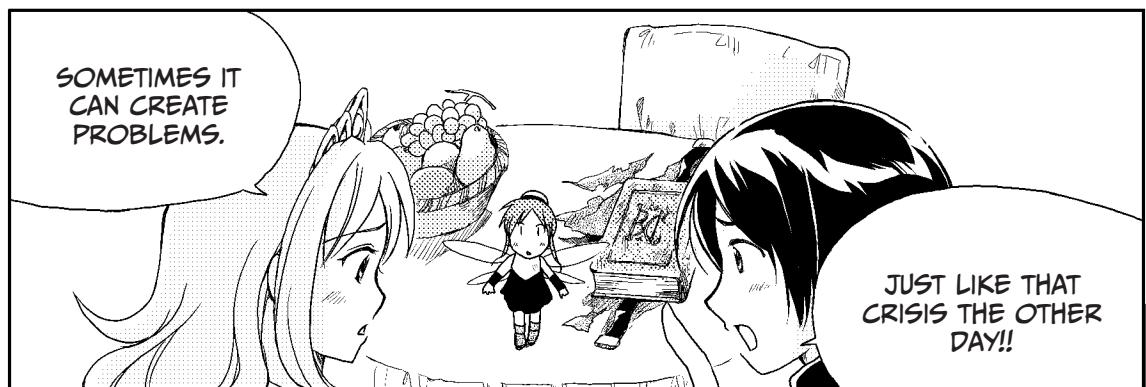
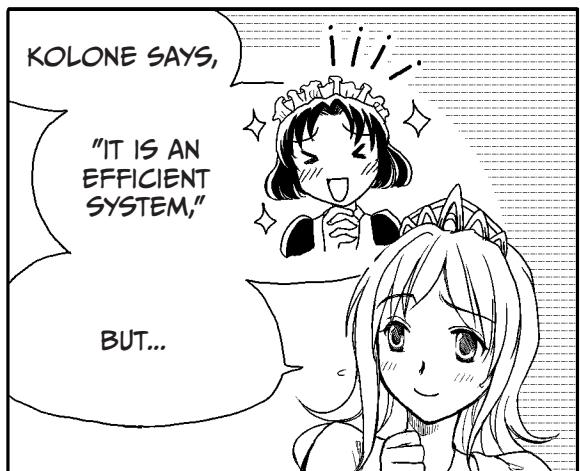
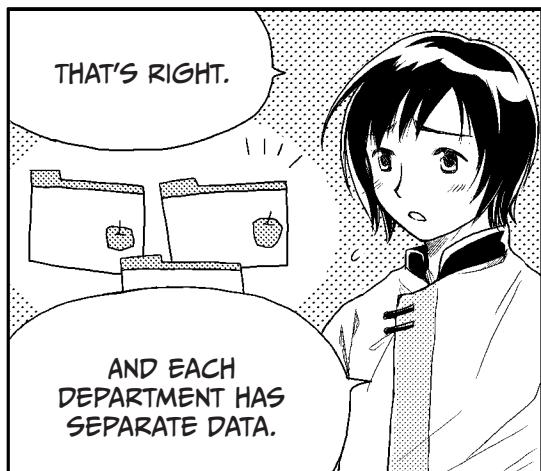
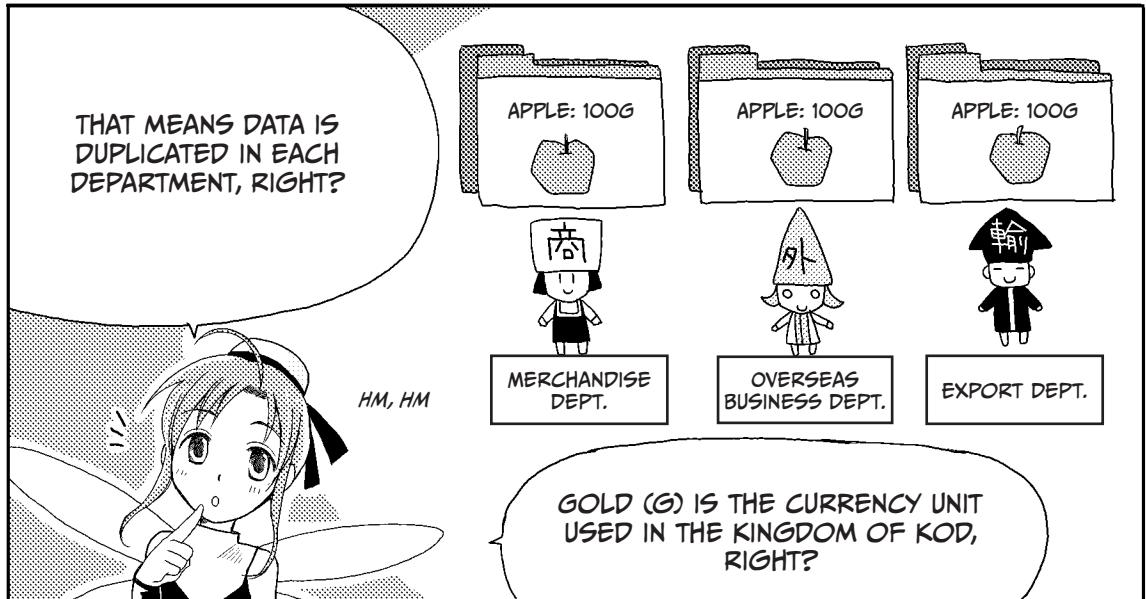
COUGH COUGH

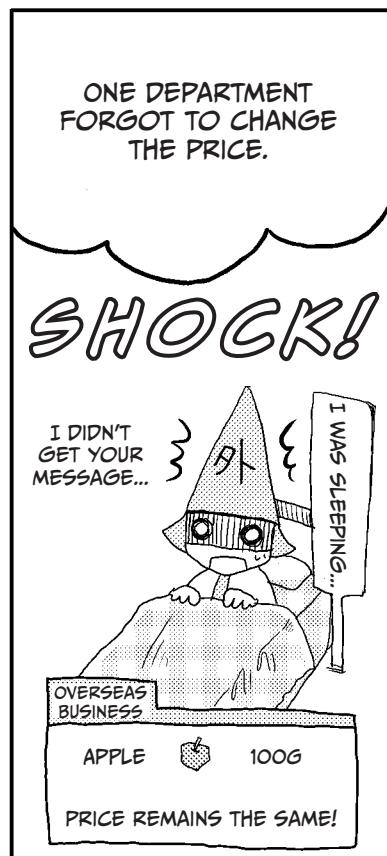
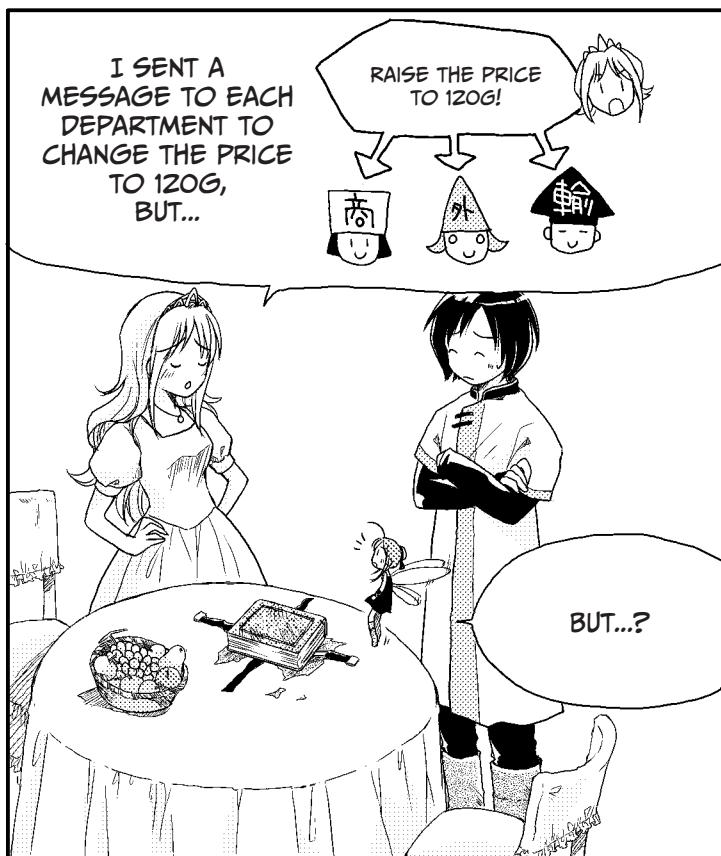
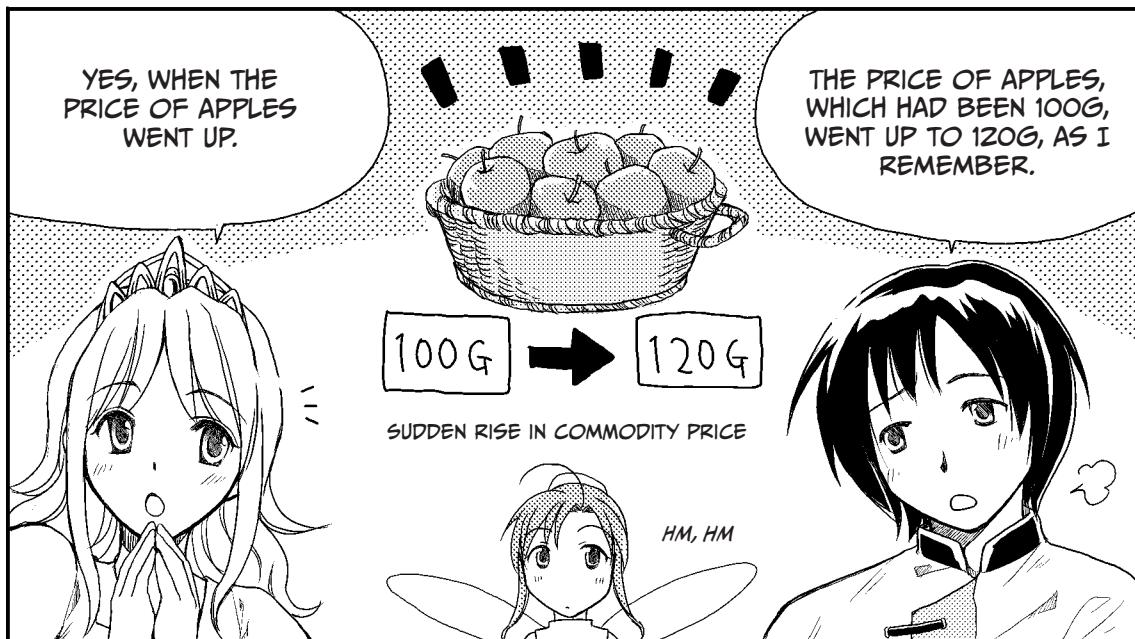


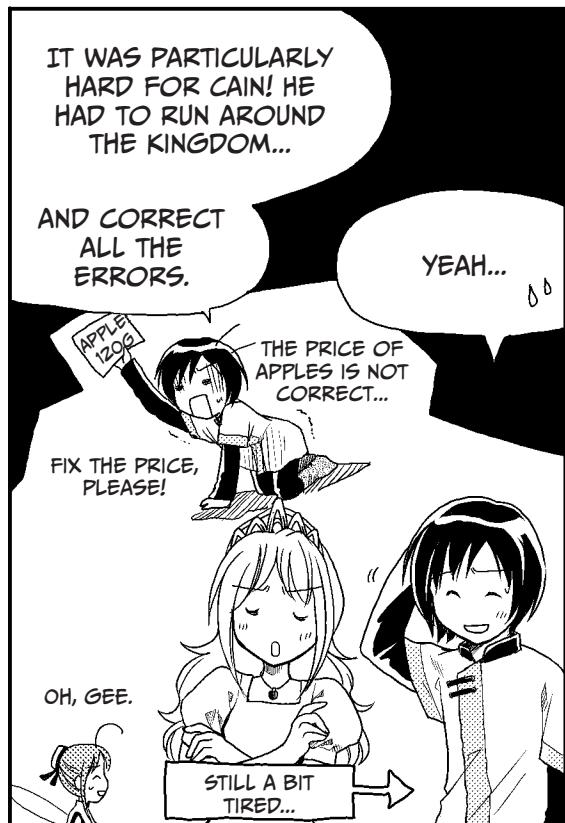
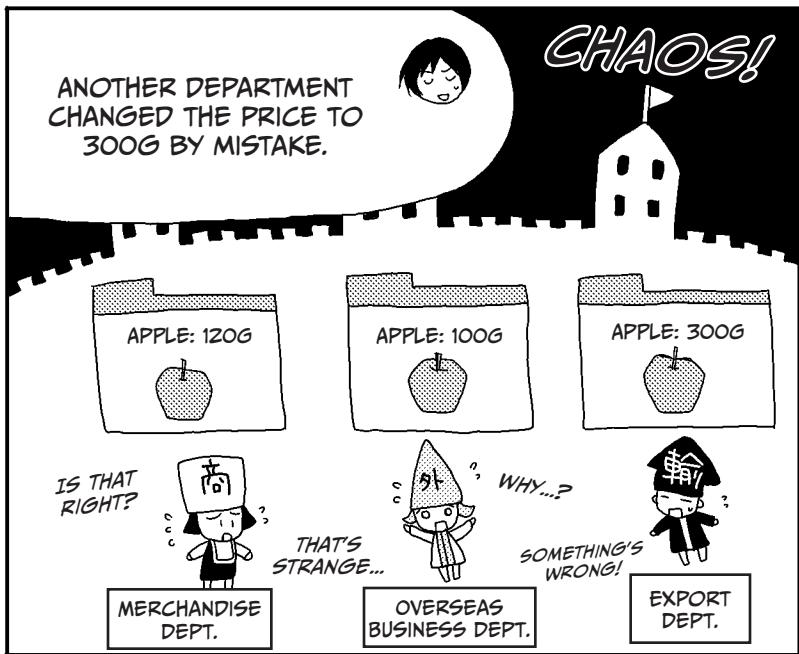


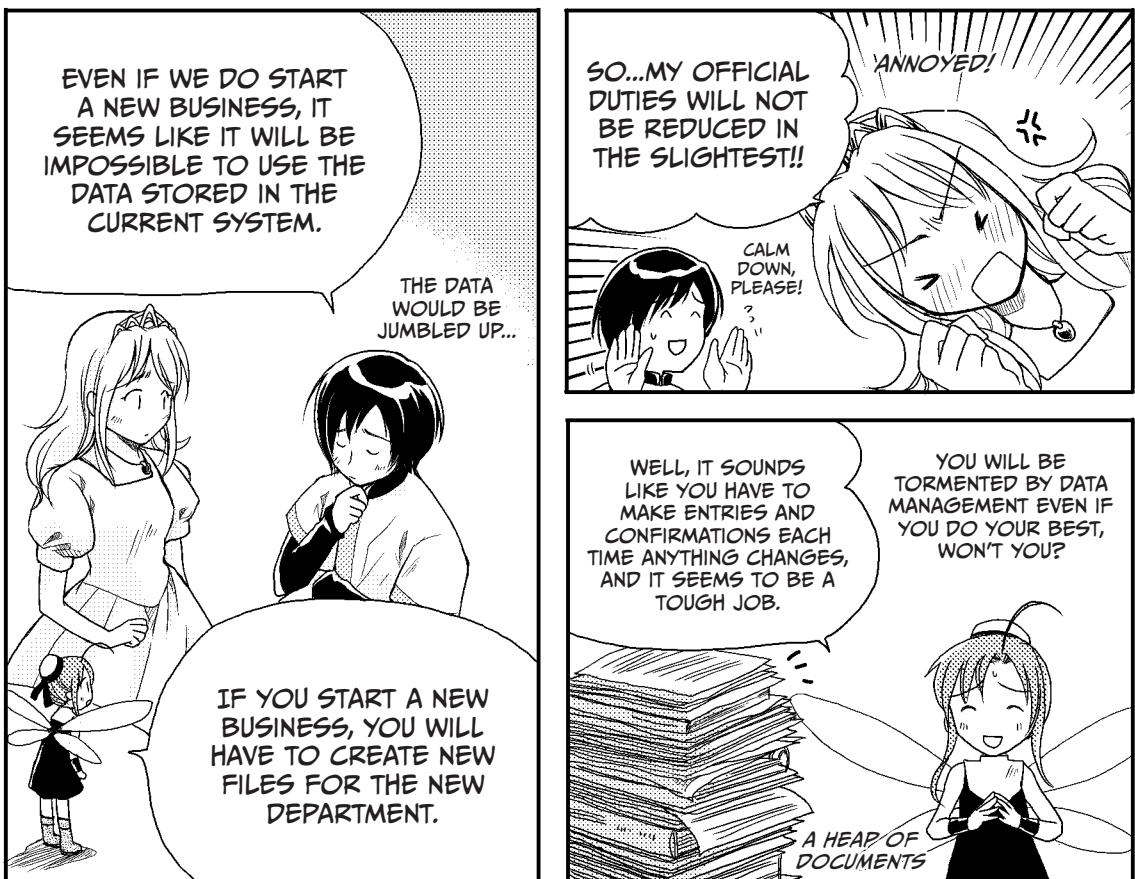


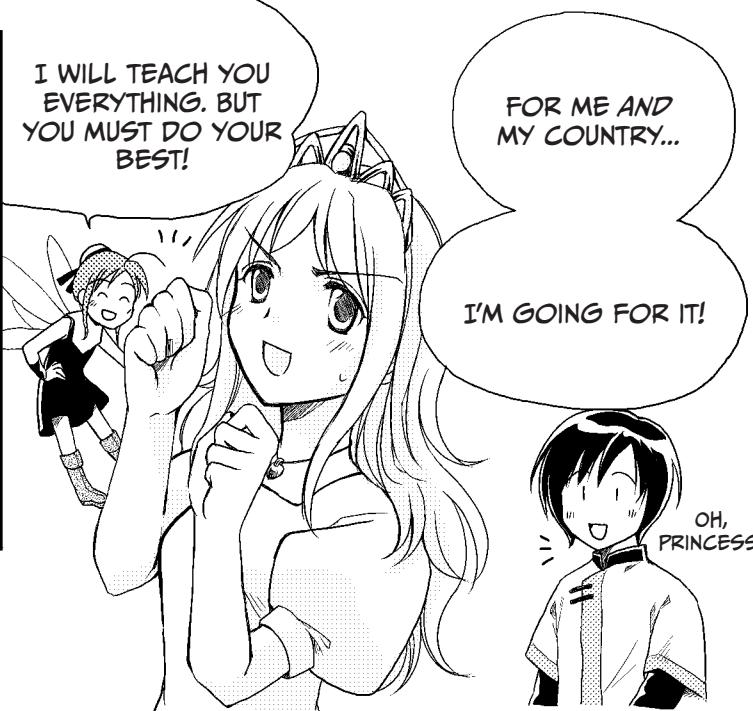
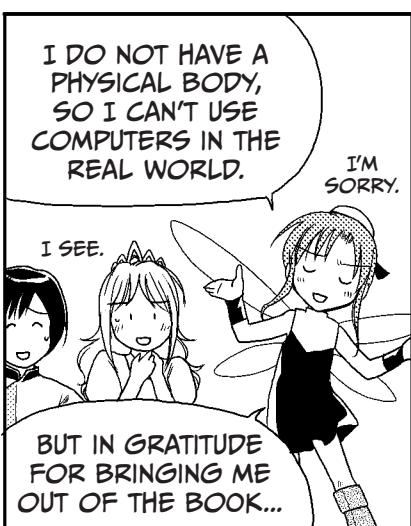
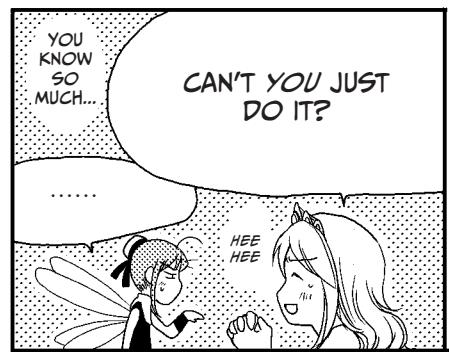
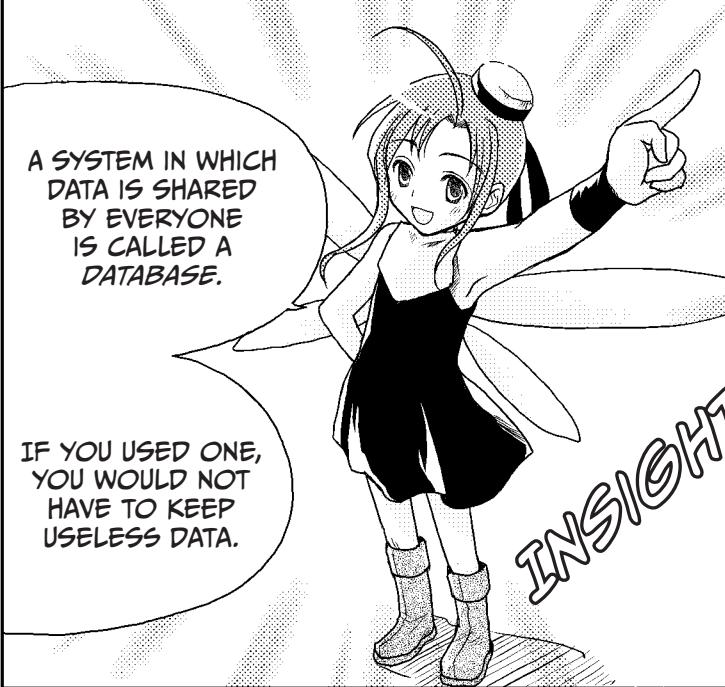










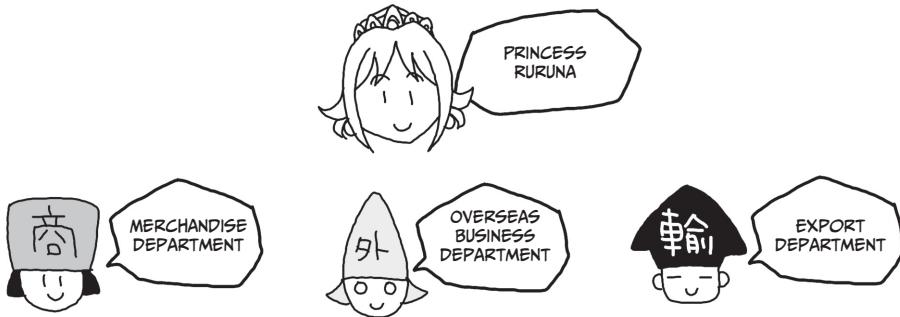


WHAT'S UP IN THE KINGDOM?



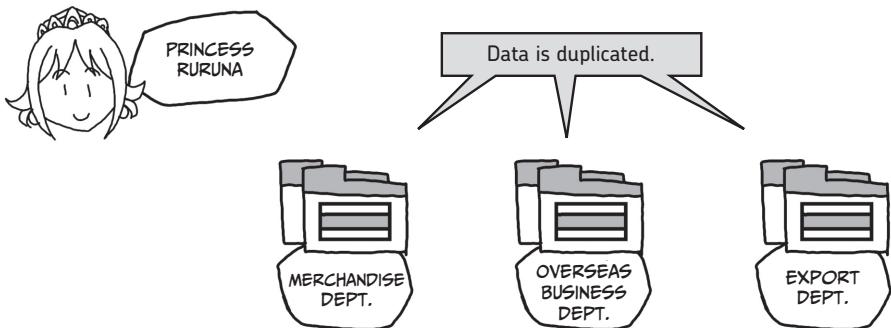
The Kingdom of Kod currently uses a file-based system to manage its data. But it seems that the current system has a few problems. What are they, in particular? Let's look at the system in detail.

The Kingdom currently has three departments: the Merchandise Department, the Overseas Business Department, and the Export Department. The Merchandise Department keeps track of all fruit produced in the country, the Overseas Business Department manages the foreign countries that are the Kingdom's business partners, and the Export Department keeps records of the amount of fruit the Kingdom exports.

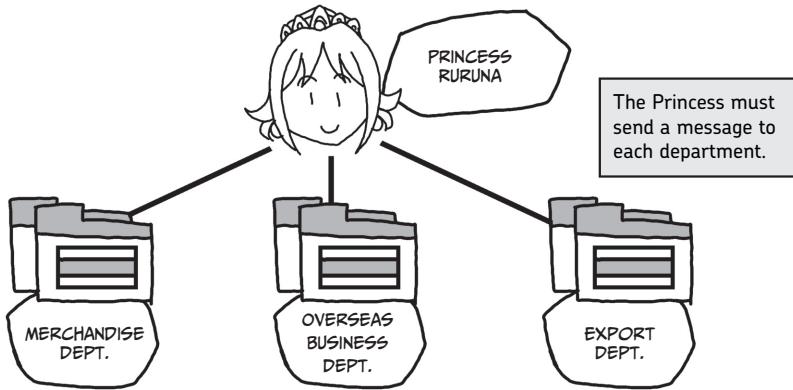


DATA IS DUPLICATED

Princess Ruruna isn't satisfied with the current system. But why not? Each department in the Kingdom manages data independently. For example, the Merchandise Department and the Export Department each create files to manage fruit data. Therefore, data is duplicated needlessly across the departments. Each department must enter the data, store the data, then print receipts for confirmation, all of which is a waste. In addition, data trapped in one particular department is never shared effectively with the other departments.



But that's not all. The system also creates problems when someone needs to change the data. For example, let's assume that the price of apples changes. To deal with this, Princess Ruruna must notify every department individually that the price of apples has changed. Isn't that inconvenient?



DATA CAN CONFLICT

It may seem easy enough to notify each department that the price of apples has changed, but it can create a new set of problems. Let's say that Princess Ruruna does notify the three departments that the price of apples has changed. However, the Overseas Business Department may forget to change the price, or the Export Department might change the price to 300G instead of 120G. These kinds of errors result in conflicting data between departments, which causes the content of the file systems to differ from the conditions of the real world. What a pain!



MERCHANTISE DEPARTMENT OVERSEAS BUSINESS DEPARTMENT EXPORT DEPARTMENT

Product name	Unit price
Melon	800G
Strawberry	150G
<i>Apple</i>	<i>120G</i>
Lemon	200G

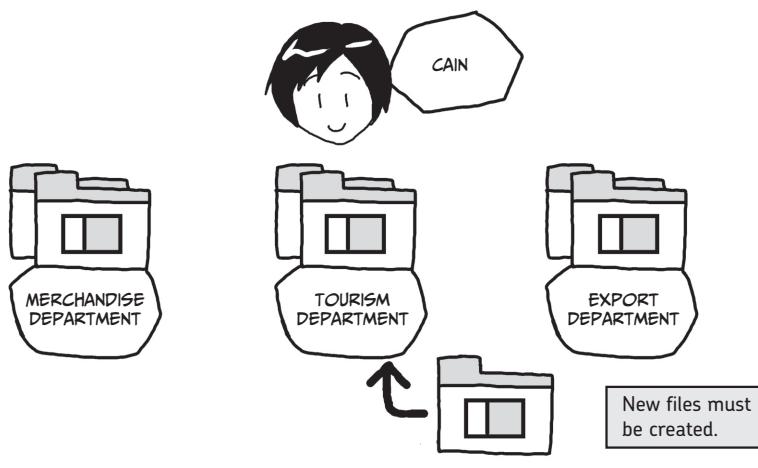
Product name	Unit price
Melon	800G
Strawberry	150G
<i>Apple</i>	<i>100G</i>
Lemon	200G

Product name	Unit price
Melon	800G
Strawberry	150G
<i>Apple</i>	<i>300G</i>
Lemon	200G

DATA IS DIFFICULT TO UPDATE

The current system not only creates conflicting data, but it also makes it difficult to respond to changes in business. For example, let's say that the King wants to launch a new Tourism Department. When a tour guide conducts a tour of the orchards and discusses the Kingdom's fruit sales, the guide will want to use the most up-to-date sales figures.

But, unfortunately, the current system does not necessarily allow the departments to access each other's data, since the files are kept independently. To manage a new tourism business, Princess Ruruna will have to make copies of all the relevant files for the Tourism Department!



FILE FOR MERCHANTISE DEPT.

Product name	Unit price	
Melon	800G	
Strawberry	150G	
Apple	120G	
Lemon	200G	

FILE FOR TOURISM DEPT.

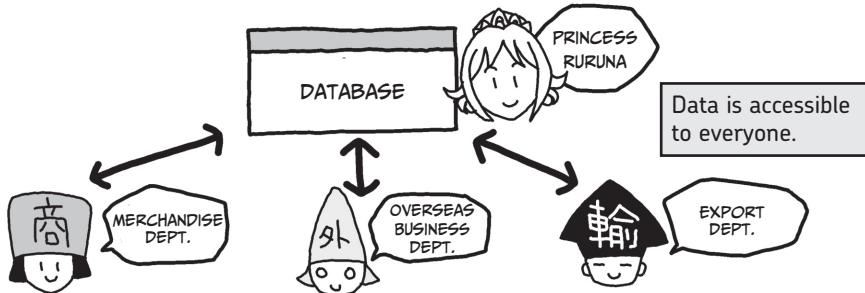
Product name	Unit price	
Melon	800G	
Strawberry	150G	
Apple	120G	
Lemon	200G	

This, in turn, increases the amount of duplicated data created when a new department starts. Considering these weaknesses, the current system is not efficient. It makes it difficult to start new projects and respond to environmental changes.

a DATABASE—that's our solution!



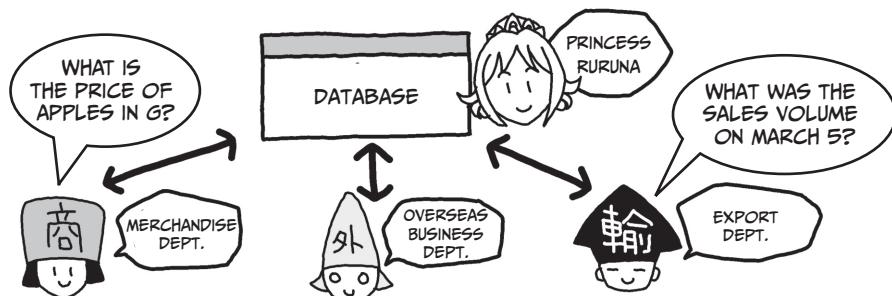
Well, why is this system so inefficient? The problems all stem from separate and independent data management. What should Ruruna and Cain do? That's right—they should create a database! They must unify the management of data for the entire Kingdom. I will show you how to do this in the next chapter.



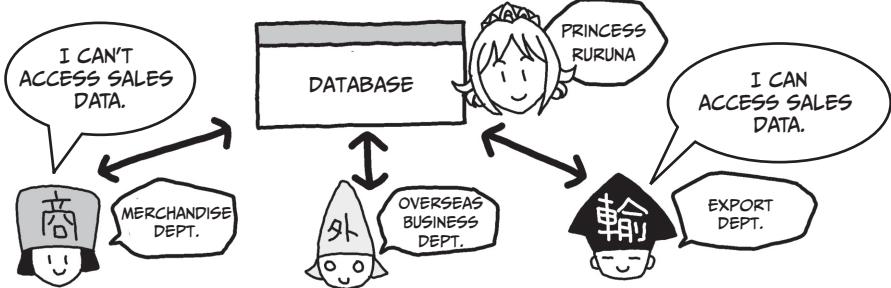
Uniform data management ensures that each department has the correct information, because each department sends a query to a single source of data. What an efficient system it is! It prevents data conflicts, and it also eliminates duplicated data, allowing for easy introduction and integration of new departments.

HOW TO USE A DATABASE

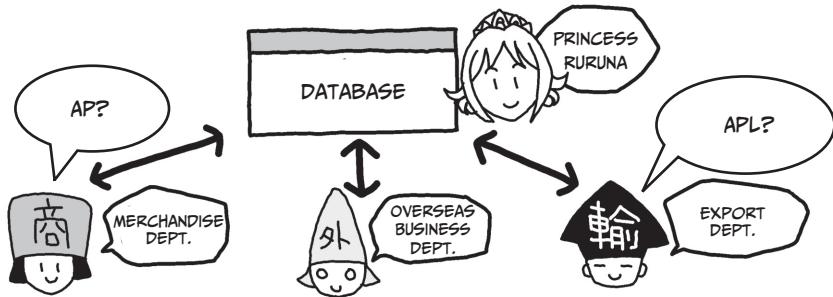
To introduce and operate a database, you must understand its unique challenges. First, the database will be used by many people, so you'll need a way for them to easily input and extract data. It needs to be a method that is easy for everybody to use.



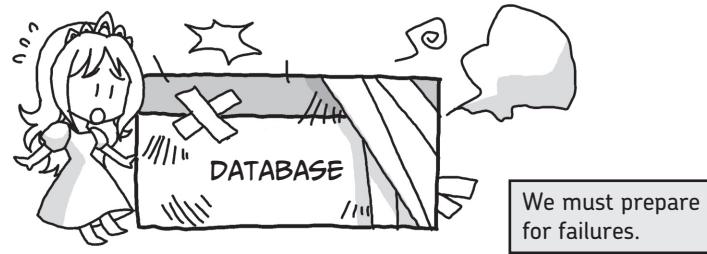
The new system also presents some risks—for example, it may make it possible for users to steal or overwrite important information like salary data, which is confidential and should be protected by an access restriction. Or, for example, only the Export Department should have access to sales data. Setting up database security and permissions is important when designing a system.



The new system can have other problems, too. The database can be used by many people at one time. Assume that someone in the Overseas Business Department and someone in the Export Department both try to change the name of a fruit at the same time—the former, from *Apple* to *AP*, and the latter, from *Apple* to *APL*. If they do this, what will happen to the product name? For a database that will be used by many people, this kind of problem must be considered.



You also need to be careful not to lose any data. Furthermore, the system may go down or a hard disk could fail, causing data to be corrupted. The database must have mechanisms to recover from these common kinds of failures.



In addition, since the database will hold a large amount of data, you must be able to perform searches at high speeds. The new system must have the power to handle that.

Let's start studying databases together with Princess Ruruna and Cain to learn how to solve these problems. Onward to Chapter 2!

SUMMARY



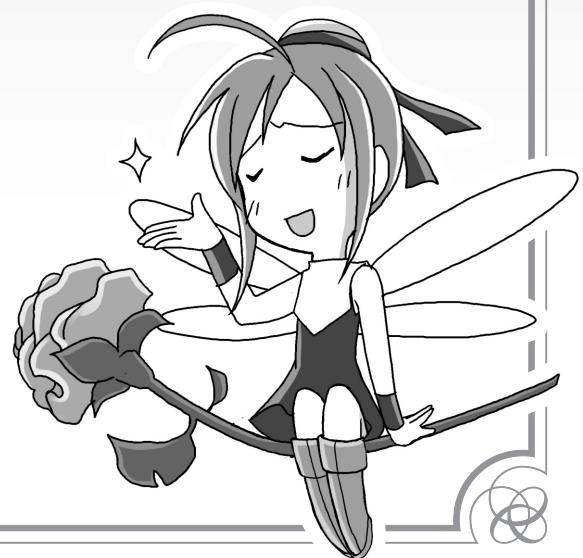
- File-based management can create conflicting data and data duplication.
- A database allows you to share data easily and prevents conflicting and duplicated data.

USING SOFTWARE TO MANAGE DATABASES

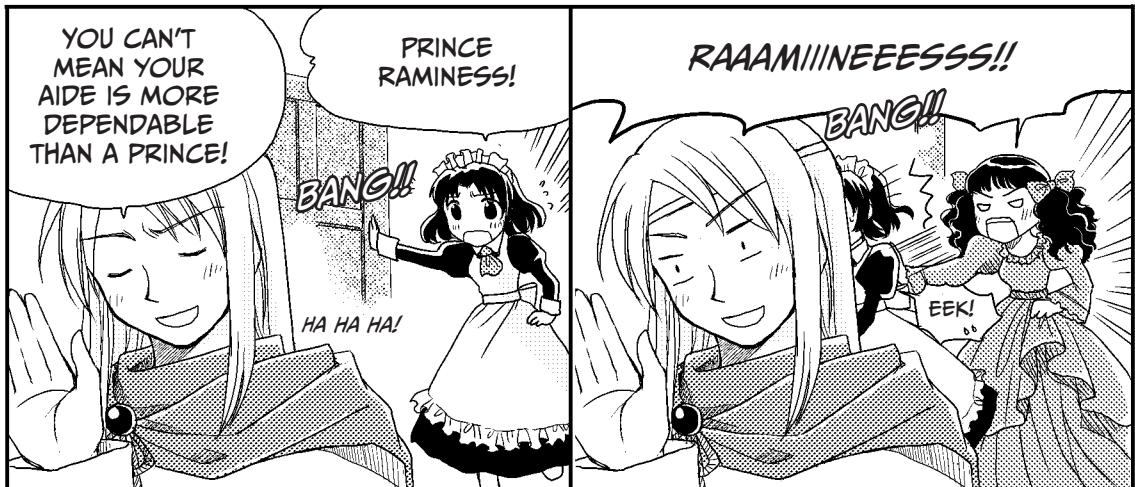
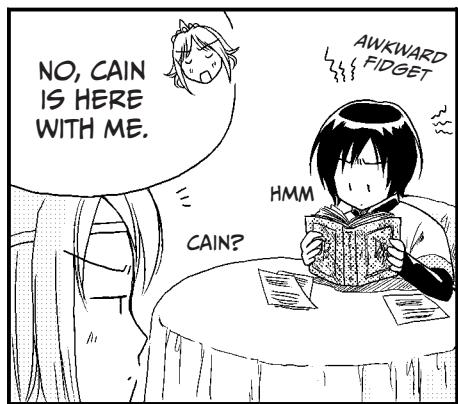
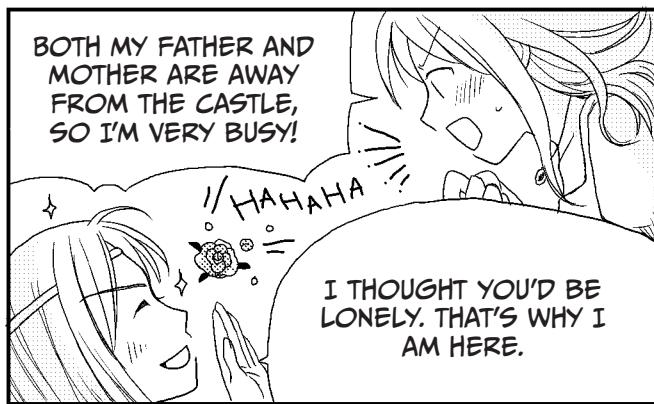
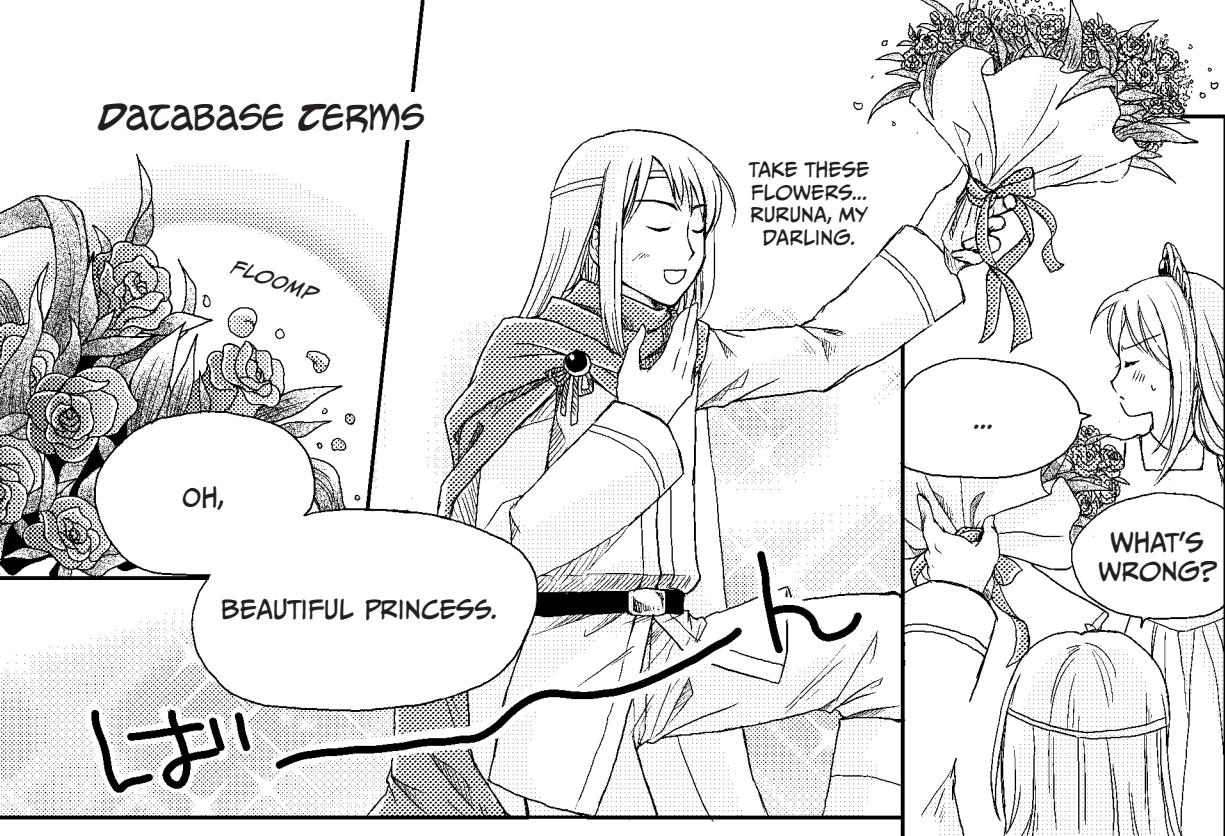
The database we are going to study is managed by software called a database management system (DBMS). A DBMS has many useful functions—it allows you to do things like input data into a database, prevent conflicting data, and retrieve a large amount of data at high speed. Thanks to our DBMS, the database can be used by many people simultaneously. In addition, a DBMS can protect the security of the database—for example, it allows the database to operate properly even if a failure occurs. In addition, the DBMS provides an easy-to-use interface between the database and its users. We'll study databases and the functions of a DBMS in the next chapter.

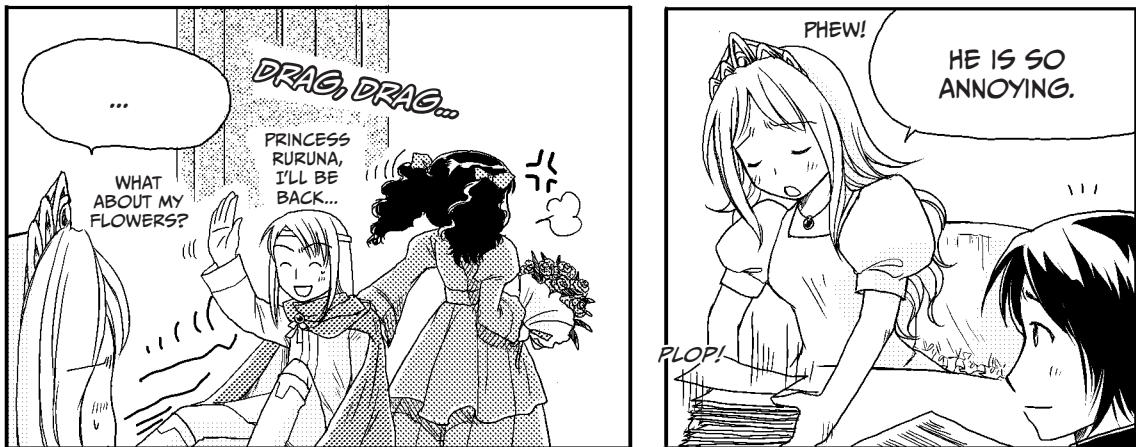
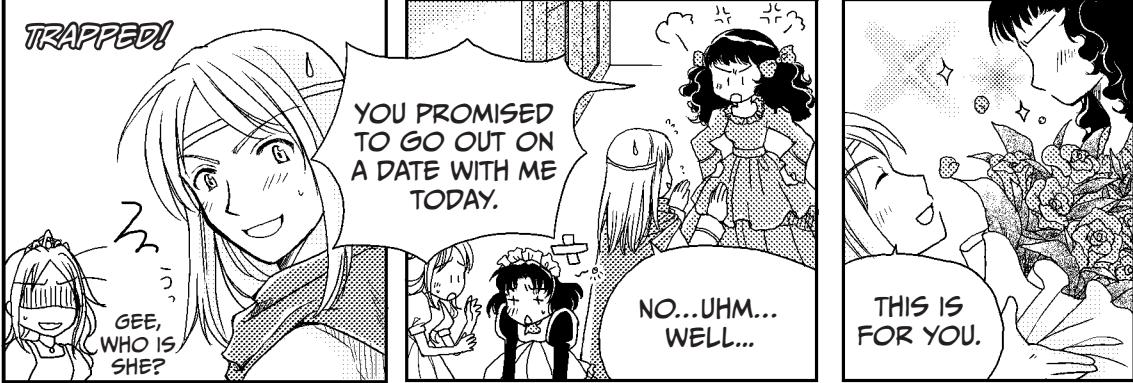
2

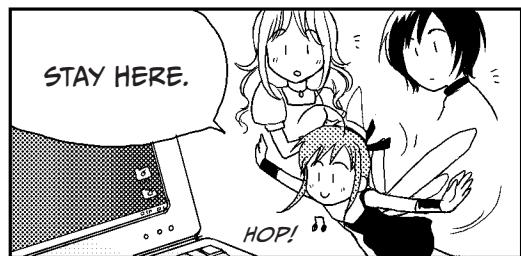
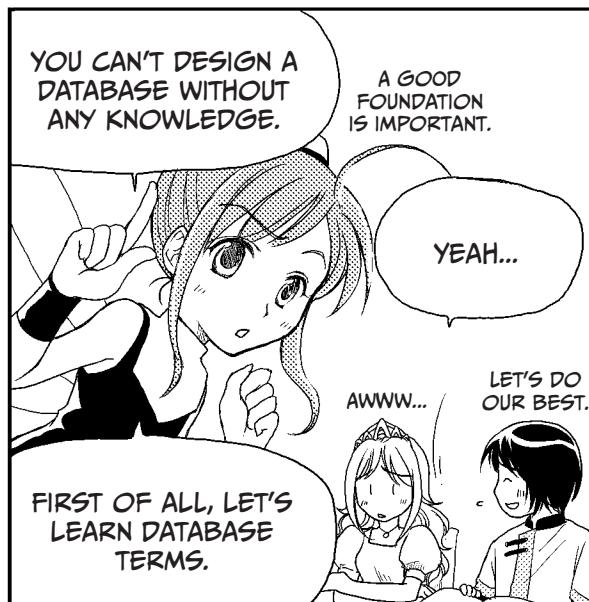
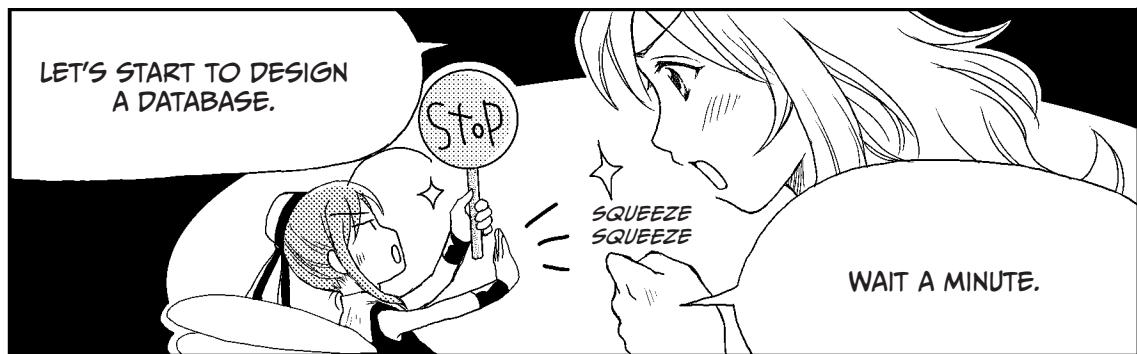
WHAT IS A RELATIONAL DATABASE?

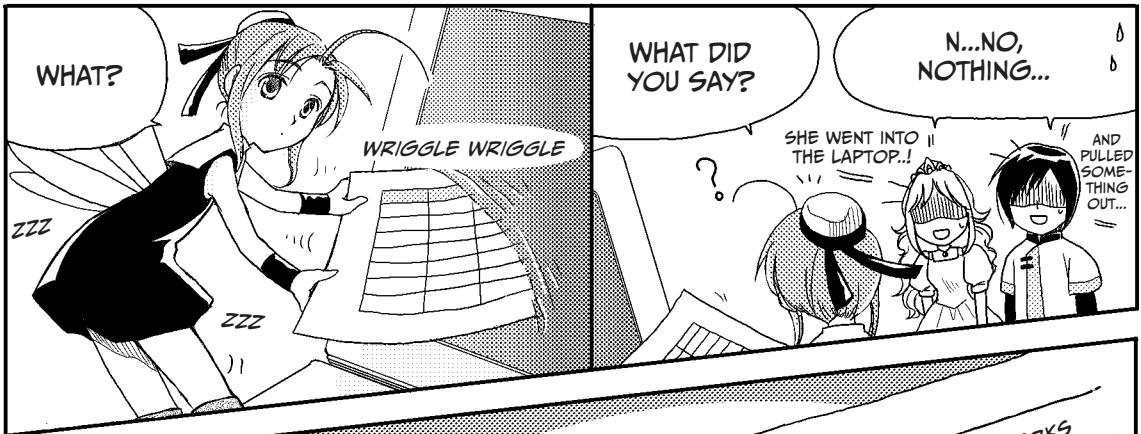


DATABASE TERMS

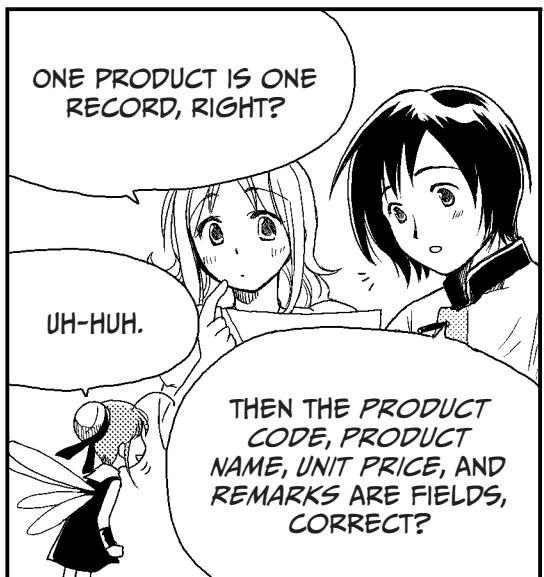
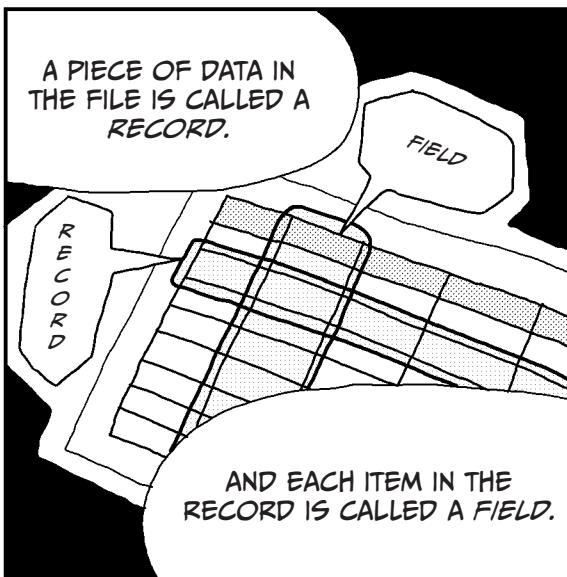


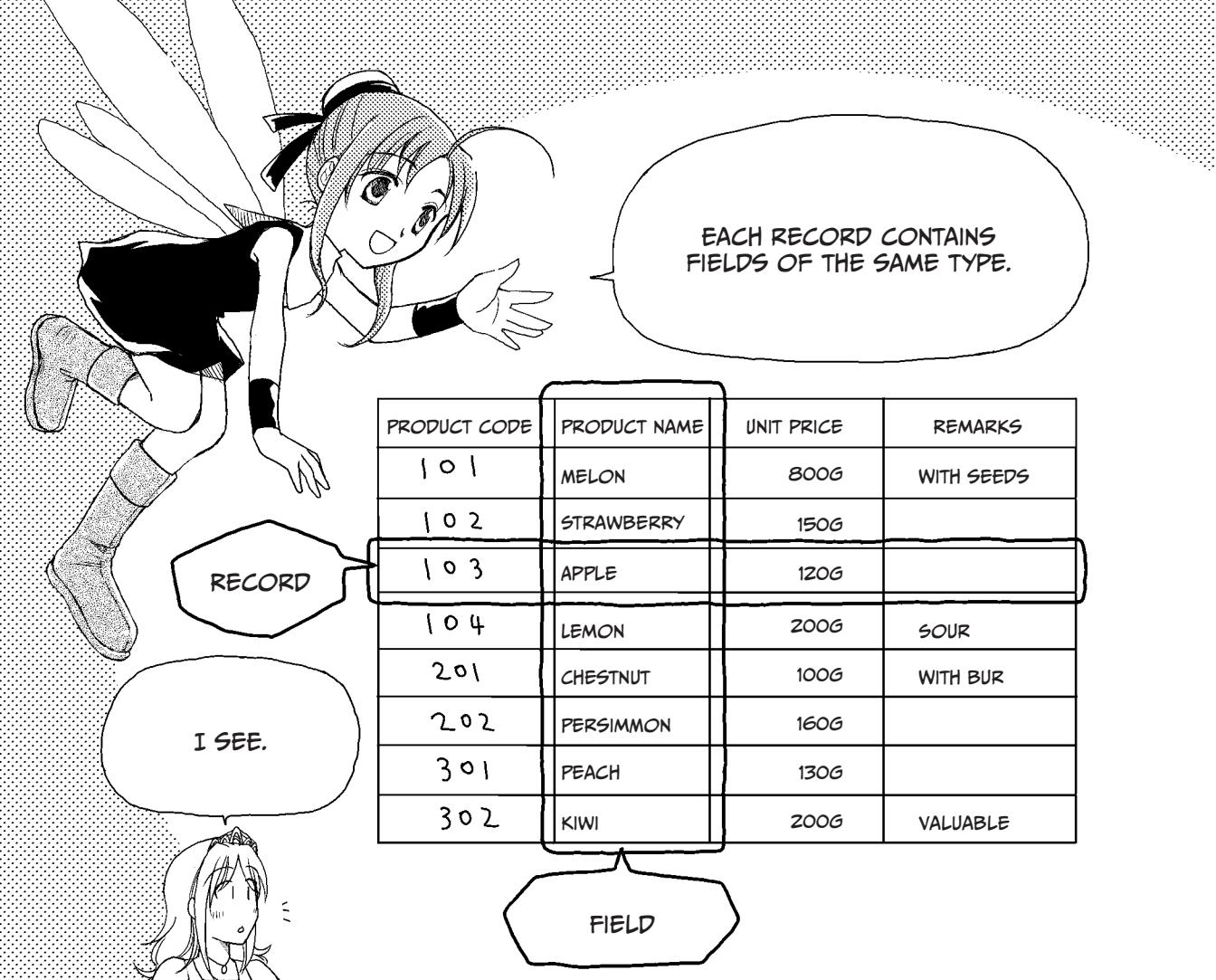






PRODUCT CODE	PRODUCT NAME	UNIT PRICE	REMARKS
101	MELON	800G	WITH SEEDS
102	STRAWBERRY	150G	
103	APPLE	120G	
104	LEMON	200G	
201	CHESTNUT	100G	
202	PERSIMMON	160G	





EACH RECORD CONTAINS
FIELDS OF THE SAME TYPE.

RECORD

I SEE.

FIELD

FOR EXAMPLE,
PRODUCT CODE
IS A THREE-DIGIT
VALUE...

AND PRODUCT NAME
IS TEN CHARACTERS
OR LESS.

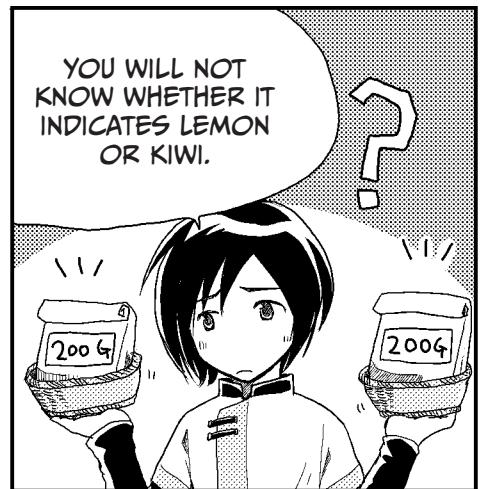
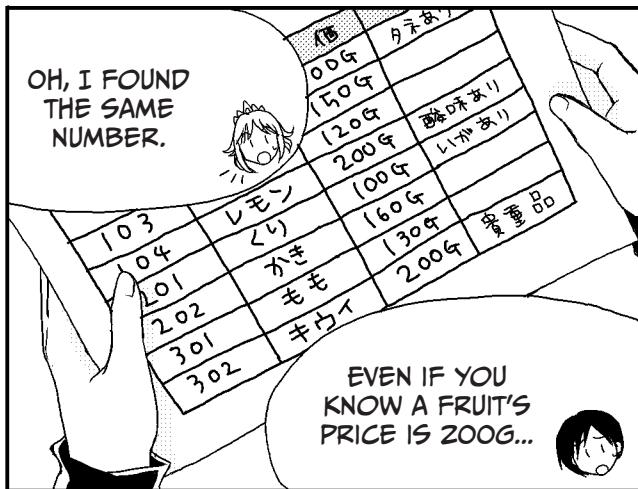
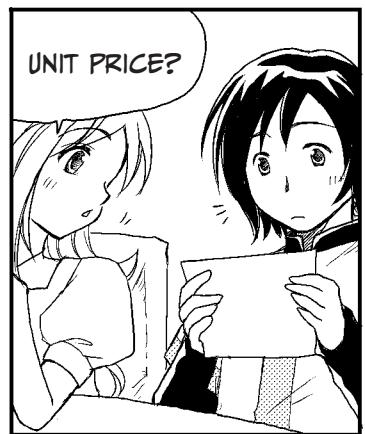
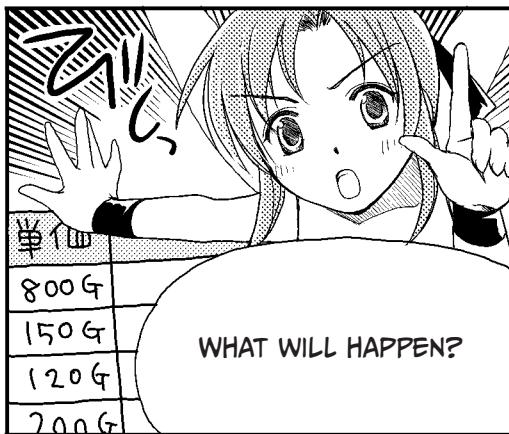
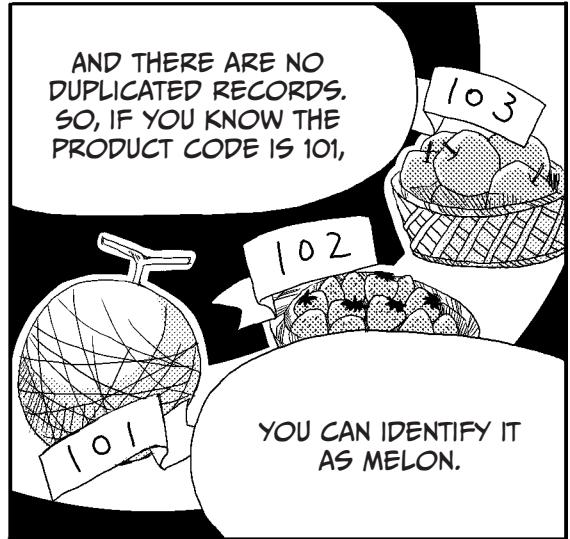
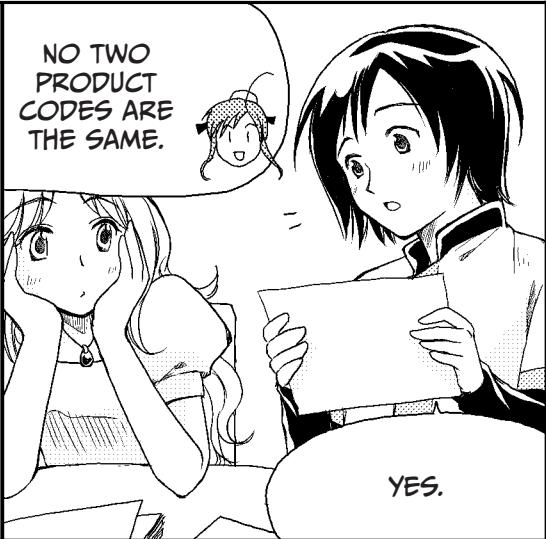
PRODUCT CODE	PRODUCT NAME
1 0 1	MELON
1 0 2	STRA
1 0 3	APPL
1 0 4	LEMON
2 0 1	CHESTNUT
2 0 2	PERSIMMON

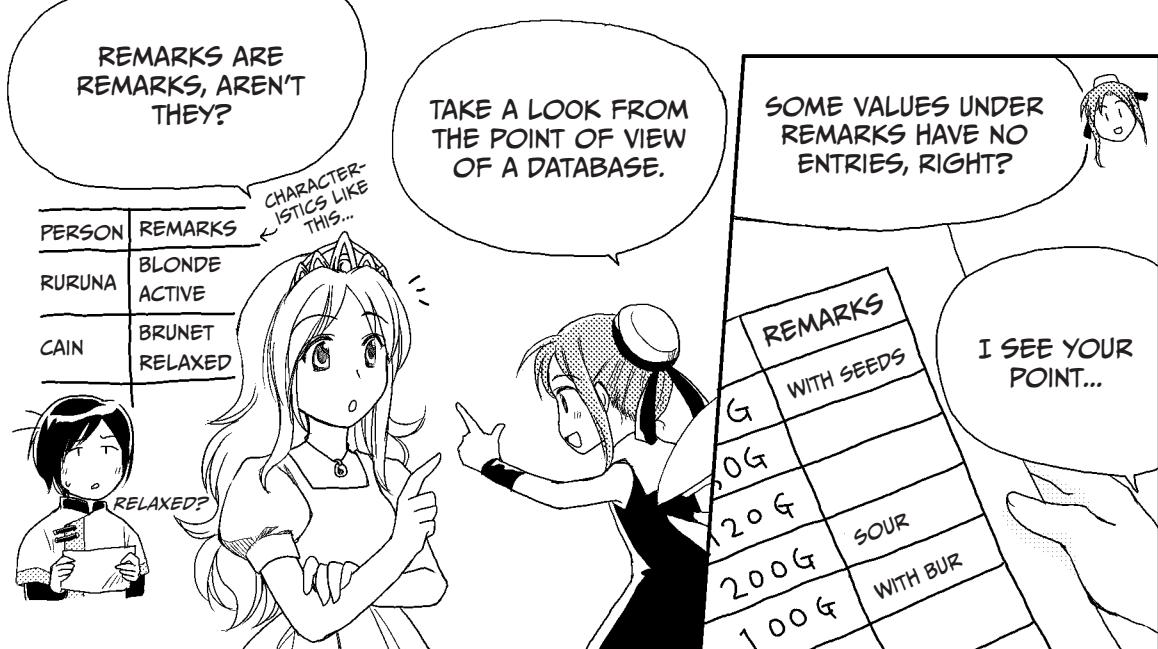
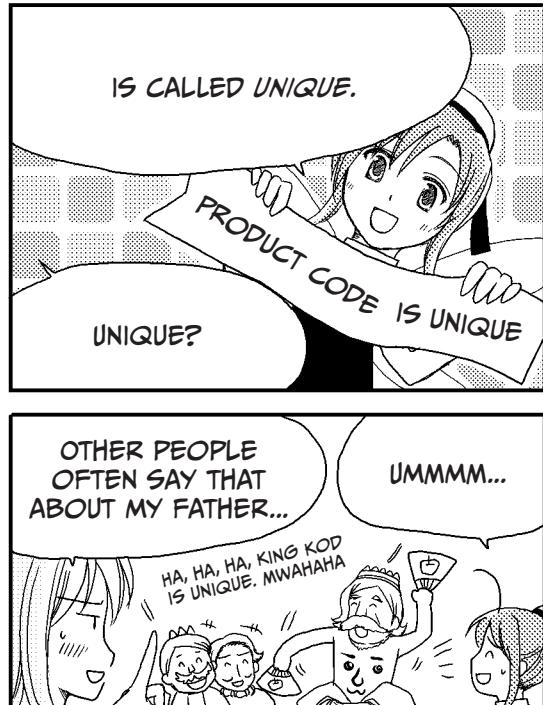
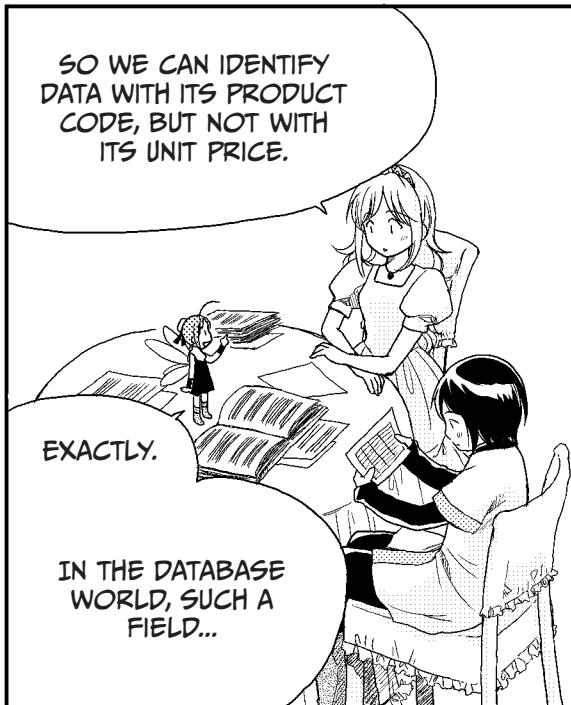
PRODUCT CODE
1 0 1
1 0 2
1 0 3
1 0 4
2 0 1
2 0 2
3 0 1
3 0 2

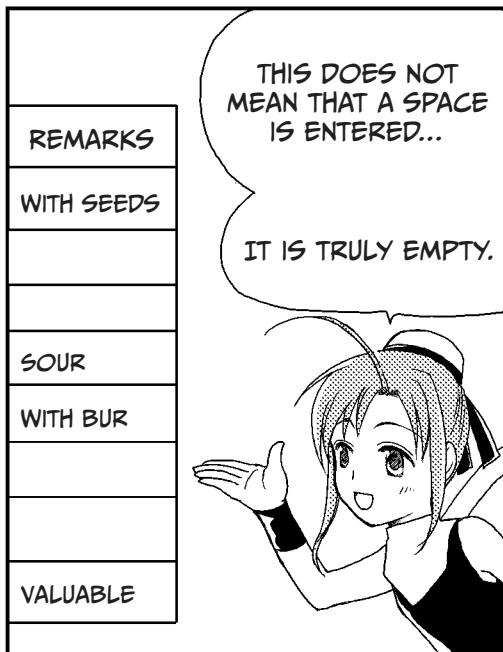
THEN, NEXT, LET'S THINK
ABOUT THE PRODUCT
CODE IN A LITTLE MORE
DETAIL.

HERE!

CAIN?







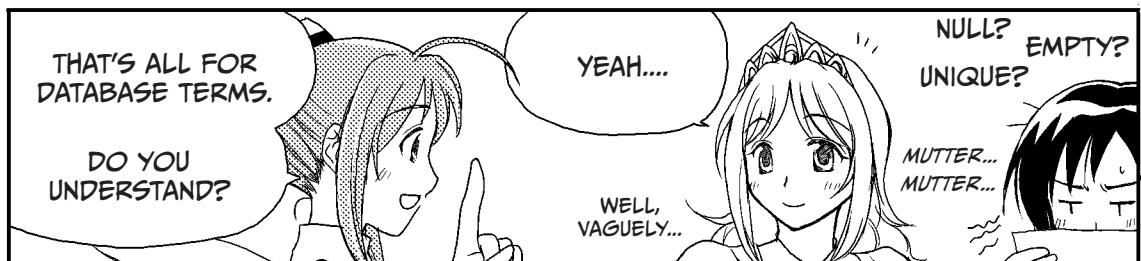
CANNOT BE NULL.

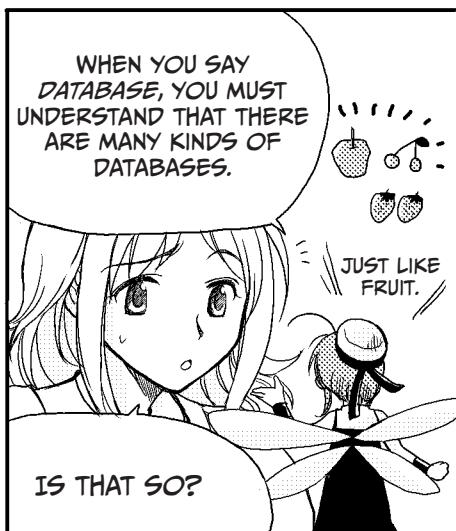
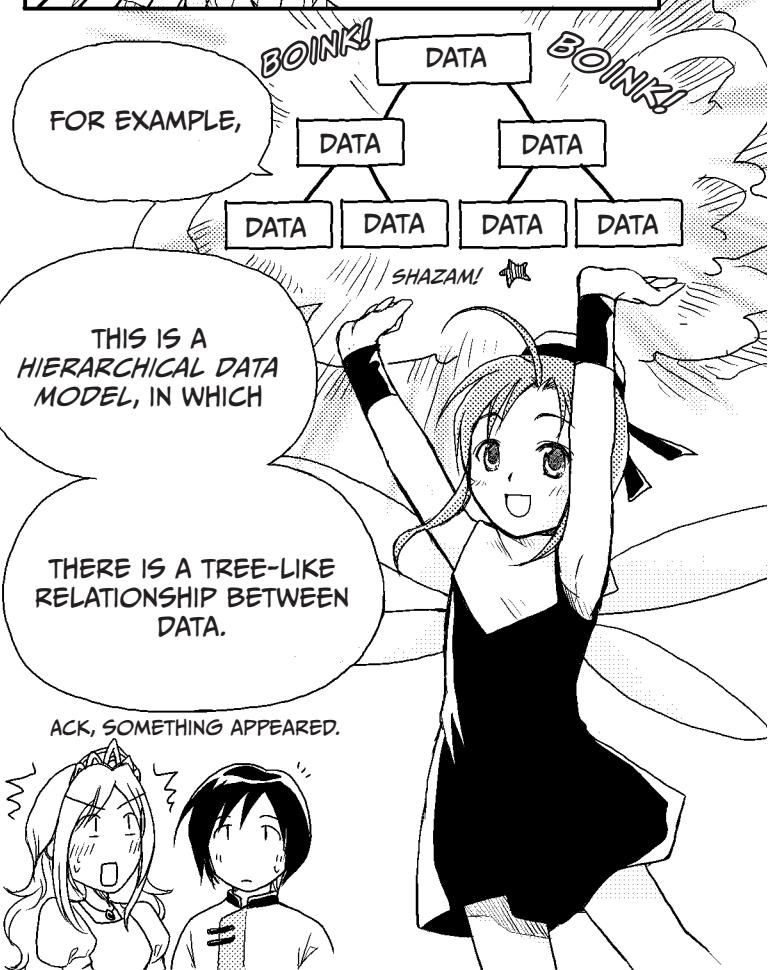
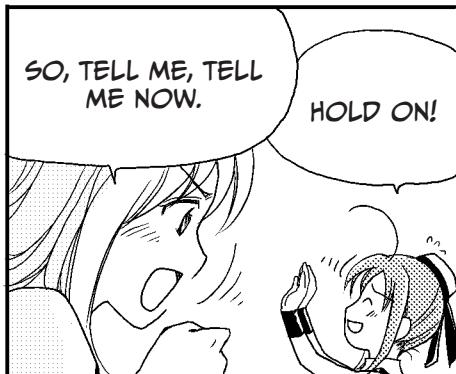
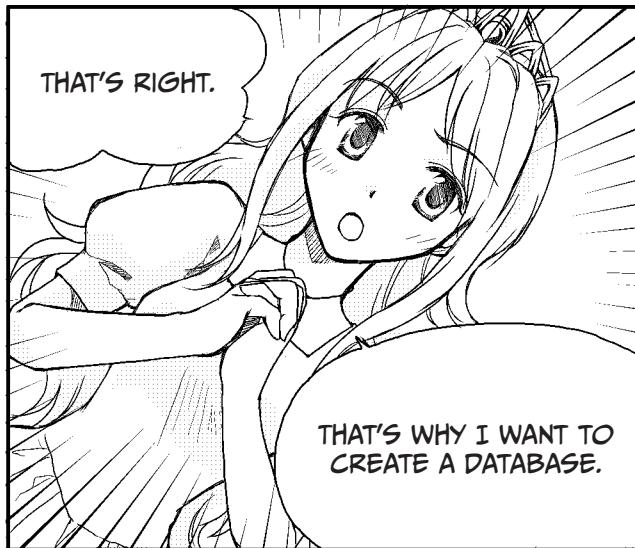
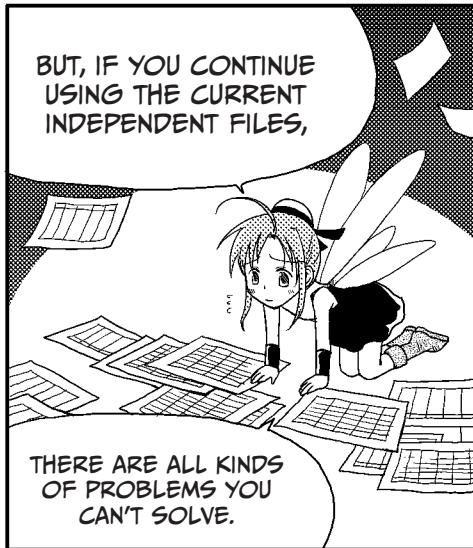
PRODUCT CODE	PRODUCT NAME	UNIT PRICE	REMARKS
101	MELON	800G	WITH SEEDS
102	STRAWBERRY	150G	
103	APPLE	120G	
104	LEMON	200G	SOUR
201	CHESTNUT	100G	WITH BUR
202	PERSIMMON	160G	
301	PEACH	130G	
302	KIWI	200G	

A NULL

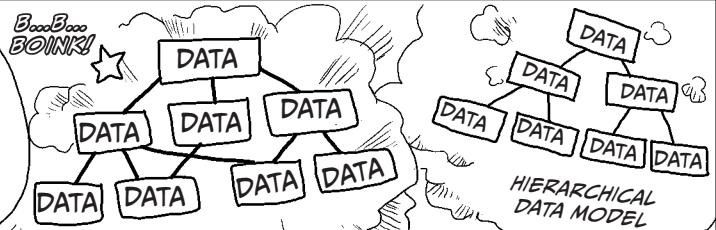
THE ABSENCE OF A VALUE
IS CALLED A NULL IN THE
DATABASE WORLD.

A NULL IS ACCEPTABLE FOR
REMARKS, BUT NOT FOR
PRODUCT CODE, WHICH
IDENTIFIES DATA.





NEXT IS THE NETWORK DATA MODEL, IN WHICH PIECES OF DATA HAVE OVERLAPPING RELATIONSHIPS WITH EACH OTHER.



ARE YOU NOT ASTONISHED AT ALL, CAIN?

I'M READY FOR ANYTHING!

BoBo... BOINK!

THEN, ARE WE GOING TO USE ONE OF THEM?

HUMPH!

SHAZAMI

NOPE!

ZZZT!

ZZB!

BANG!

BANG
BANG!

TENSED UP

AS A MATTER OF FACT, ANOTHER KIND IS MUCH EASIER TO USE THAN THESE TWO.

YOU WERE SCARED, WEREN'T YOU?

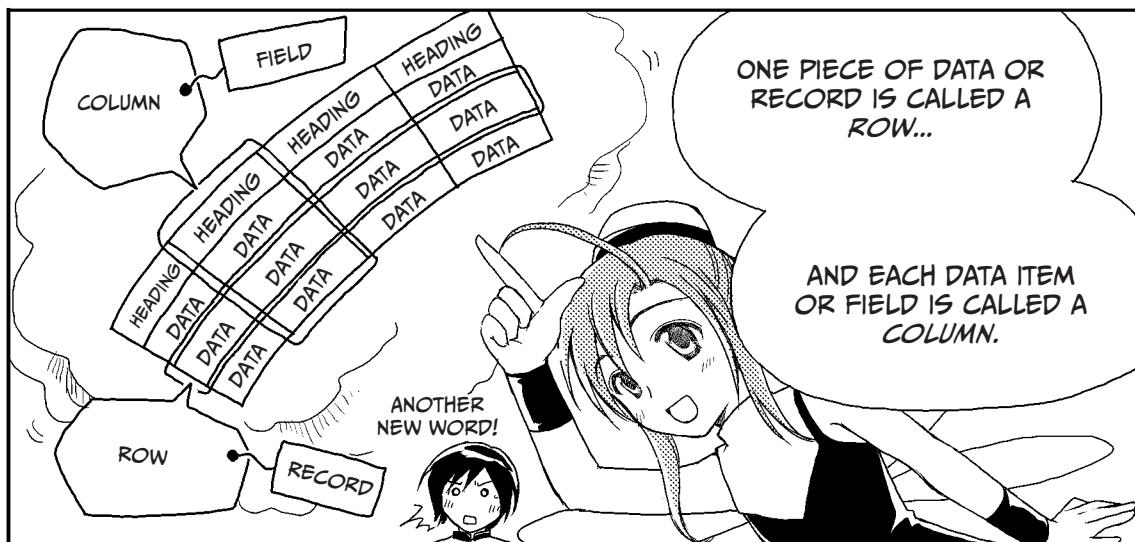
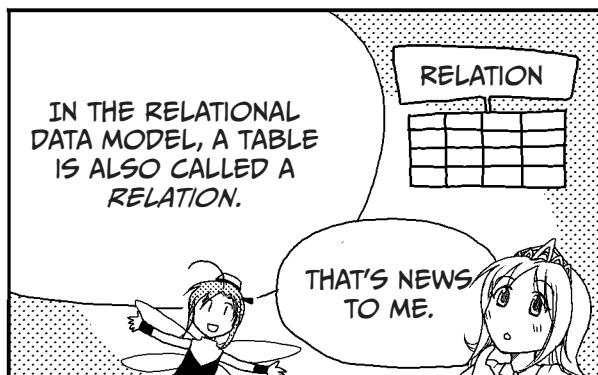
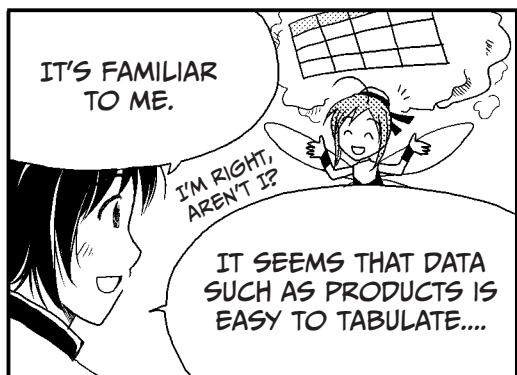
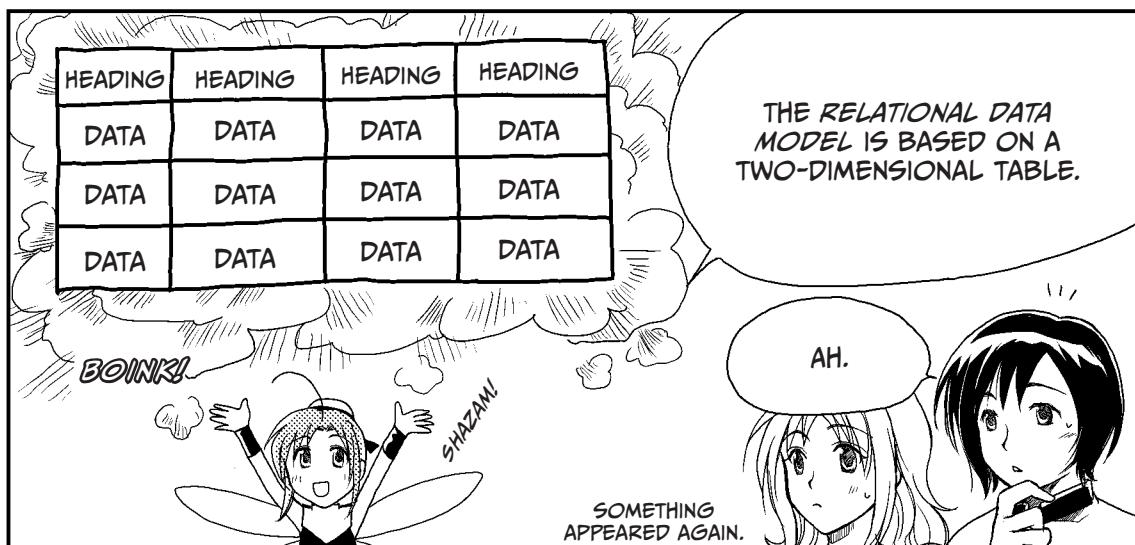
NO, NOT AT ALL.

IT IS CALLED...

A RELATIONAL DATA MODEL.

RELATIONAL?

Relational Databases



IN ADDITION,
A FIELD IS
SOMETIMES GIVEN
AN IMPORTANT
ROLE IN THE
DATABASE.

THIS SPECIAL FIELD
IS CALLED A KEY.

IMPORTANT
ROLE?

YES. FOR
EXAMPLE,

THE PRODUCT
CODE IN THE FILE
YOU SAW A LITTLE
WHILE AGO.

THE FIELD SERVES AN
IMPORTANT ROLE: TO
IDENTIFY DATA.

THIS CODE
IS CALLED A
PRIMARY KEY.

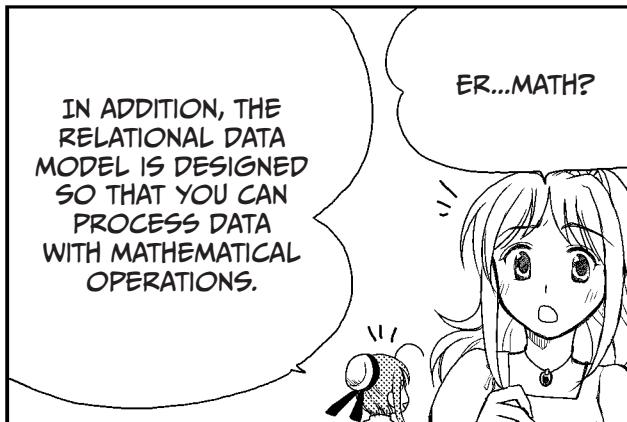
I DIDN'T KNOW
THERE WERE SO
MANY TERMS.

WELL, I'M FAMILIAR
WITH TABLES.

IT IS EASY TO
UNDERSTAND IF YOU
CAN PROCESS DATA
USING A TABLE.

THIS IS ONE MERIT
OF THE RELATIONAL
DATA MODEL.

EVEN PEOPLE WHO
DO NOT KNOW MUCH
ABOUT DATABASES
CAN PROCESS DATA.

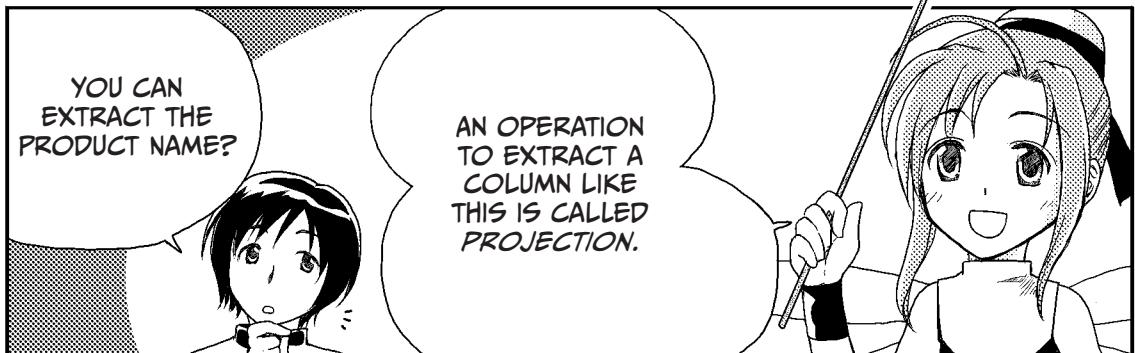


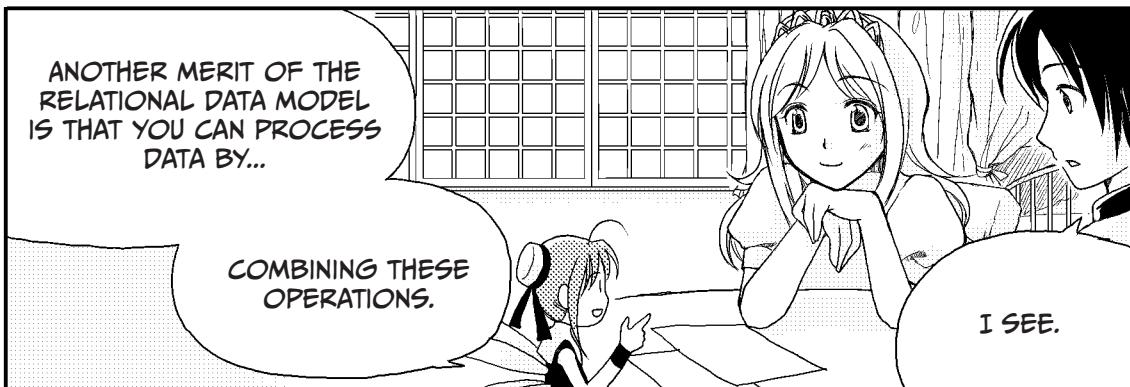
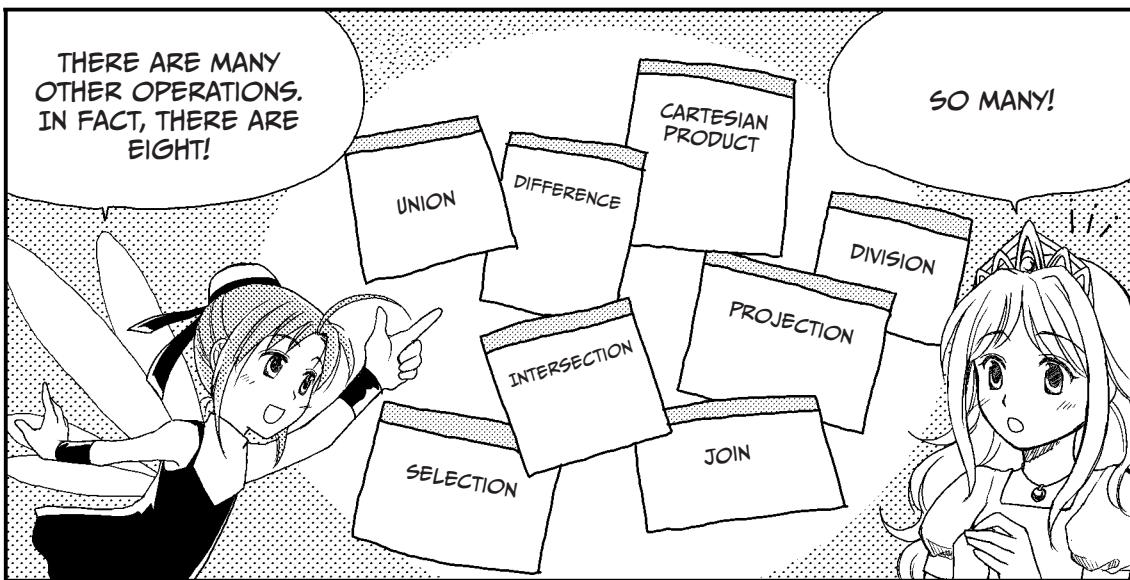
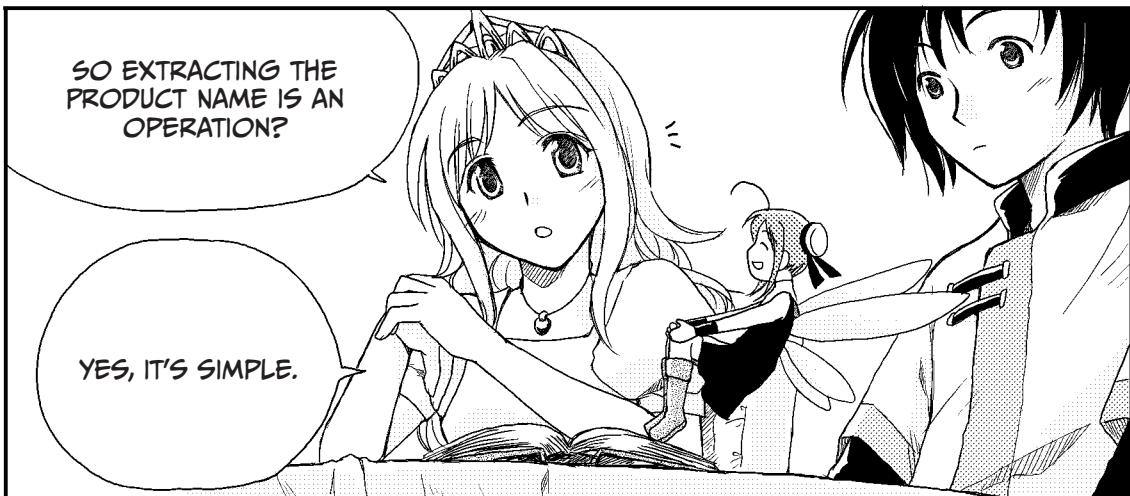
FOR INSTANCE, LET'S LOOK BACK AT THE PRODUCT TABLE.

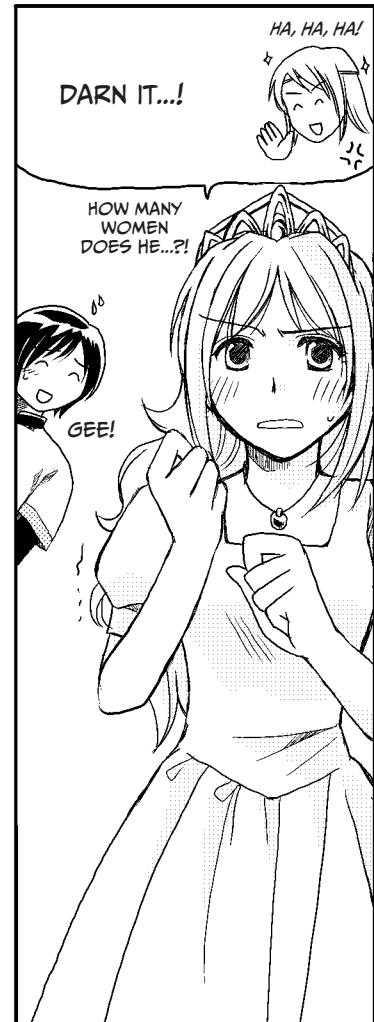
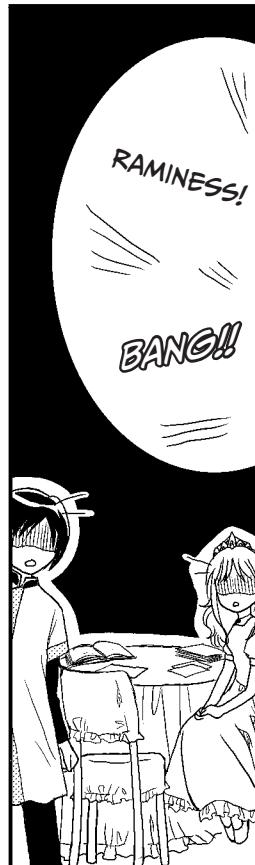
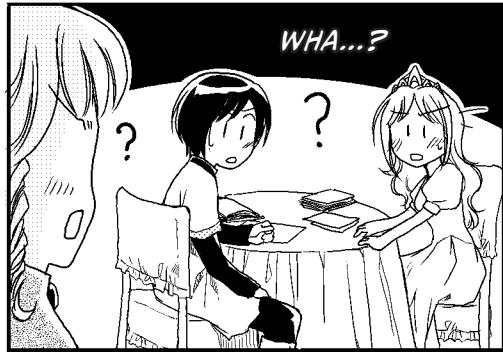
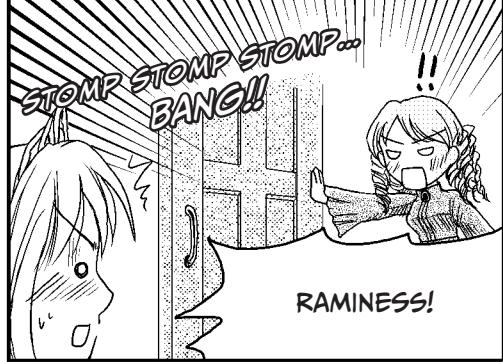
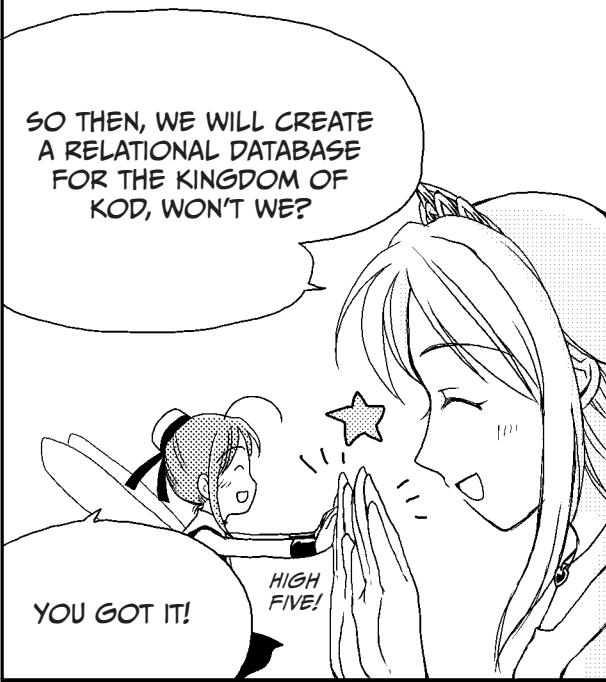
PRODUCT CODE	PRODUCT NAME	UNIT PRICE	REMARKS
101	MELON	800G	WITH SEEDS
102	STRAWBERRY	150G	
103	APPLE	120G	
104	LEMON	200G	SOUR
201	CHESTNUT	100G	WITH BUR
202	PERSIMMON	160G	
301	PEACH	130G	
302	KIWI	200G	VALUABLE

PRODUCT NAME
MELON
STRAWBERRY
APPLE
LEMON
CHESTNUT
PERSIMMON
PEACH
KIWI

MAGIC!





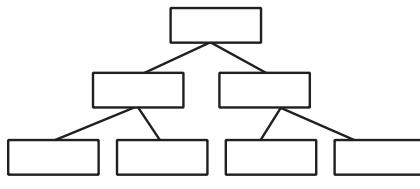


TYPES OF DATA MODELS

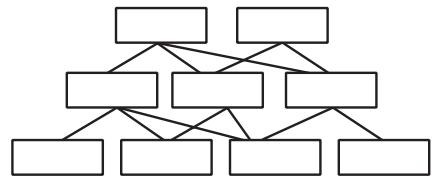


When you use the term *database*, what kind of database do you mean? There are many types available for data management. The data association and operation methods that a database uses is called its *data model*. There are three commonly used data models.

As I described to Ruruna and Cain, the first type is the hierarchical data model. In the *hierarchical data model*, child data has only one piece of parent data. The second type is the network data model. Unlike the hierarchical data model, in the *network data model*, child data can have multiple pieces of parent data.



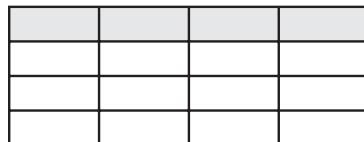
Hierarchical data model



Network data model

To use either of these models, you must manage data by keeping the physical location and the order of data in mind. Therefore, it is difficult to perform a flexible and high-speed search of your data if you use a hierarchical or network data model.

The third type of model is the relational data model. A *relational* database processes data using the easy-to-understand concept of a table. Let's discuss this model in more detail.



Relational data model

DATA EXTRACTION OPERATIONS

How is data extracted in a relational database? You can process and extract data in a relational database by performing stringently defined mathematical operations. There are eight main operations that you can use, and they fall into two categories—set operations and relational operations.

SET OPERATIONS

The union, difference, intersection, and Cartesian product operations are called *set operations*. These operations work upon one or more sets of rows to produce a new set of rows. In short, they determine which rows from the input appear in the output. Let's look at some examples using Product Table 1 and Product Table 2.

PRODUCT TABLE 1

Product name	Unit price
Melon	800G
Strawberry	150G
Apple	120G
Lemon	200G

PRODUCT TABLE 2

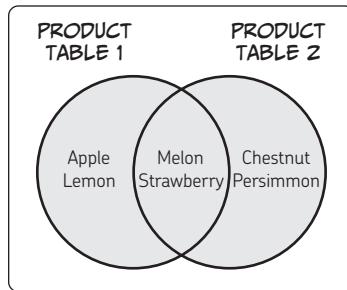
Product name	Unit price
Melon	800G
Strawberry	150G
Chestnut	100G
Persimmon	350G

UNION

Carrying out the *union* operation allows you to extract all products included in Product Table 1 and Product Table 2. The result is below.

Product name	Unit price
Melon	800G
Strawberry	150G
Apple	120G
Lemon	200G
Chestnut	100G
Persimmon	350G

Performing a union operation extracts all rows in the two tables and combines them. The following figure shows what the data from the two tables looks like once a union operation has been performed. All rows in Product Table 1 and Product Table 2 have been extracted.

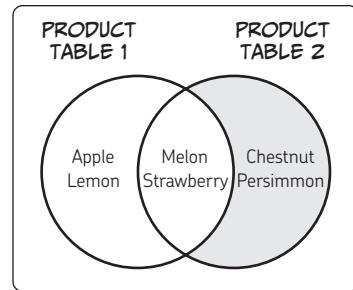
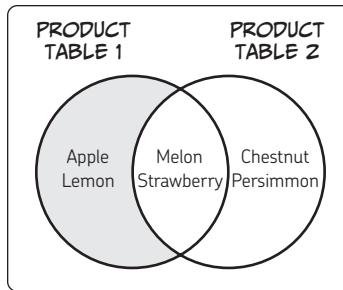


DIFFERENCE

Difference is an operation that extracts rows from just *one* of the tables. For example, a difference operation can extract all of the products from the first table that are not included in the second. The results depend on which table contains rows to extract, and which table has rows to exclude.

Product name	Unit price
Apple	120G
Lemon	200G

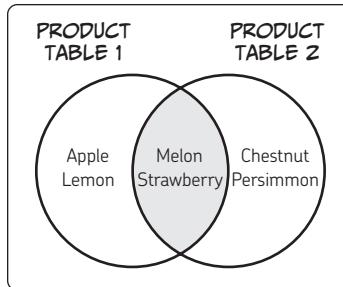
Product name	Unit price
Chestnut	100G
Persimmon	350G



INTERSECTION

You can also extract products that are included in *both* Product Table 1 and Product Table 2. This operation is called an *intersection* operation. Here is the result of the intersection of Product Tables 1 and 2.

Product name	Unit price
Melon	800G
Strawberry	150G



CARTESIAN PRODUCT

The *Cartesian product* operation is a method that combines all rows in the two tables. Let's look at the Product and Export Destination Tables below.

The Cartesian product operation combines all rows in the two tables. In this example, it resulted in $3 \times 3 = 9$ rows. Notice that the column names (or fields) in these two tables are not the same—unlike our previous examples.

PRODUCT TABLE

Product code	Product name	Unit price
101	Melon	800G
102	Strawberry	150G
103	Apple	120G

EXPORT DESTINATION TABLE

Export dest. code	Export dest. name
12	The Kingdom of Minanmi
23	Alpha Empire
25	The Kingdom of Ritol

3
rows

CARTESIAN PRODUCT



Product code	Product name	Unit price	Export dest. code	Export dest. name
101	Melon	800G	12	The Kingdom of Minanmi
101	Melon	800G	23	Alpha Empire
101	Melon	800G	25	The Kingdom of Ritol
102	Strawberry	150G	12	The Kingdom of Minanmi
102	Strawberry	150G	23	Alpha Empire
102	Strawberry	150G	25	The Kingdom of Ritol
103	Apple	120G	12	The Kingdom of Minanmi
103	Apple	120G	23	Alpha Empire
103	Apple	120G	25	The Kingdom of Ritol

3 x 3 =
9 rows



RELATIONAL OPERATIONS

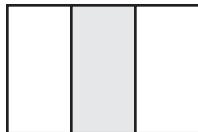
A relational database is designed so that data can be extracted by set operations and relational operations. Let's look at the other four operations specific to a relational database, called *relational operations*—projection, selection, join, and division.

PROJECTION

Projection is an operation that extracts columns from a table. In the example shown here, this operation is used to extract only product names included in the Product Table.

Product name
Melon
Strawberry
Apple
Lemon

Think of projection as extracting “vertically,” as shown below.

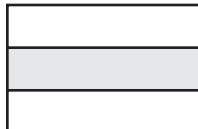


SELECTION

The *selection* operation extracts two rows from a table.

Product name	Unit price
Melon	800G
Strawberry	150G

Selection is like projection, but it extracts rows instead of columns. Selection extracts data “horizontally.”



JOIN

The *join* operation is a very powerful one. This operation literally refers to the work of joining tables. Let's look at the tables below as an example.

PRODUCT TABLE

Product code	Product name	Unit price
101	Melon	800G
102	Strawberry	150G
103	Apple	120G
104	Lemon	200G

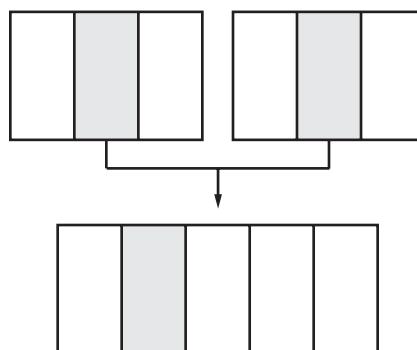
SALES TABLE

Date	Product code	Quantity
11/1	102	1,100
11/1	101	300
11/5	103	1,700
11/8	101	500

The Product Code columns in these two tables represent the same information. On November 1st, 1,100 strawberries (product code 102) were sold. The Sales Table does not include the product name, but it does include the product code. In other words, the Sales Table allows you to understand which product was sold by making reference to the product code, which is the *primary key* in the Product Table. The product code in the Sales Table is a *foreign key*. Joining the two tables so that the foreign key refers to the primary key results in the following table.

Date	Product code	Product name	Unit price	Quantity
11/1	102	Strawberry	150G	1,100
11/1	101	Melon	800G	300
11/5	103	Apple	120G	1,700
11/8	101	Melon	800G	500

This creates a new dynamic table of sales data, including date, product code, product name, unit price, and quantity. The figure below shows a join—the shaded area represents a column that appears in both original tables.



DIVISION

Finally, let's look at division. *Division* is an operation that extracts the rows whose column values match those in the second table, but only returns columns that don't exist in the second table. Let's look at an example.

SALES TABLE

Export dest. code	Export dest. name	Date
12	The Kingdom of Minanmi	3/5
12	The Kingdom of Minanmi	3/10
23	Alpha Empire	3/5
25	The Kingdom of Ritol	3/21
30	The Kingdom of Sazanna	3/25

EXPORT DESTINATION TABLE

Export dest. code	Export dest. name
12	The Kingdom of Minanmi
23	Alpha Empire

Dividing the Sales Table by the Export Destination Table results in the following table. This allows you to find the dates when fruit was exported to both the Alpha Empire and the Kingdom of Minanmi.

Date
3/5

QUESTIONS



Now, let's answer some questions to see how well you understand relational databases. The answers are on page 48.

Q1

What do you call the key referring to a column in a different table in a relational database?

Q2

The following table displays information about books. Which item can you use as a primary key? The ISBN is the International Standard Book Number, a unique identifying number given to every published book. Some books may have the same title.

ISBN	Book name	Author name	Publication date	Price
------	-----------	-------------	------------------	-------

Q3

What do you call the operation used here to extract data?

Export dest. code	Export dest. name
12	The Kingdom of Minanmi
23	Alpha Empire
25	The Kingdom of Ritol
30	The Kingdom of Sazanna



Export dest. code	Export dest. name
25	The Kingdom of Ritol

Q4

What do you call the operation used here to extract data?

Export dest. code	Export dest. name
12	The Kingdom of Minanmi
23	Alpha Empire
25	The Kingdom of Ritol
30	The Kingdom of Sazanna

Export dest. code	Export dest. name
15	The Kingdom of Paronu
22	The Kingdom of Tokanta
31	The Kingdom of Taharu
33	The Kingdom of Mariyon



Export dest. code	Export dest. name
12	The Kingdom of Minanmi
15	The Kingdom of Paronu
22	The Kingdom of Tokanta
23	Alpha Empire
25	The Kingdom of Ritol
30	The Kingdom of Sazanna
31	The Kingdom of Taharu
33	The Kingdom of Mariyon

Q5

What do you call the operation used here to extract data?

Export dest. code	Export dest. name
12	The Kingdom of Minanmi
23	Alpha Empire
25	The Kingdom of Ritol
30	The Kingdom of Sazanna

Export dest. code	Date
12	3/1
23	3/1
12	3/3
30	3/5
12	3/6
25	3/10



Export dest. code	Date	Export dest. name
12	3/1	The Kingdom of Minanmi
23	3/1	Alpha Empire
12	3/3	The Kingdom of Minanmi
30	3/5	The Kingdom of Sazanna
12	3/6	The Kingdom of Minanmi
25	3/10	The Kingdom of Ritol

THE RELATIONAL DATABASE PREVAILS!

In a relational database, you can use eight different operations to extract data. The extracted results are tabulated. If you combine the operations explained in this section, you can extract data for any purpose. For example, you can use the name and price of a product to create gross sales aggregate data for it. Relational databases are popular because they're easy to understand and provide flexible data processing.

SUMMARY



- One row of data is called a *record*, and each column is called a *field*.
- A column that can be used to identify data is called a *primary key*.
- In a relational database, you can process data using the concept of a table.
- In a relational database, you can process data based on mathematical operations.

ANSWERS

Q1 Foreign key

Q2 ISBN

Q3 Selection

Q4 Union

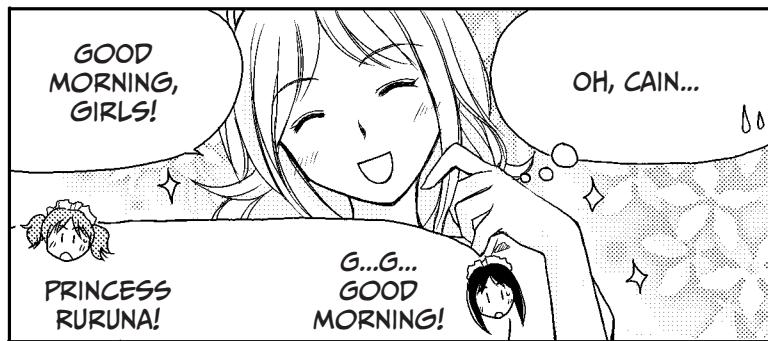
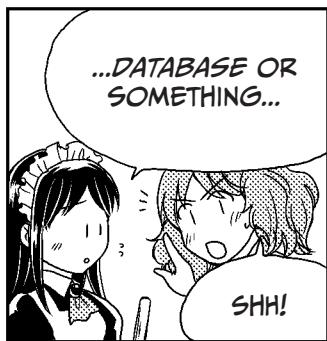
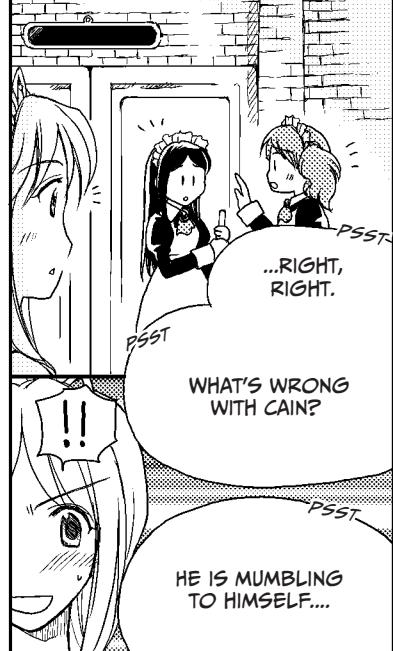
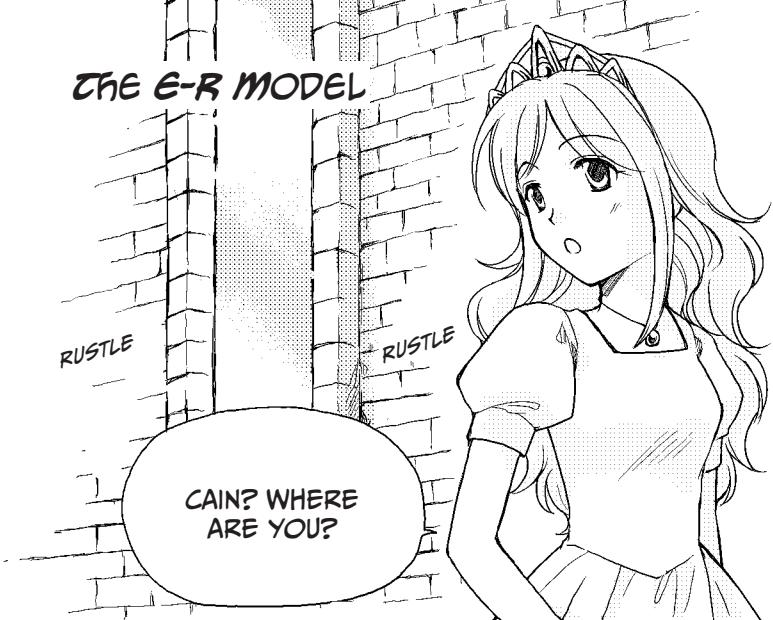
Q5 Join

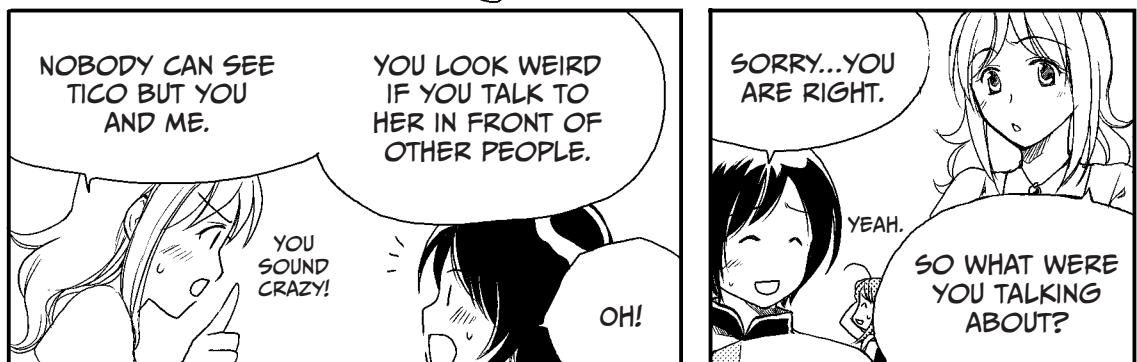
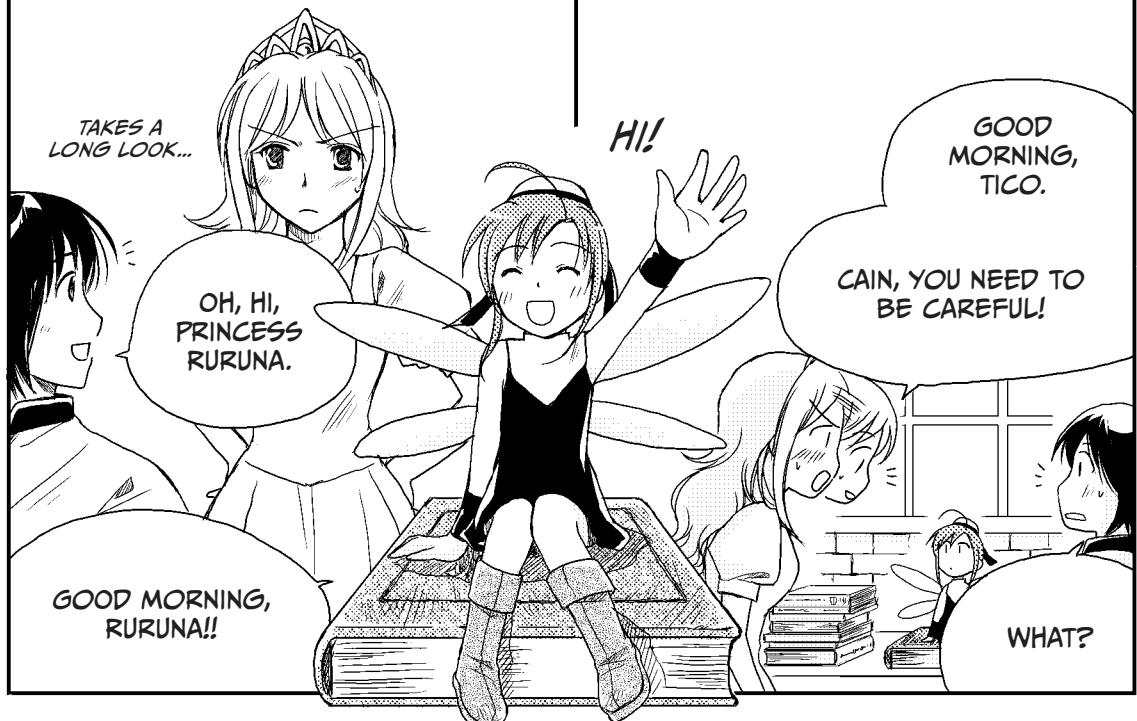
3

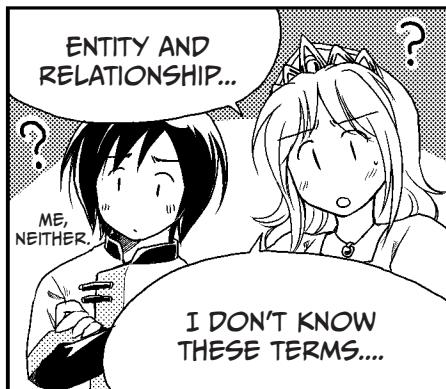
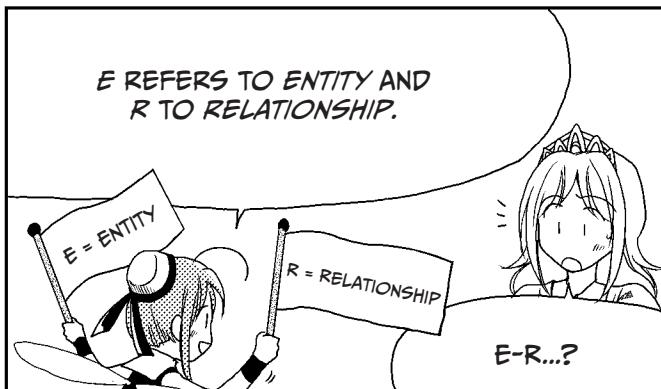
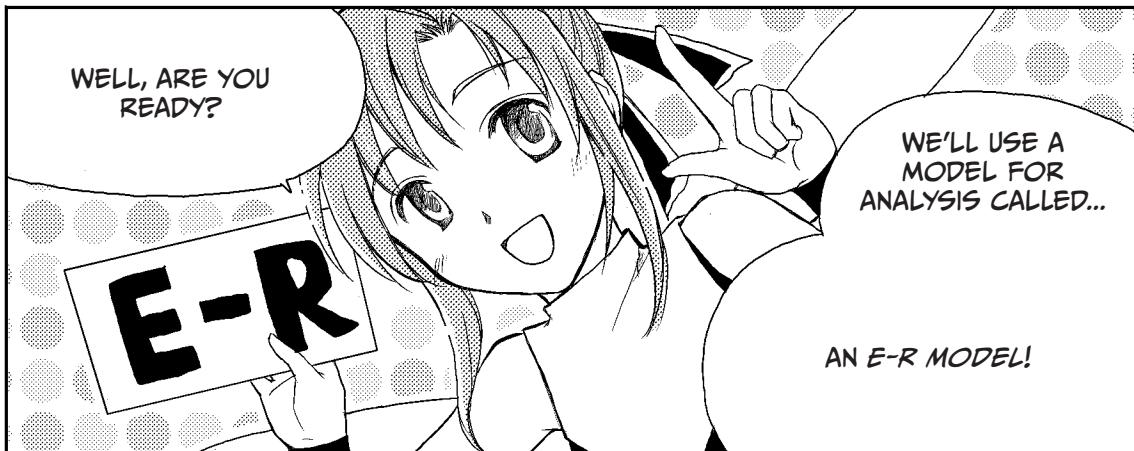
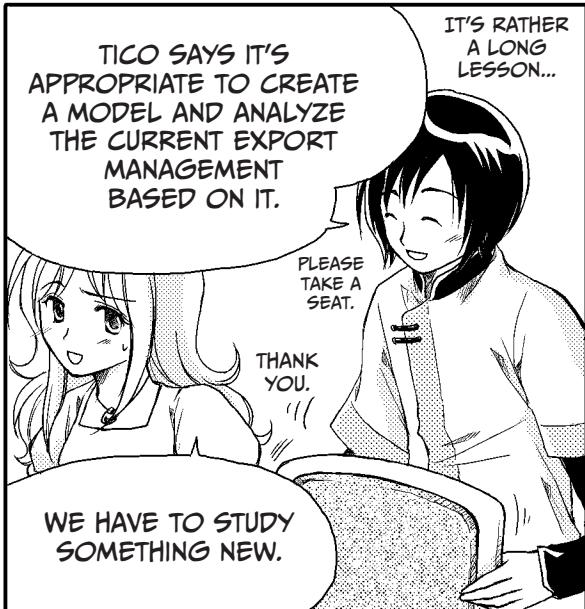
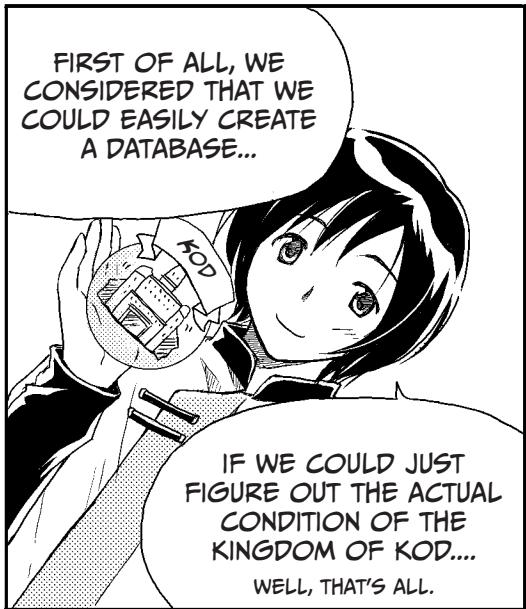
LET'S DESIGN a DATABASE!

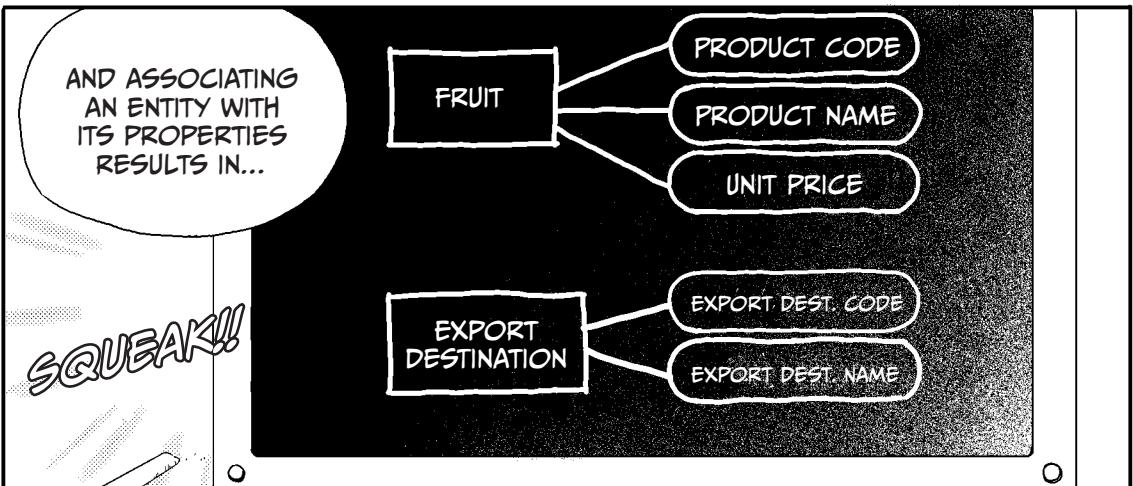
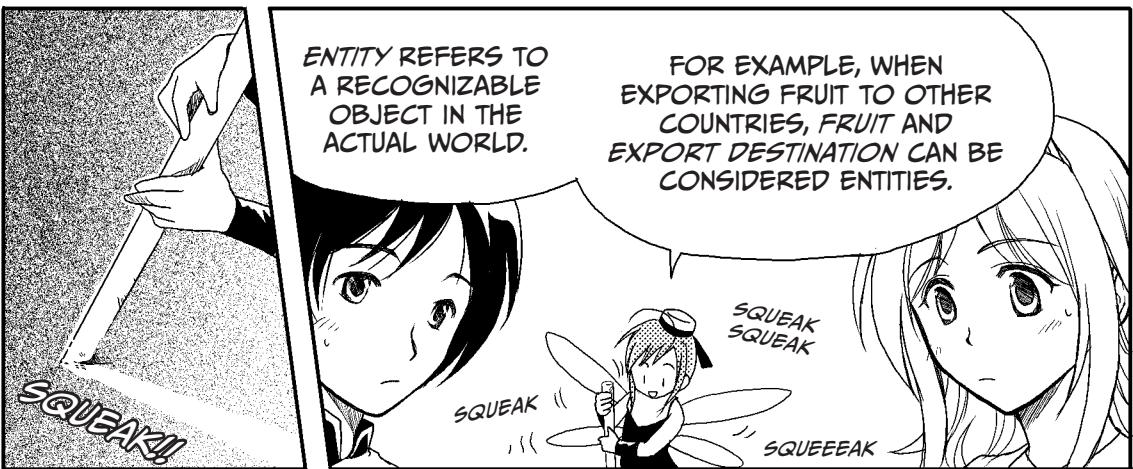
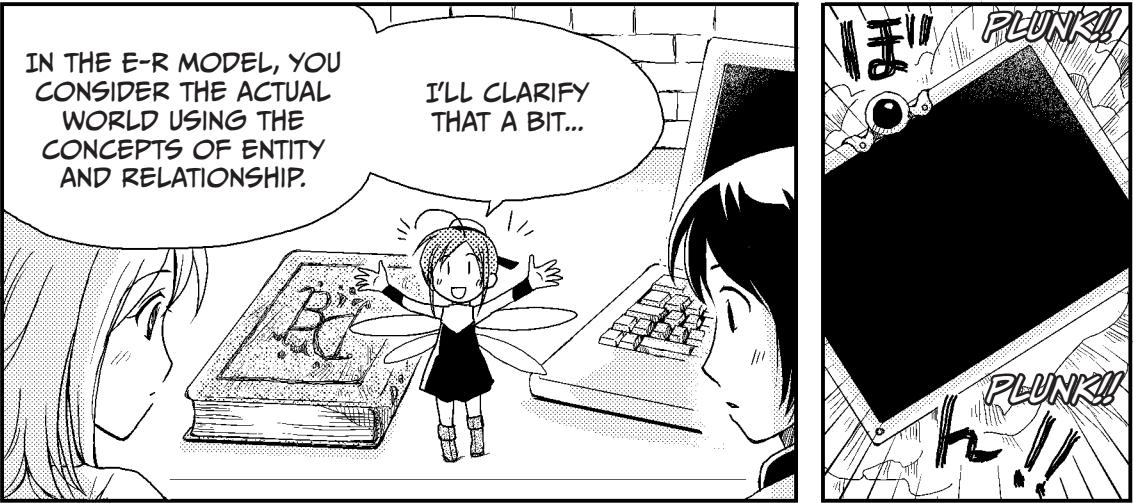


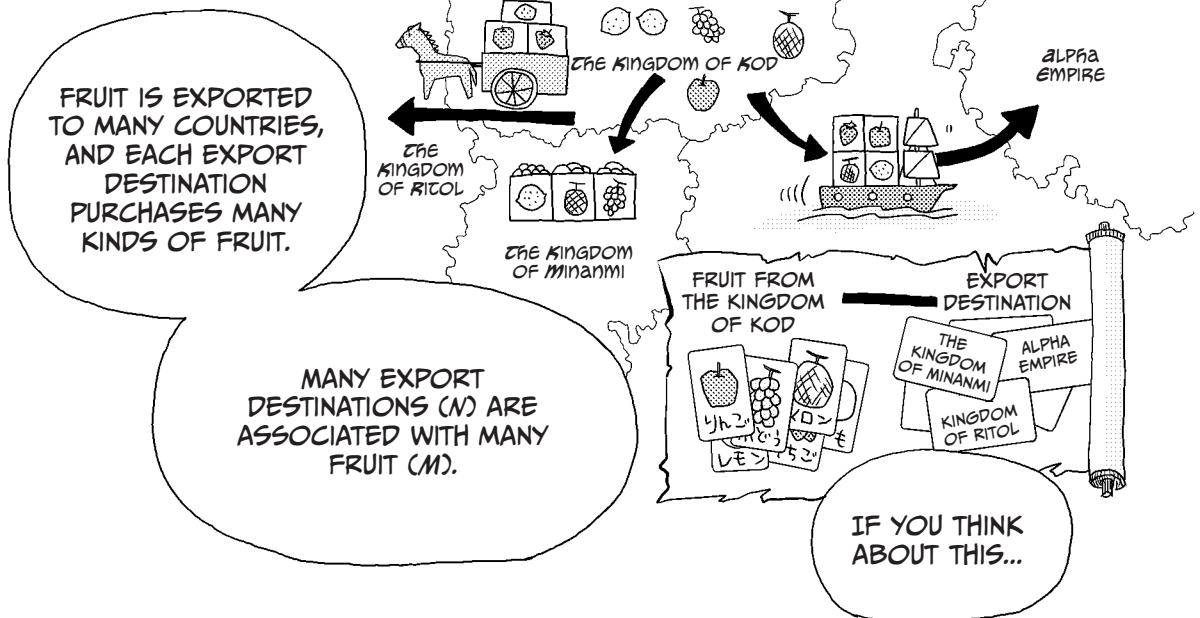
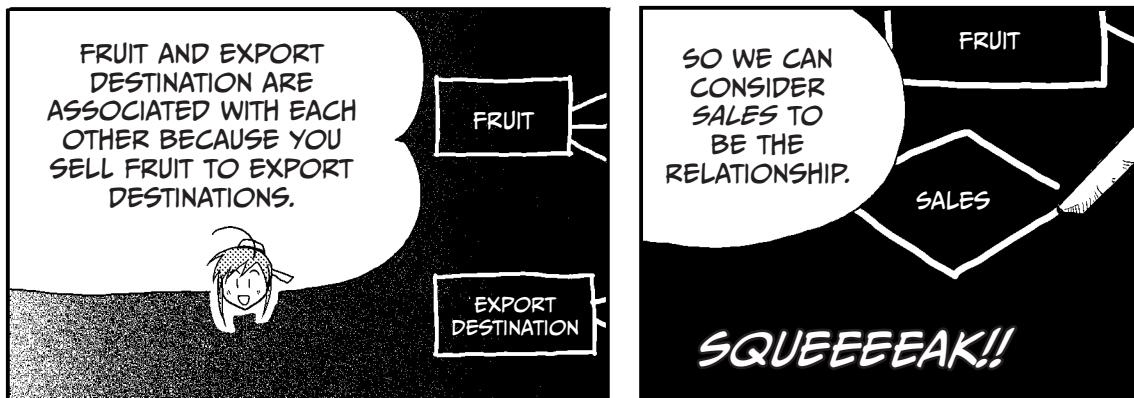
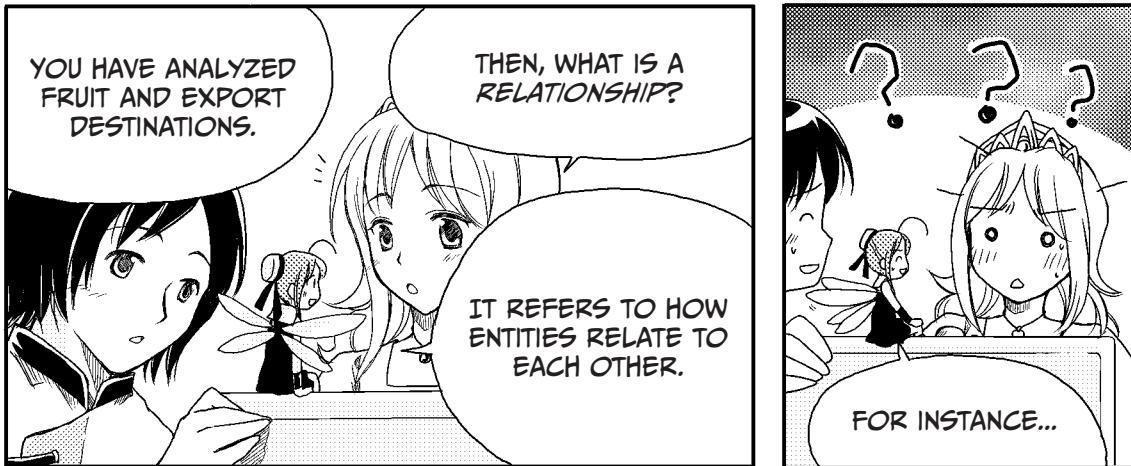
THE E-R MODEL

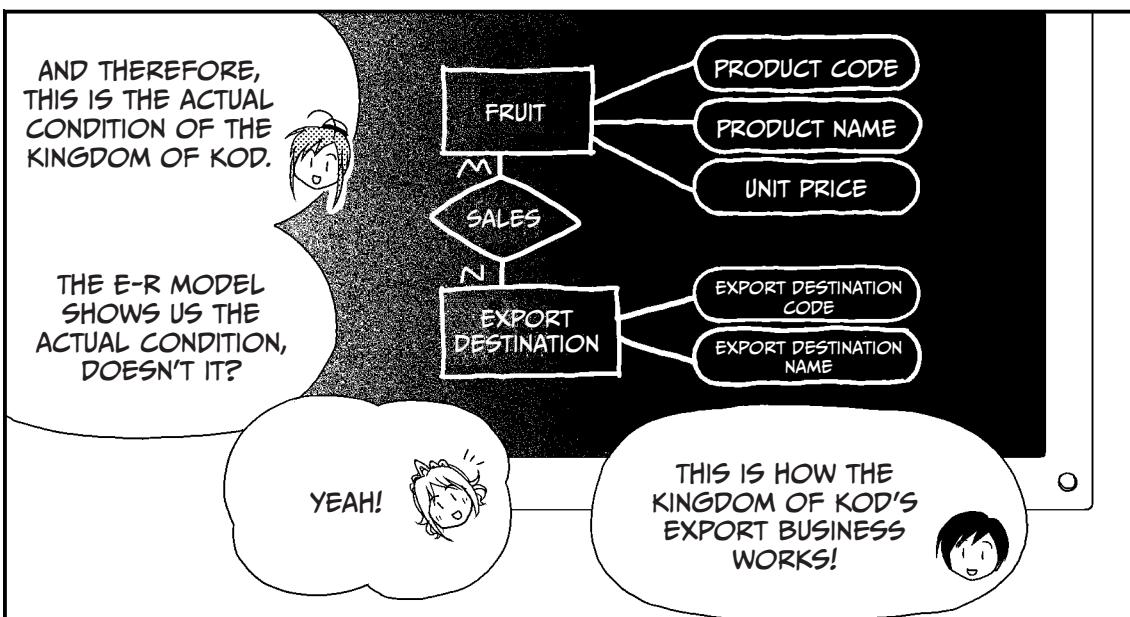
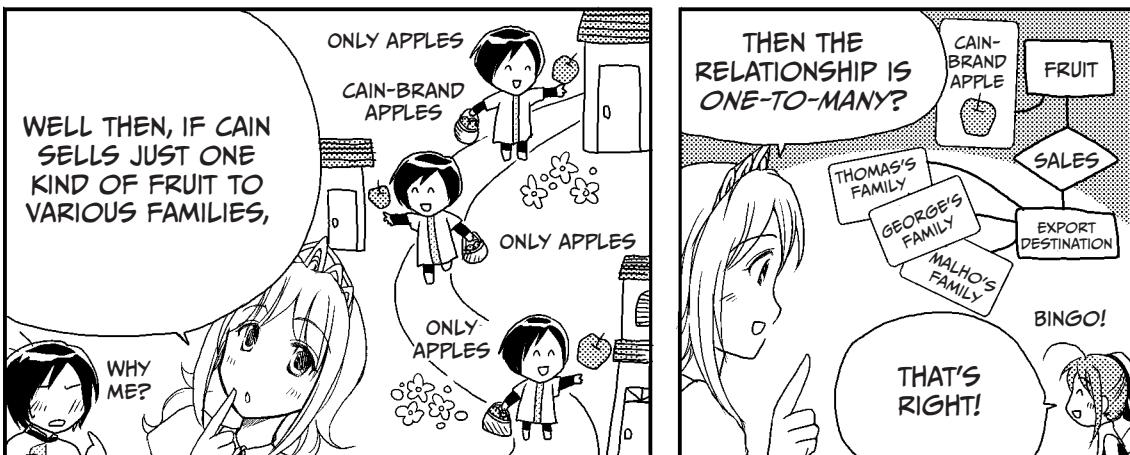
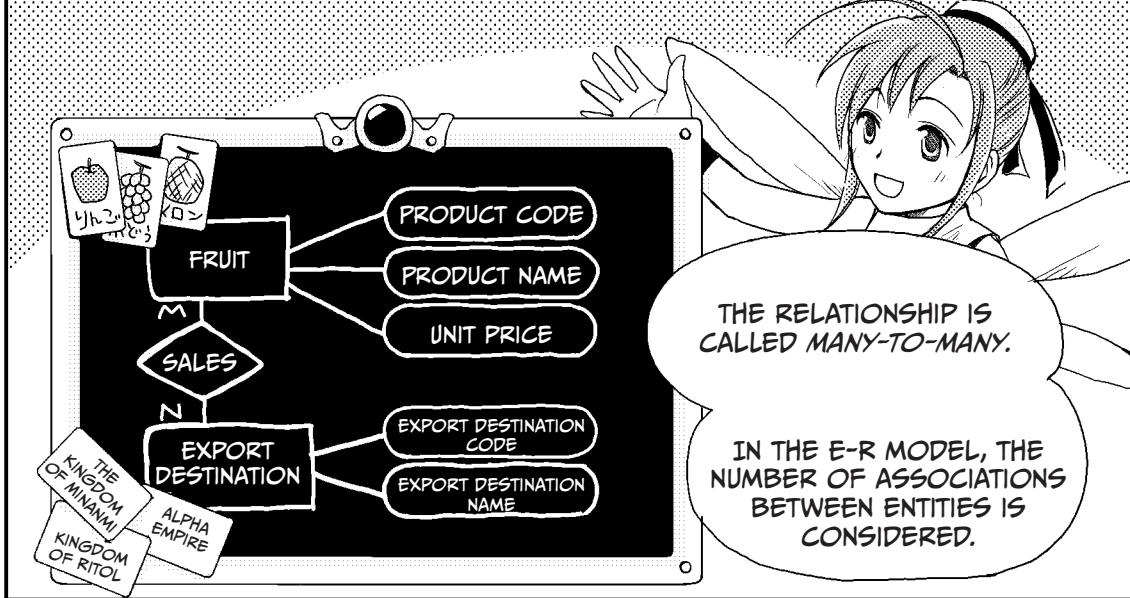




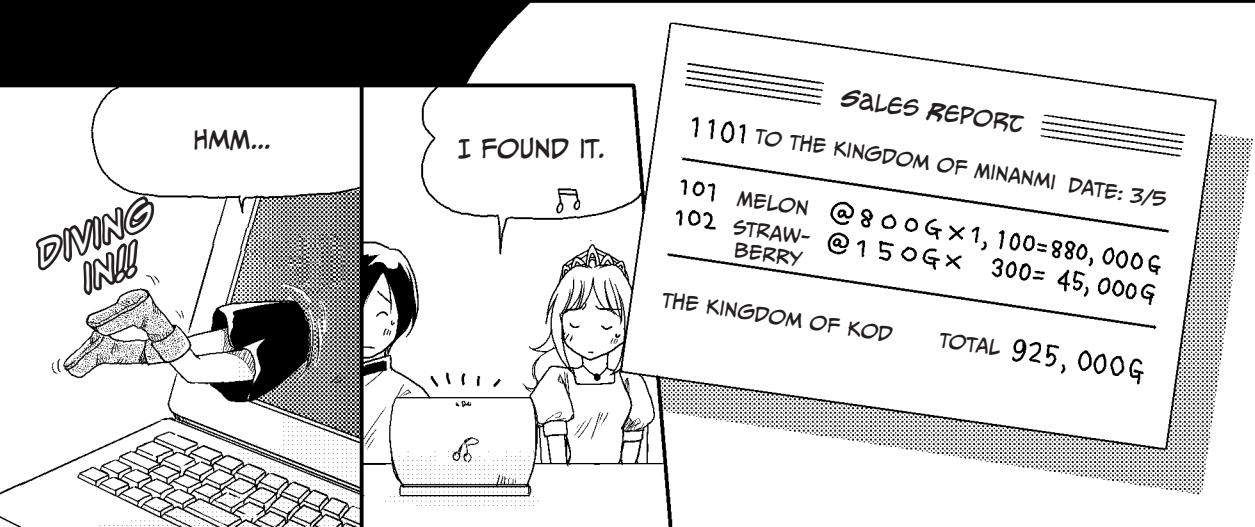
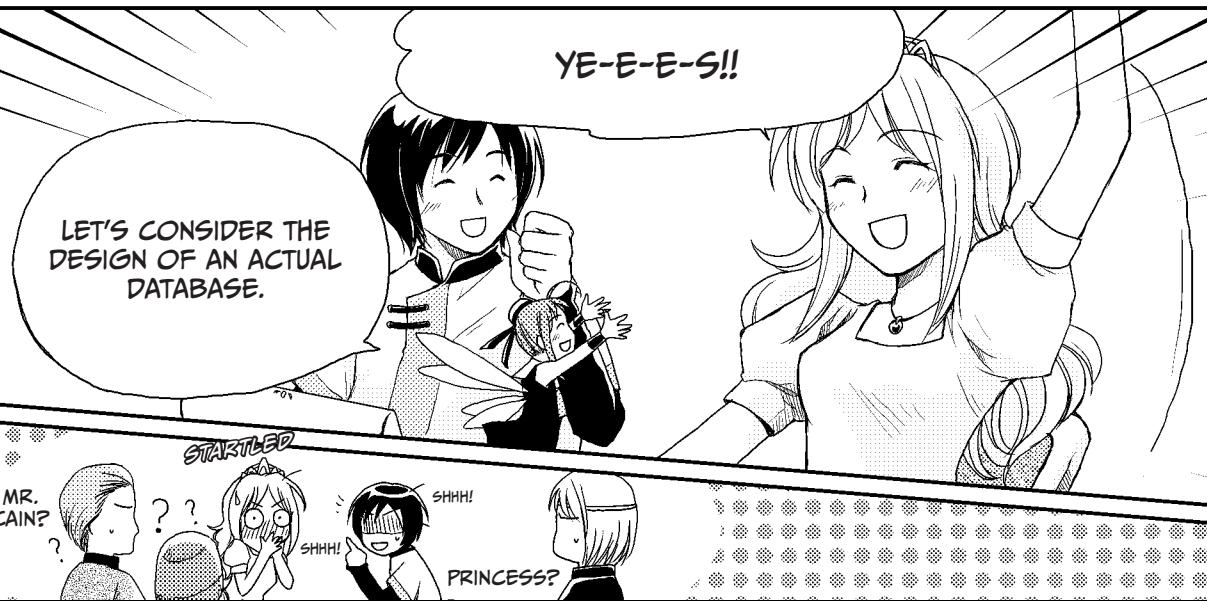
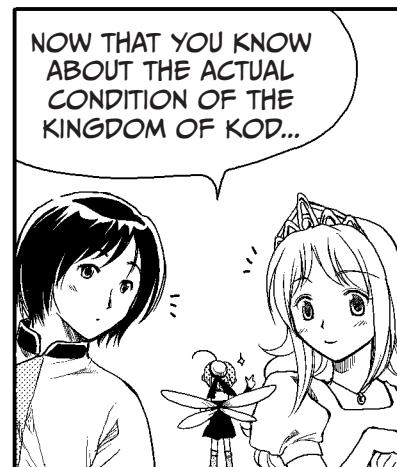
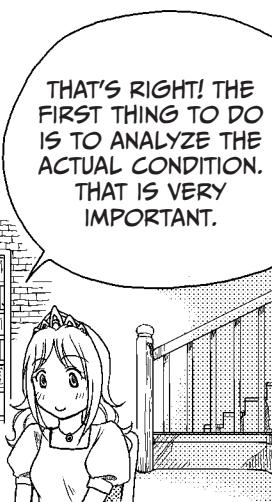
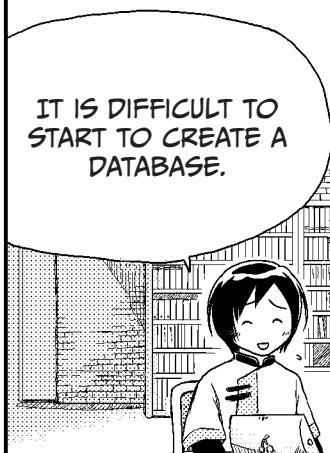


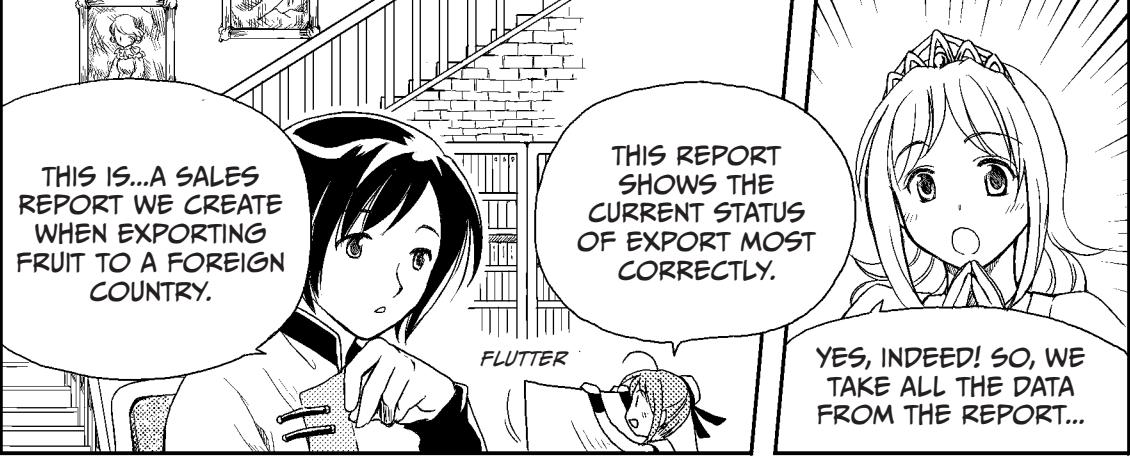






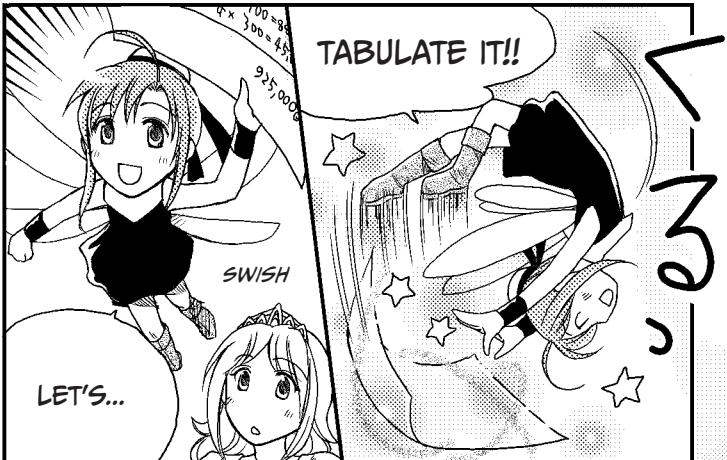
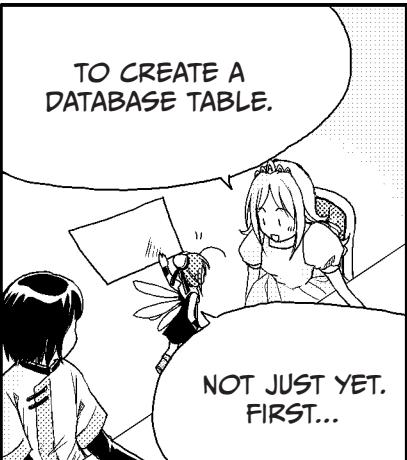
NORMALIZING A TABLE





FLUTTER

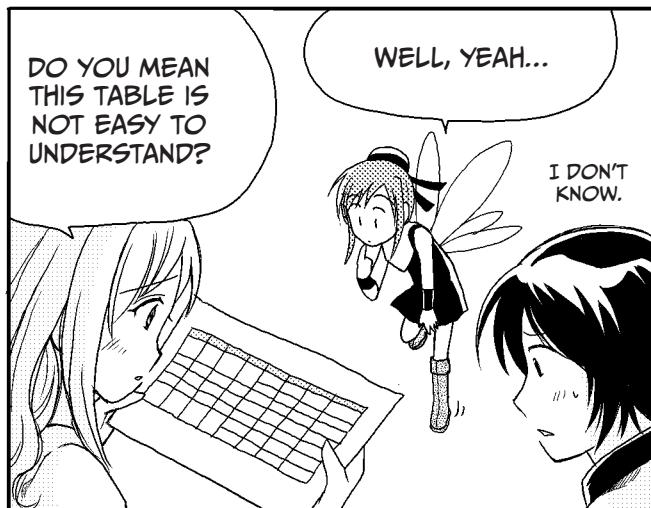
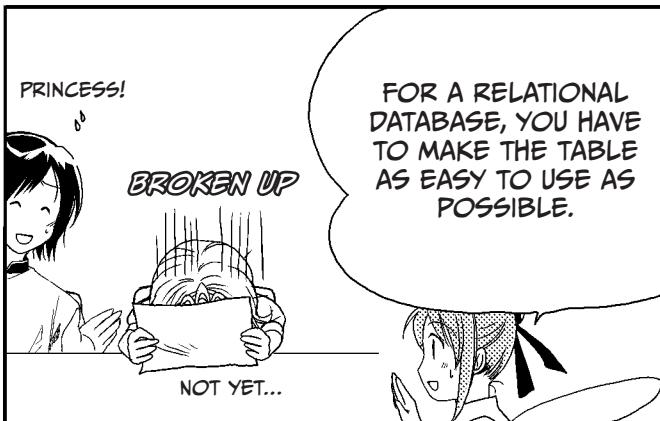
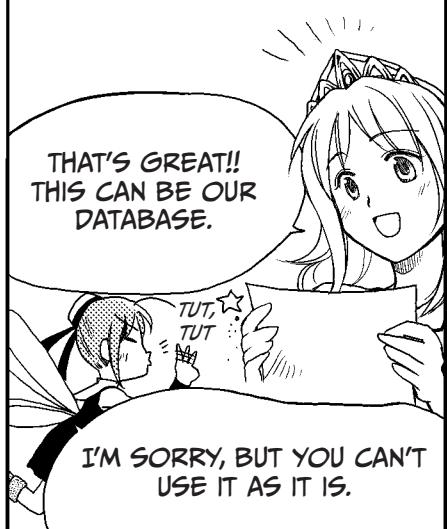
YES, INDEED! SO, WE TAKE ALL THE DATA FROM THE REPORT...



HERE YOU ARE.

REPORT CODE	DATE	EXPORT DEST. CODE	EXPORT DEST. NAME	PRODUCT CODE	PRODUCT NAME	UNIT PRICE	QUANTITY
1101	3/5	12	THE KINGDOM OF MINANMI	101	MELON	800G	1,100
				102	STRAWBERRY	150G	300
1102	3/7	23	ALPHA EMPIRE	103	APPLE	120G	1,700
1103	3/8	25	THE KINGDOM OF RITOL	104	LEMON	200G	500
1104	3/10	12	THE KINGDOM OF MINANMI	101	MELON	800G	2,500
1105	3/12	25	THE KINGDOM OF RITOL	103	APPLE	120G	2,000
				104	LEMON	200G	700

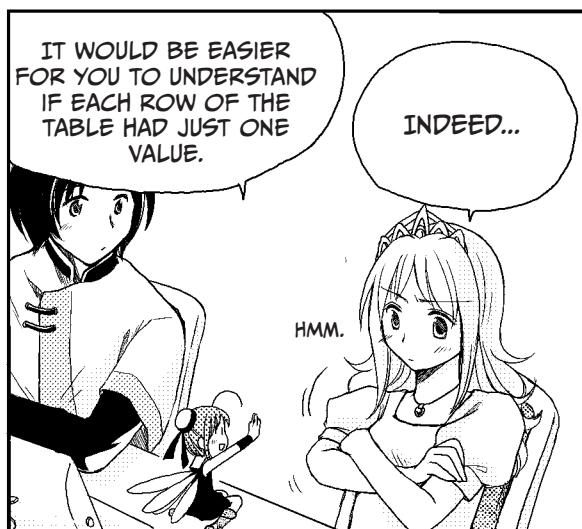
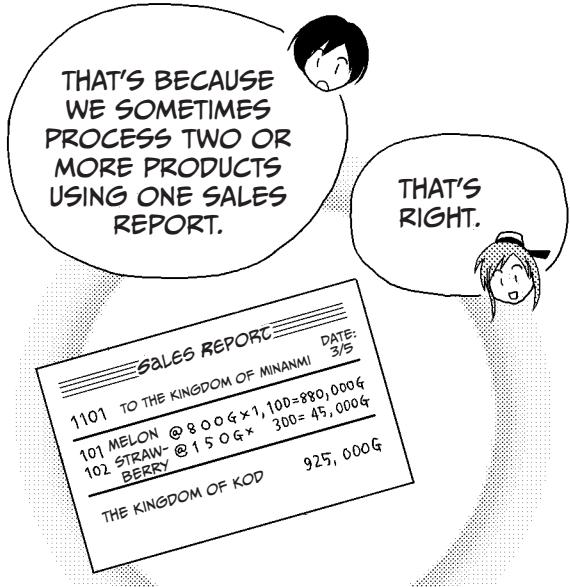
TABLE CREATED FROM SALES REPORT

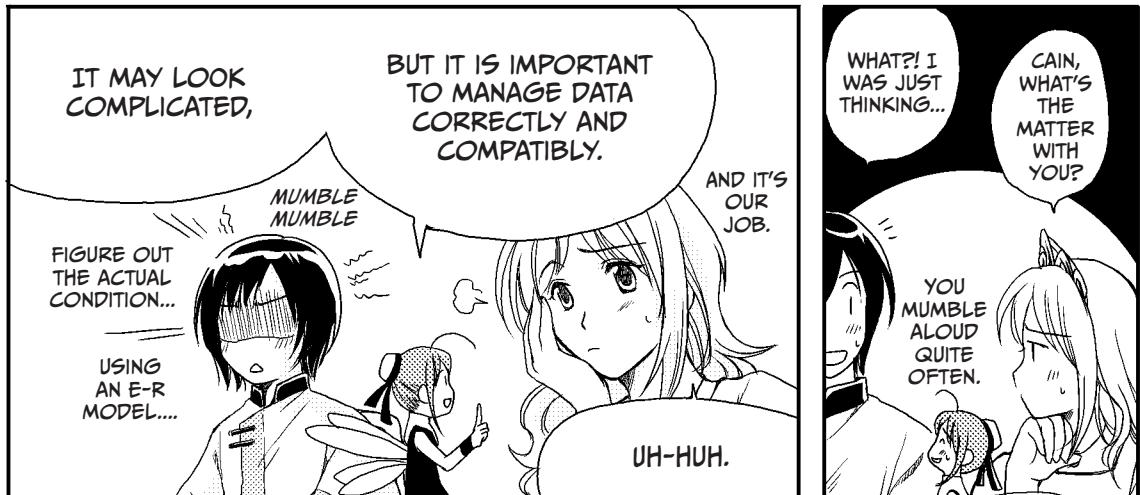
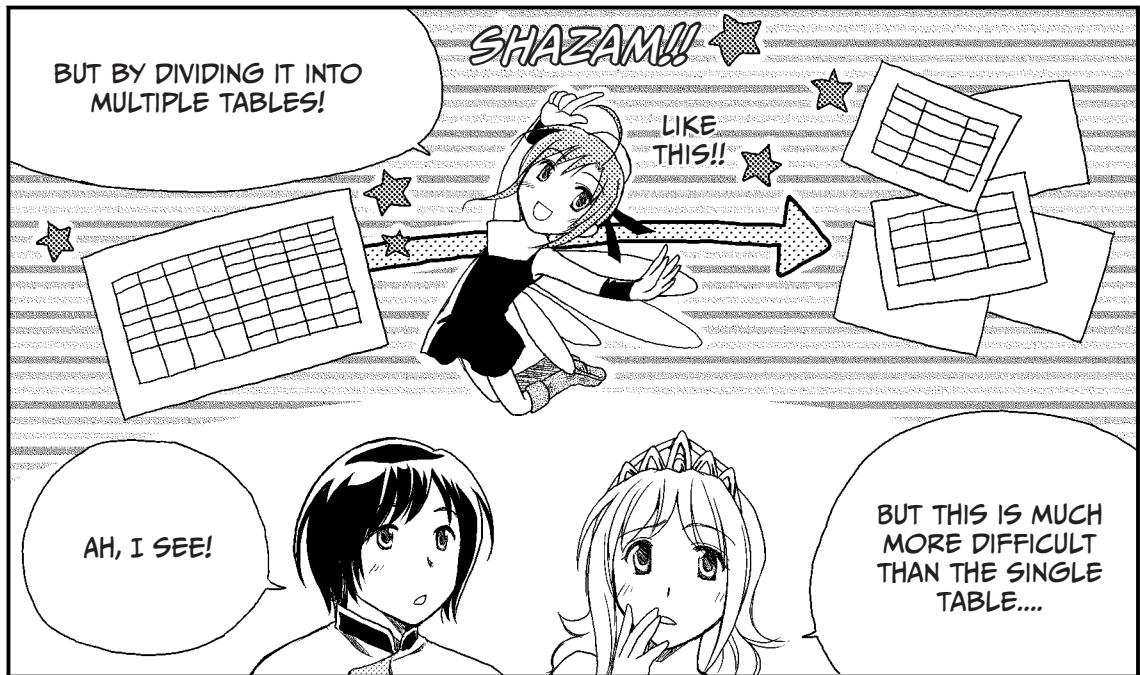
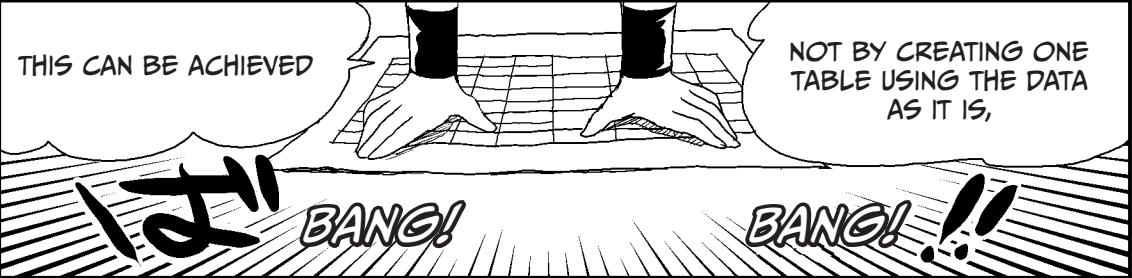


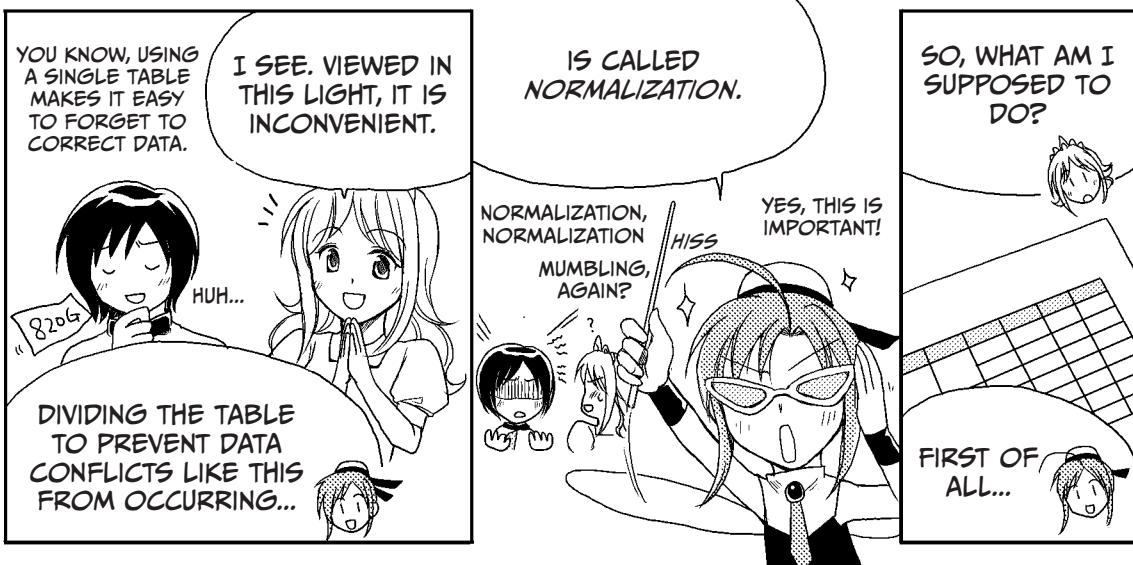
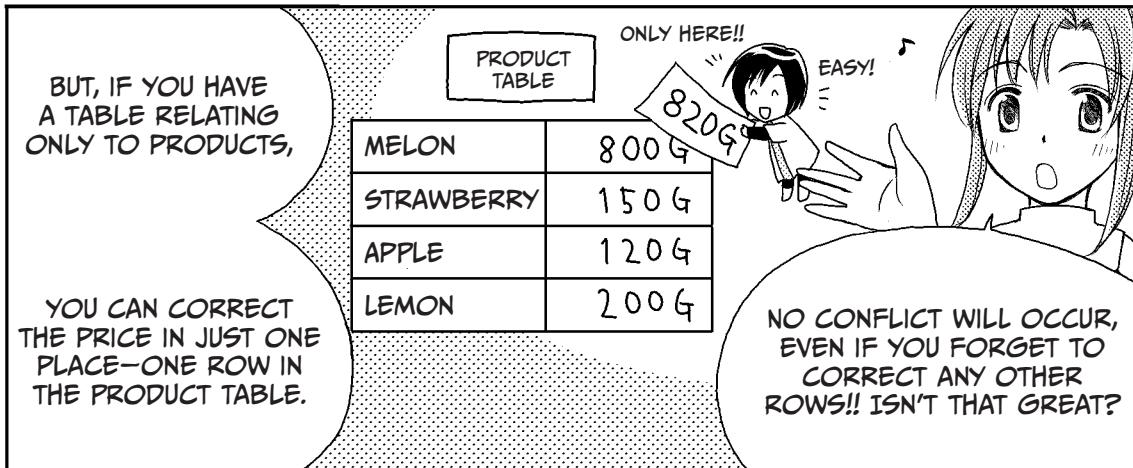
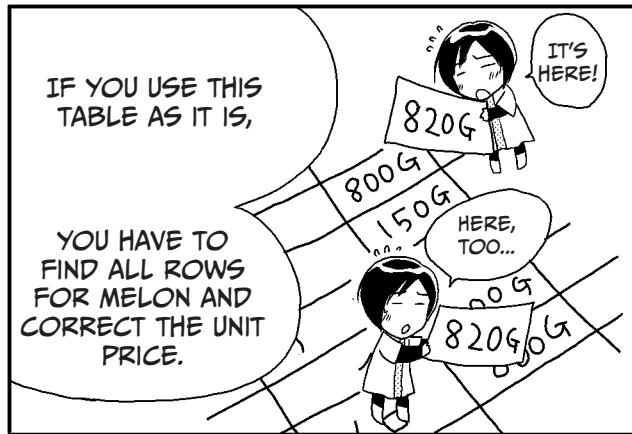
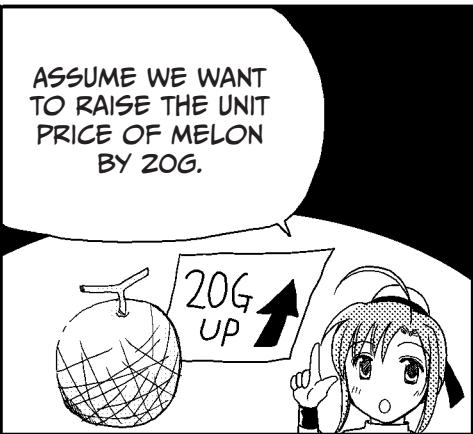
YOU SEE, THERE
ARE TWO ROWS
OF PRODUCT ITEMS
IN ONE ROW OF
DATA.

EXPORT DEST CODE	DESCRIPTION	PRODUCT CODE	PRODUCT NAME	UNIT PRICE	QUANTITY
12	THE KINGDOM OF MINANMI	101	MELON	800 G	300
7	ALPHA EMPIRE	102	STRAW BERRY	150 G	1,700
23	THE KINGDOM OF KOTOL	103	APPLE	200 G	500
25	THE KINGDOM OF MINANMI	104	LEMON	200 G	2,500
18	THE KINGDOM OF KOTOL	105			
3/10	12	THE KINGDOM OF KOTOL			
25		106			

AH...







LET'S TRY
CHANGING IT SO
THAT ONE ROW
HAS ONE VALUE.

REPEATED DATA IS A CLUE
THAT ROWS HAVE TO BE
DIVIDED.

SO, I'LL DIVIDE IT INTO...

ONE TABLE WITH DATE,
EXPORT DESTINATION
CODE, AND EXPORT
DESTINATION NAME...

SALES TABLE (FIRST NORMAL FORM (1))

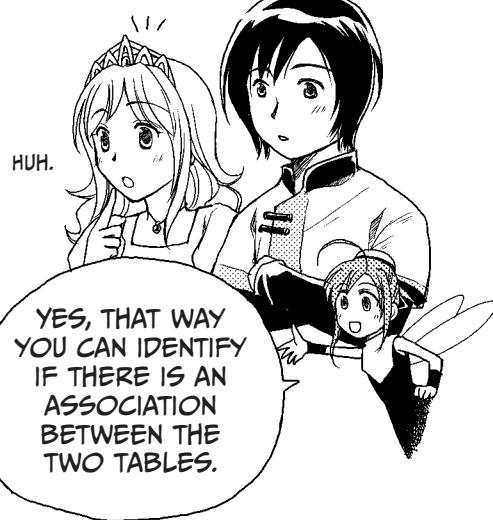
REPORT CODE	DATE	EXPORT DEST. CODE	EXPORT DEST. NAME
1101	3/5	12	THE KINGDOM OF MINANMI
1102	3/7	23	ALPHA EMPIRE
1103	3/8	25	THE KINGDOM OF RITOL
1104	3/10	12	THE KINGDOM OF MINANMI
1105	3/12	25	THE KINGDOM OF RITOL

AND ANOTHER TABLE WITH
PRODUCT CODE, PRODUCT
NAME, UNIT PRICE, AND
QUANTITY.

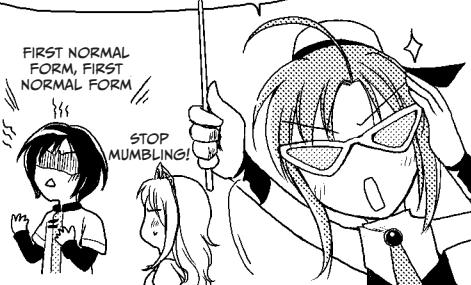
BUT THE REPORT
CODE IS PROVIDED
IN BOTH TABLES,
ISN'T IT?

SALES TABLE (FIRST NORMAL FORM (2))

REPORT CODE	PRODUCT CODE	PRODUCT NAME	UNIT PRICE	QUANTITY
1101	101	MELON	800G	1,100
1101	102	STRAWBERRY	150G	300
1102	103	APPLE	120G	1,700
1103	104	LEMON	200G	500
1104	101	MELON	800G	2,500
1105	103	APPLE	120G	2,000
1105	104	LEMON	200G	700

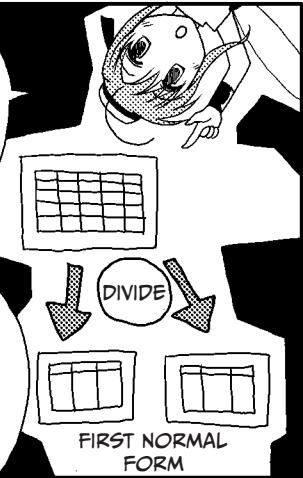


THE TABLE THAT RESULTS FROM A DIVISION LIKE THIS IS CALLED THE FIRST NORMAL FORM.



THE TABLE THAT HAS ROWS WITH TWO OR MORE VALUES BEFORE IT IS DIVIDED IS CALLED THE UNNORMALIZED FORM.

IT MEANS THAT THE FIRST NORMAL FORM IS CREATED BY DIVIDING THE UNNORMALIZED FORM.



LET'S SEE...

WAIT A MINUTE.

THESE ARE THE "FIRST NORMAL FORMS." DOES THAT MEAN THERE ARE THE "SECOND" AND "THIRD" NORMAL FORMS, TOO?

BINGO!

THE FIRST NORMAL FORM CANNOT BE USED AS A RELATIONAL DATABASE TABLE AS IT IS.

HANG IN THERE!

AH, I SEE...

COME ON!!

IT'S A LONG WAY!!

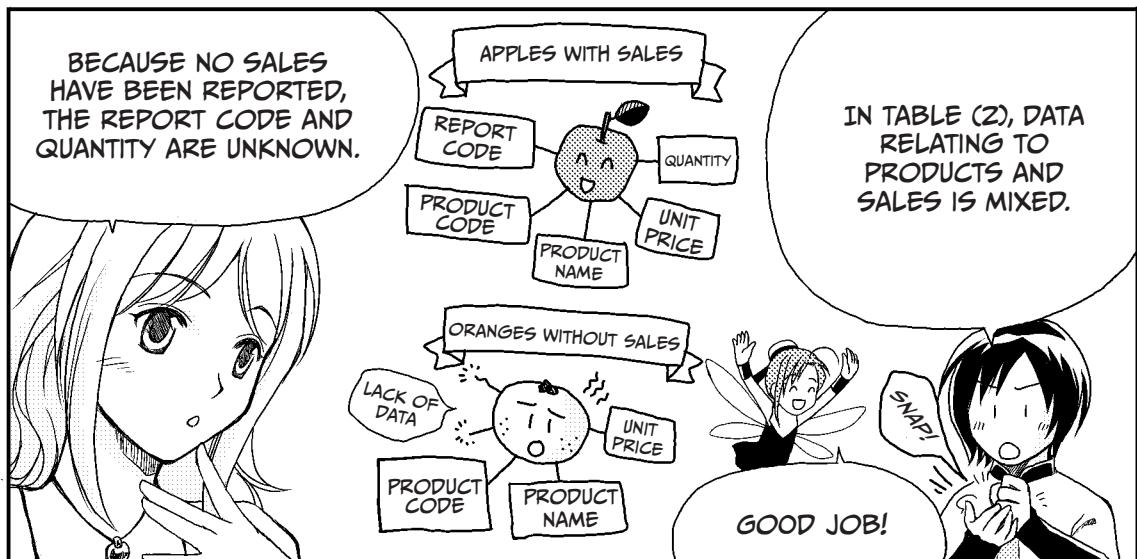
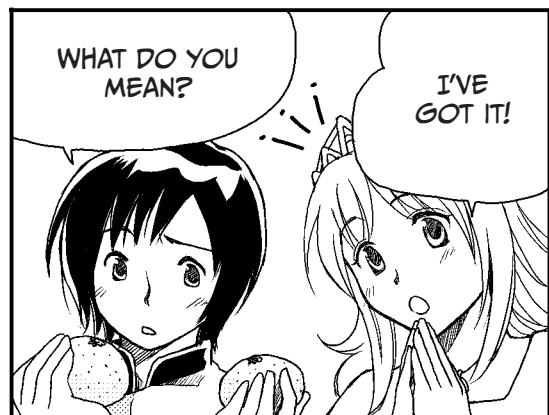
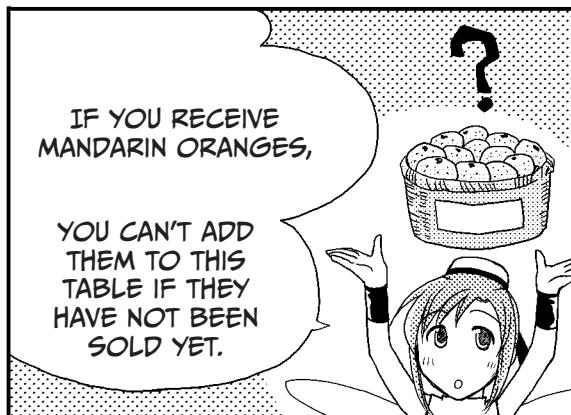
STUMBLE STUMBLE

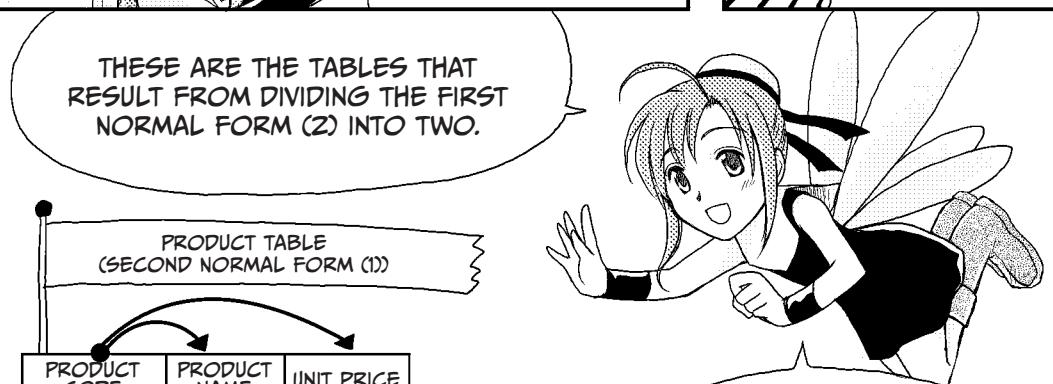
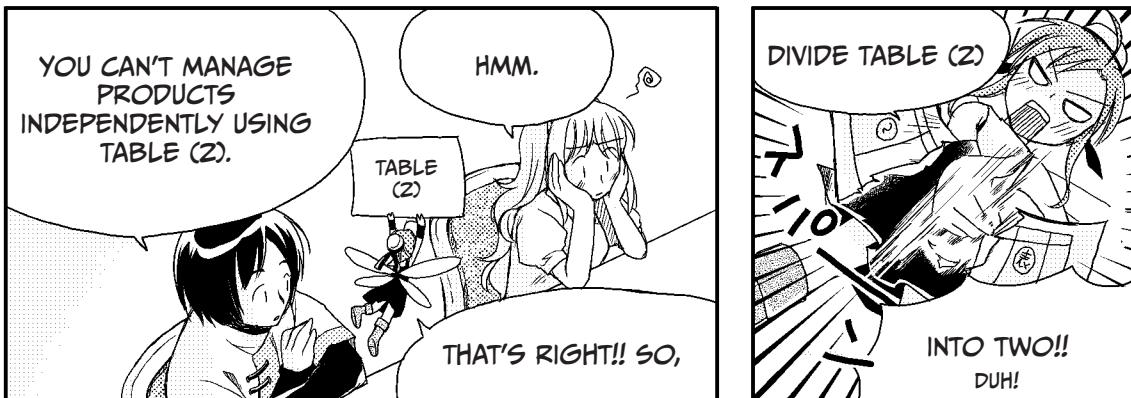
Mr. Relational Database



REPORT CODE	PRODUCT CODE	PRODUCT NAME	UNIT PRICE	QUANTITY
1101	101	MELON	800G	1,100
1101	102	STRAWBERRY	150G	300

SALES STATEMENT TABLE
(FIRST NORMAL FORM (1))



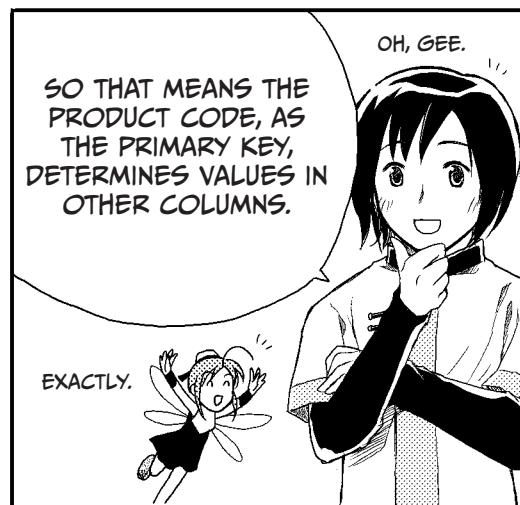


SALES STATEMENT TABLE (SECOND NORMAL FORM (2))

REPORT CODE	PRODUCT CODE	QUANTITY
1101	101	1,100
1101	102	300
1102	103	1,700
1103	104	500
1104	101	2,500
1105	103	2,000
1105	104	700



IF A VALUE IN THE PRODUCT CODE COLUMN IS DETERMINED, WE CAN FIND THE VALUES IN THE PRODUCT NAME AND UNIT PRICE COLUMNS.



FOR DATA RELATING TO SALES STATEMENT ITEMS IN TABLE (2),

AND IN THIS TABLE, THE PRIMARY KEY DETERMINES VALUES IN OTHER COLUMNS.

BUT...

FOR TABLE (2), CONSIDER THE COMBINATION OF REPORT CODE AND PRODUCT CODE AS A PRIMARY KEY.



PRIMARY KEY	
REPORT CODE	PRODUCT CODE

IN SOME CASES, TWO KINDS OF PRODUCTS SELL AT THE SAME TIME...

IN OTHER CASES, ONE KIND OF PRODUCT SELLS IN DIFFERENT QUANTITIES.

THIS MEANS...

YOU DIVIDE THE TABLE SO THAT WHEN A PRIMARY KEY IS DETERMINED, VALUES IN OTHER COLUMNS ARE DETERMINED.

UNDERSTAND?

①

PRODUCT CODE	PRODUCT NAME	UNIT PRICE
--------------	--------------	------------

PRIMARY KEY

②

REPORT CODE	PRODUCT CODE	QUANTITY
-------------	--------------	----------

PRIMARY KEY

I SEE.

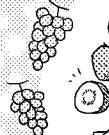
THE TABLE THAT RESULTS FROM DIVISION ACCORDING TO THIS RULE IS CALLED



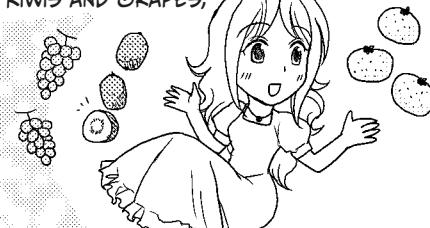
THE SECOND NORMAL FORM.

WE CAN ADD THE MANDARIN ORANGES WE WERE TALKING ABOUT EARLIER TO THE SECOND NORMAL FORM (1).

WE CAN ALSO ADD KIWIS AND GRAPES,



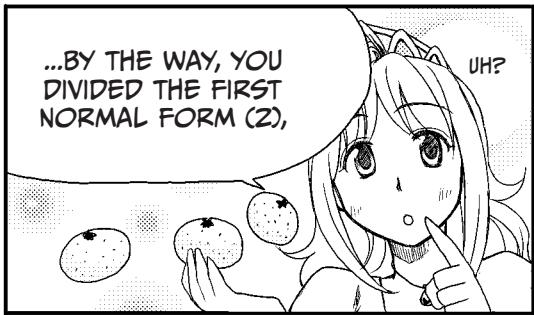
WHICH HAVE NOT BEEN SOLD YET!



EVEN IF THE PRICE OF MELON CHANGES, WE JUST CORRECT THE DATA ON ONE ROW, RIGHT?

820G

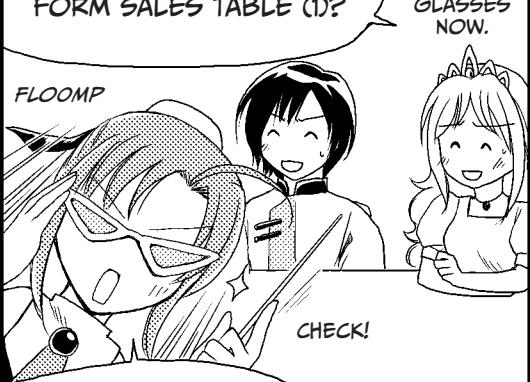
...BY THE WAY, YOU DIVIDED THE FIRST NORMAL FORM (2),



UH?

SO ISN'T IT NECESSARY TO DIVIDE THE FIRST NORMAL FORM SALES TABLE (1)?

OH, YOU ARE WEARING GLASSES NOW.



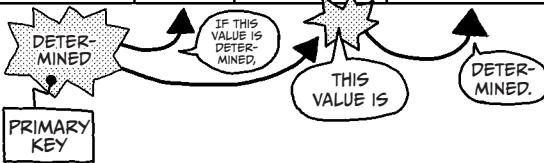
GOOD POINT!

CHECK!

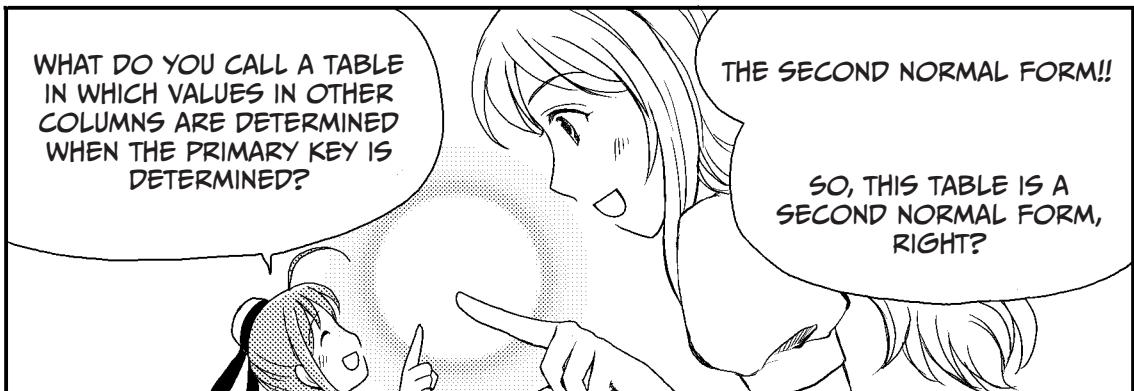
YEAH!!

SALES TABLE
(FIRST NORMAL FORM (1))

REPORT CODE	DATE	EXPORT DEST. CODE	EXPORT DEST. NAME
1101	3/5	12	THE KINGDOM OF MINANMI
1102	3/7	23	ALPHA EMPIRE
1103	3/8	25	THE KINGDOM OF RITOL
1104	3/10	12	THE KINGDOM OF MINANMI
1105	3/12	25	THE KINGDOM OF RITOL

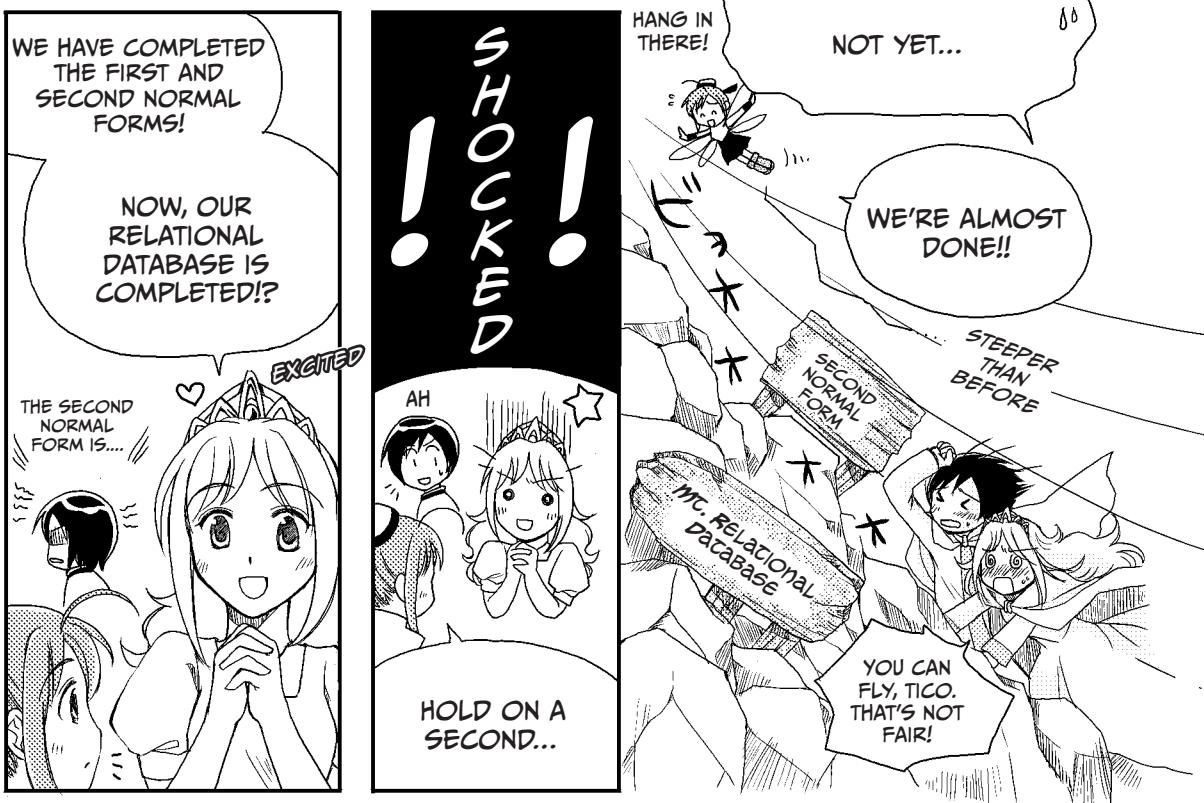


FOR THIS TABLE, IF ONE VALUE IN REPORT CODE IS DETERMINED, ALL OTHER VALUES IN DATE, EXPORT DESTINATION CODE, AND EXPORT DESTINATION NAME ARE DETERMINED.

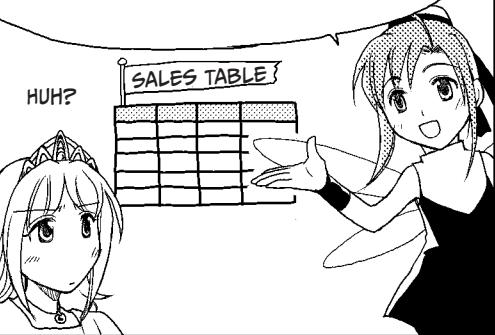


SALES TABLE (FIRST NORMAL FORM (1))			
REPORT CODE	DATE	EXPORT DEST. CODE	EXPORT DEST. NAME
		THE KINGDOM OF MINANMI	
		ALPHA EMPIRE	
		THE KINGDOM OF RITOL	

SALES TABLE (SECOND NORMAL FORM (3))			
REPORT CODE	DATE	EXPORT DEST. CODE	EXPORT DEST. NAME
		THE KINGDOM OF MINANMI	
		ALPHA EMPIRE	
		THE KINGDOM OF RITOL	



LOOK AT THE SECOND NORMAL FORM (3) AGAIN.

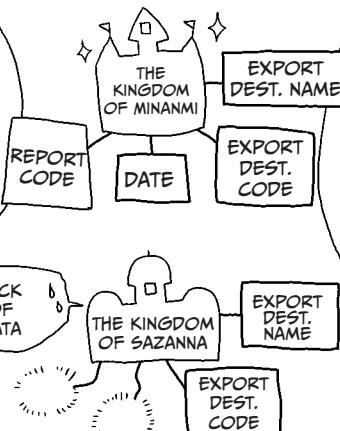


YOU CAN'T MANAGE EXPORT DESTINATIONS WITH THIS TABLE.

THINK,
THINK,
THINK...
AH!

SALES TABLE (SECOND NORMAL FORM (3))			
REPORT CODE	DATE	EXPORT DEST. CODE	EXPORT DEST. NAME
1101	3/5	12	THE KINGDOM OF MINAMI
	3/7	23	ALPHA EMPIRE
	3/8	25	THE KINGDOM OF RITOL

THE KINGDOM OF SAZANNA, TO WHICH NO FRUIT HAS BEEN EXPORTED, CANNOT BE MANAGED BY ADDING IT TO THIS TABLE.



IN TABLE (3), DATA RELATING TO EXPORT DESTINATIONS AND SALES IS MIXED.

HMM.

HOW CAN WE MANAGE EXPORT DESTINATIONS INDEPENDENTLY?

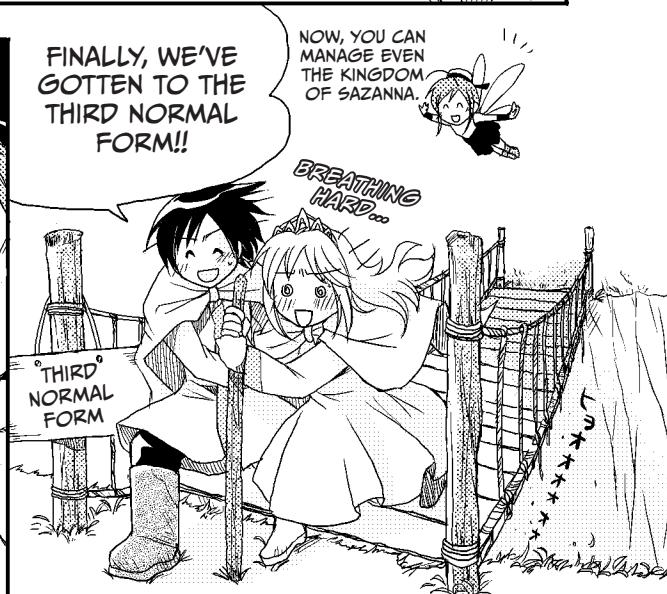
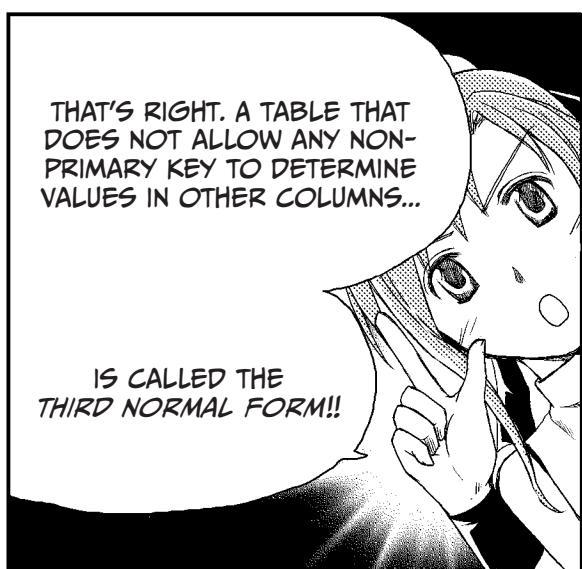
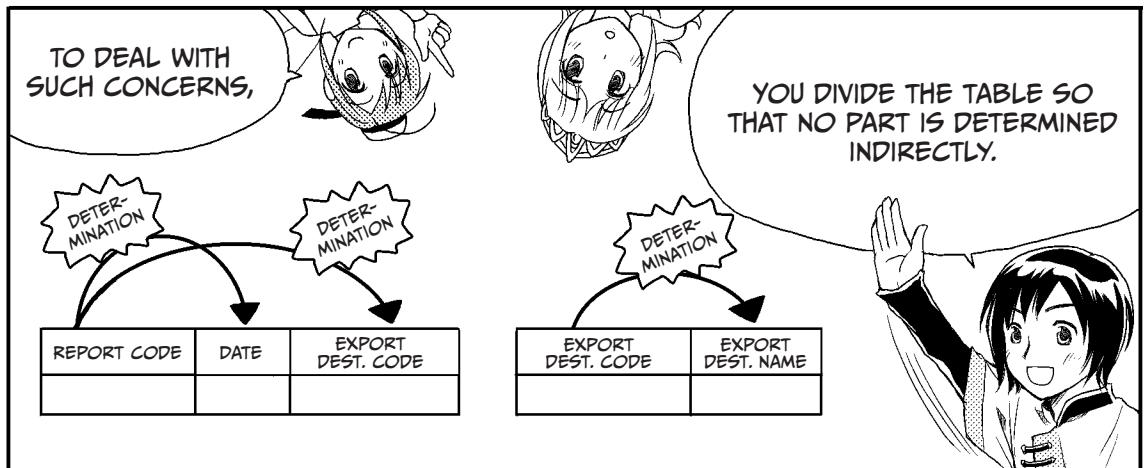
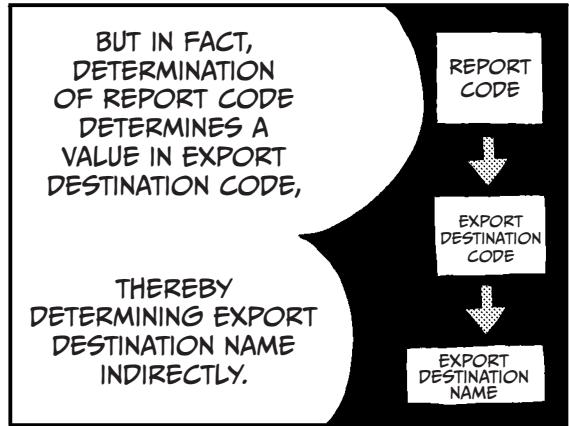
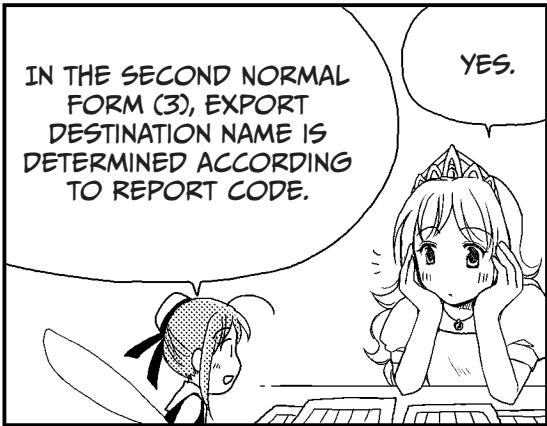
AGAIN, DIVIDE IT!



SALES TABLE (THIRD NORMAL FORM (1))		
REPORT CODE	DATE	EXPORT DEST. CODE
1101	3/5	12
1102	3/7	23
1103	3/8	25
1104	3/10	12
1105	3/12	25

EXPORT DESTINATION TABLE (THIRD NORMAL FORM (2))	
EXPORT DEST. CODE	EXPORT DEST. NAME
12	THE KINGDOM OF MINAMI
23	ALPHA EMPIRE
25	THE KINGDOM OF RITOL

SHAZAM!



SALES TABLE

REPORT CODE	DATE	EXPORT DEST. CODE
1101	3/5	12
1102	3/7	23
1103	3/8	25
1104	3/10	12
1105	3/12	25

EXPORT DESTINATIONS TABLE

EXPORT DEST. CODE	EXPORT DESTINATION NAME
12	THE KINGDOM OF MINANMI
23	ALPHA EMPIRE
25	THE KINGDOM OF RITOL

SALES STATEMENT TABLE

REPORT CODE	PRODUCT CODE	QUANTITY
1101	101	1,100
1101	102	300
1102	103	1,700
1103	104	500
1104	101	2,500
1105	103	2,000
1105	104	700

PRODUCT TABLE

PRODUCT CODE	PRODUCT NAME	UNIT PRICE
101	MELON	800G
102	STRAWBERRY	150G
103	APPLE	120G
104	LEMON	200G

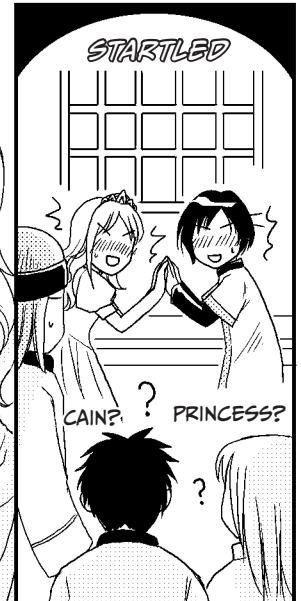
A RELATIONAL DATABASE NORMALLY USES TABLES DIVIDED UP TO THE THIRD NORMAL FORM.

NOW, OUR DATABASE TABLE IS COMPLETE!

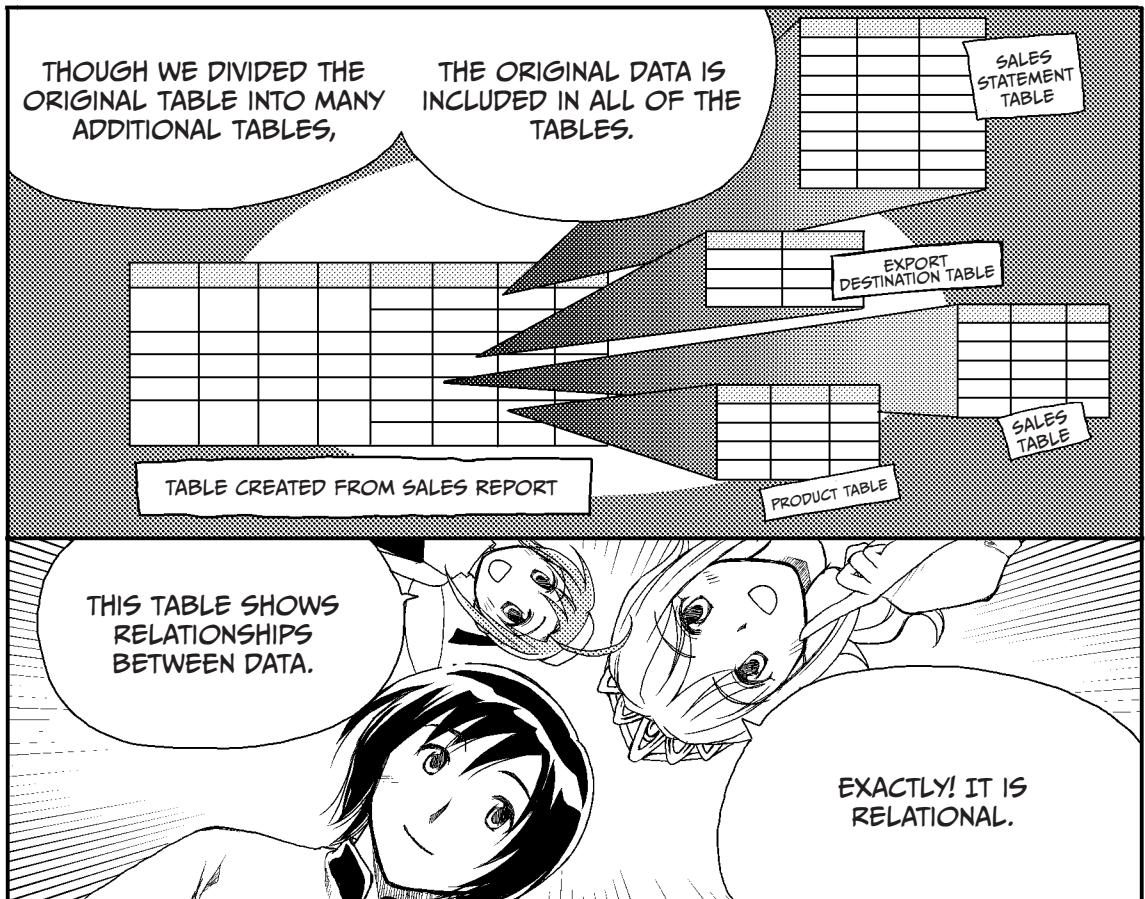
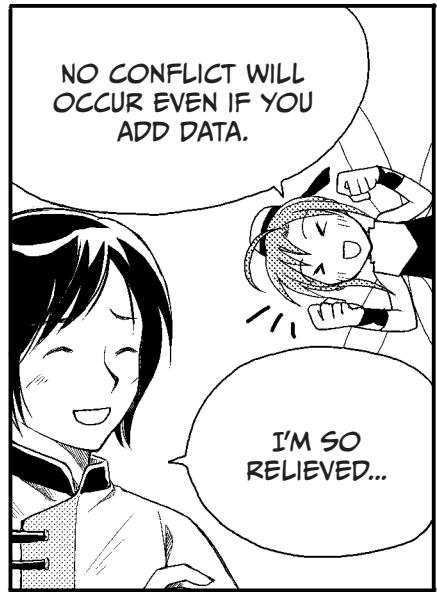
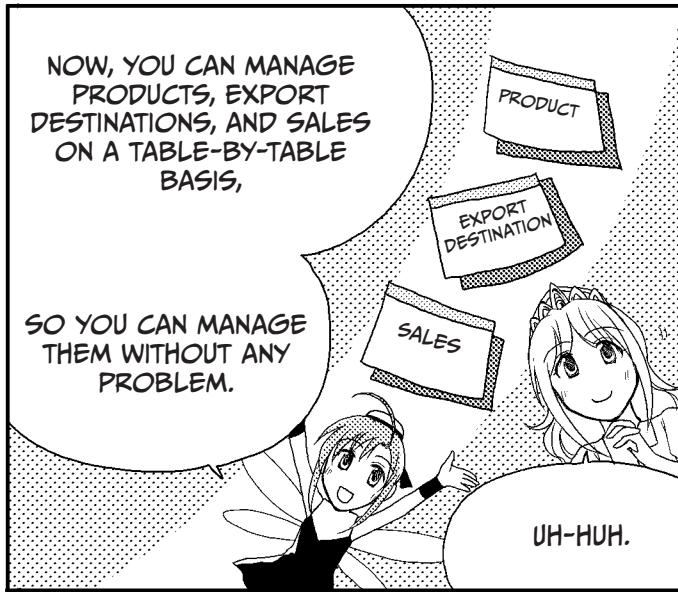


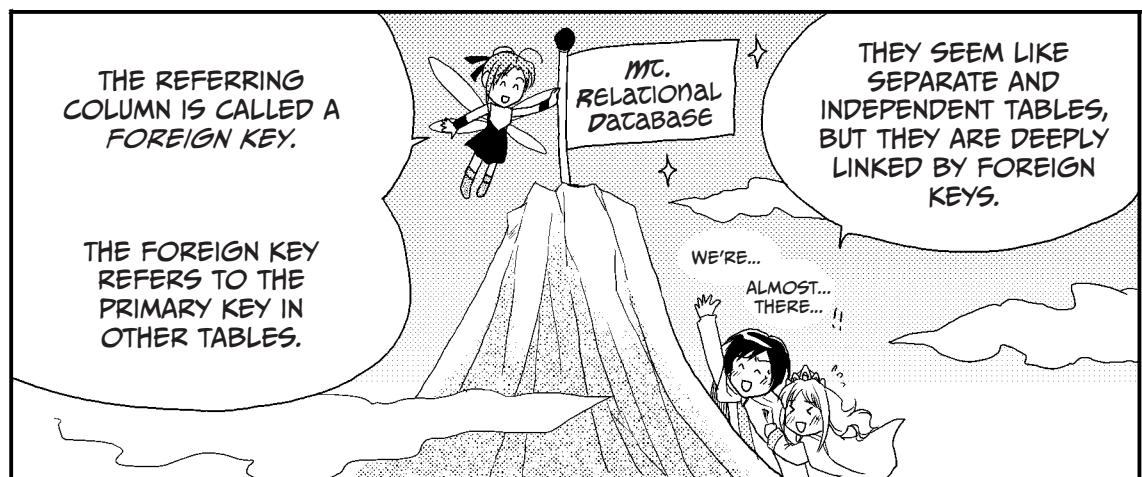
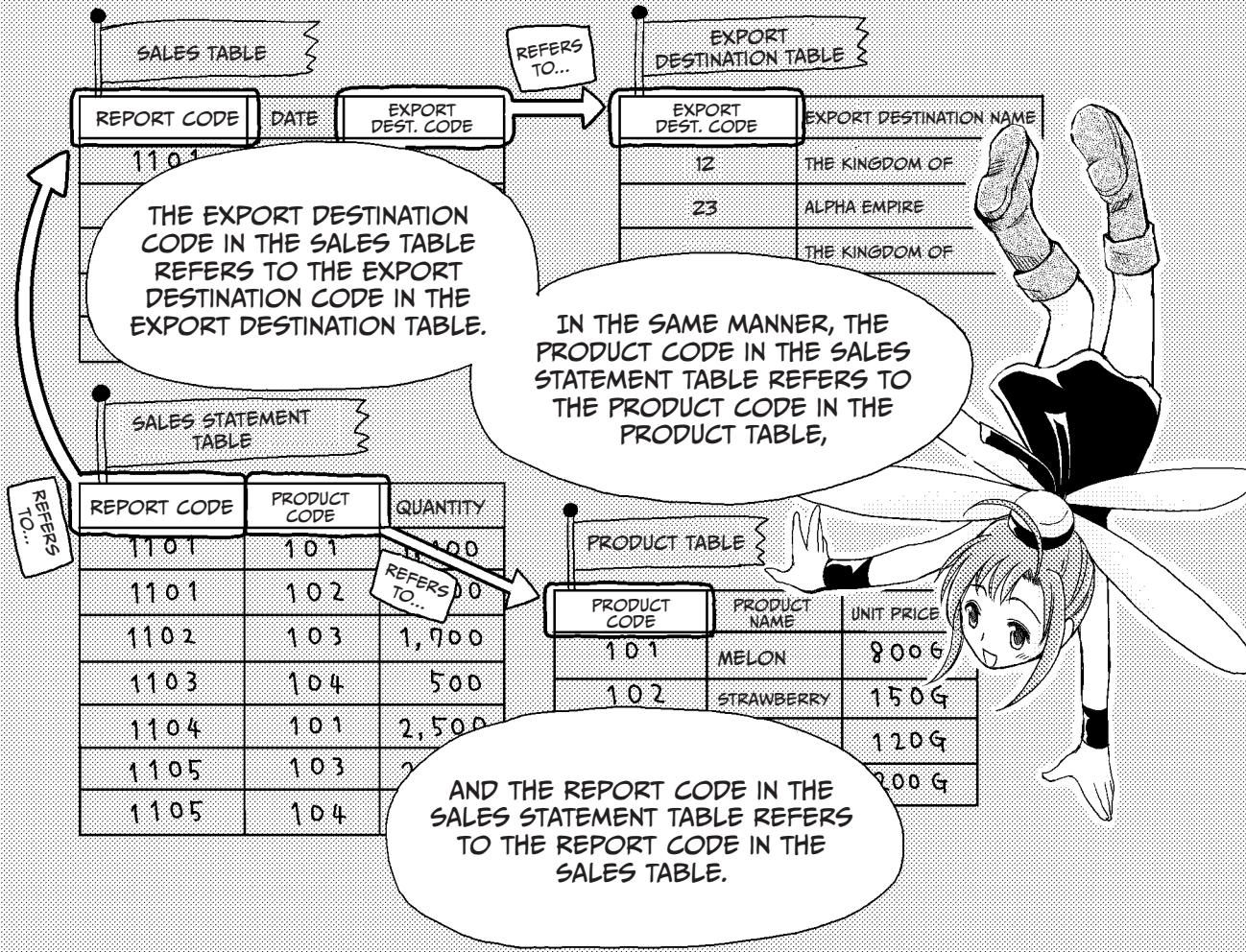
THESE ARE THE TABLES THAT RESULT WHEN YOU DIVIDE A TABLE UP TO THE THIRD NORMAL FORM.

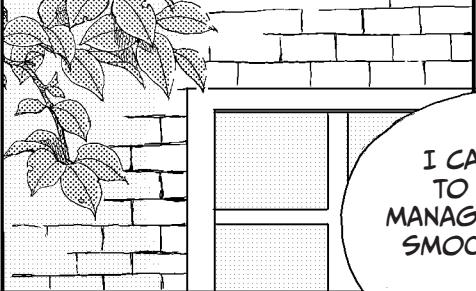
UP HIGH!



+ - ZOMBIE! ?



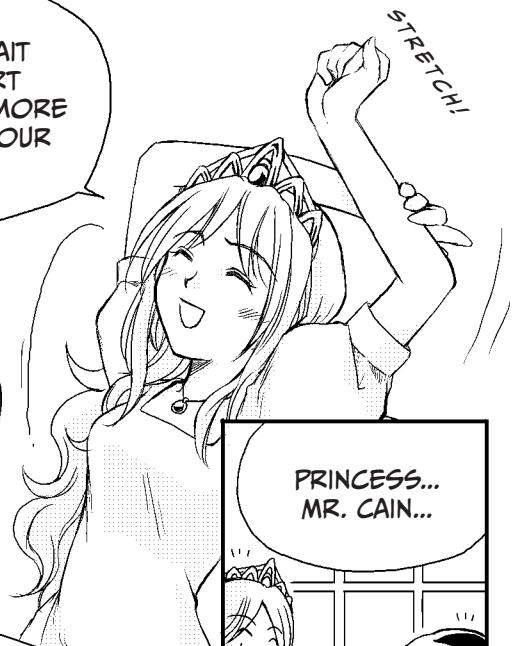




I CAN HARDLY WAIT
TO MAKE EXPORT
MANAGEMENT RUN MORE
SMoothly USING OUR
DATABASE.



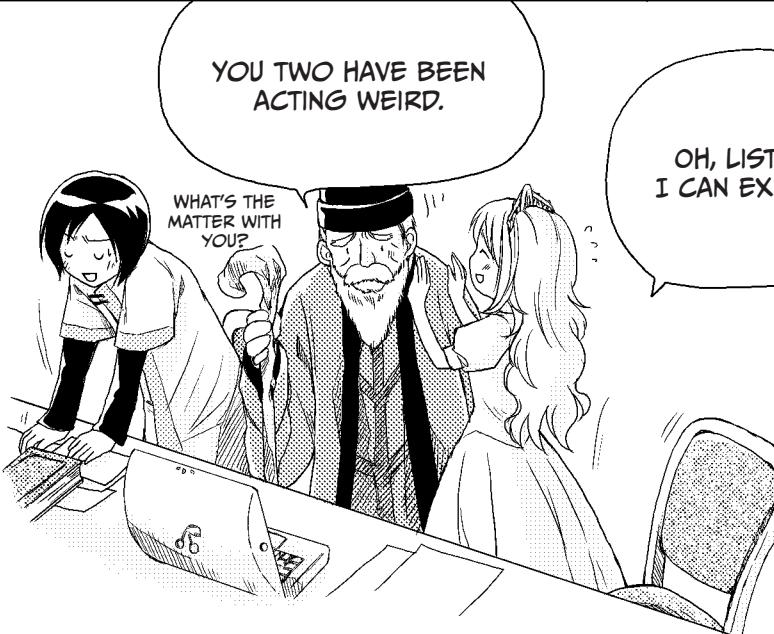
YES.



PRINCESS...
MR. CAIN...



YOU TWO HAVE BEEN
ACTING WEIRD.



OH, LISTEN...
I CAN EXPLAIN.



ME? OF
COURSE
NOT.

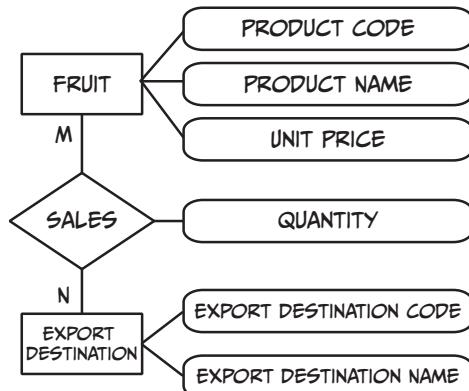
WHAT IS THE E-R MODEL?



Princess Ruruna and Cain have figured out the actual condition of the Kingdom of Kod using an E-R (entity-relationship) model. When you try to create a database yourself, the first step is to determine the conditions of the data you are trying to model.

Using the E-R model, try to define an entity in your data. An *entity* is a real-world object or “thing,” such as *fruit* or *export destination*.

In addition, an E-R model shows the relationship between entities. Princess Ruruna and Cain performed their analysis on the assumption that there was a relationship called *sales* between fruit and export destination. Fruit is exported to multiple export destinations, while each export destination also imports multiple kinds of fruit. For this reason, an analysis was made for the E-R model assuming that there was a relationship called *many-to-many* between fruit and export destinations. M fruit have a relationship with N export destinations. The number of associations between entities is called *cardinality*.

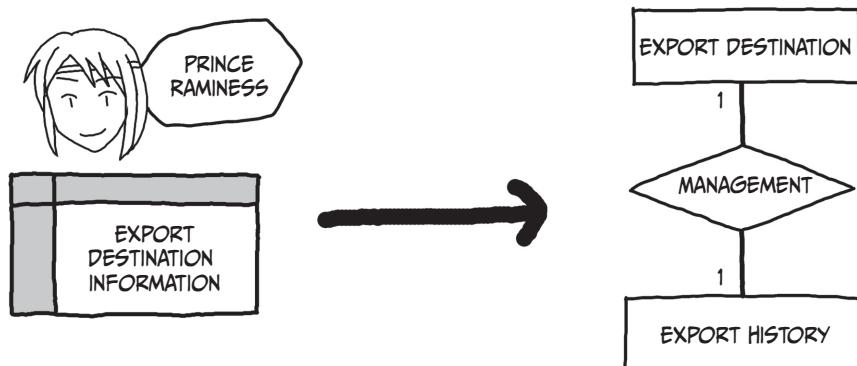


HOW TO ANALYZE THE E-R MODEL

How would you perform analyses in the cases below? Think about it.

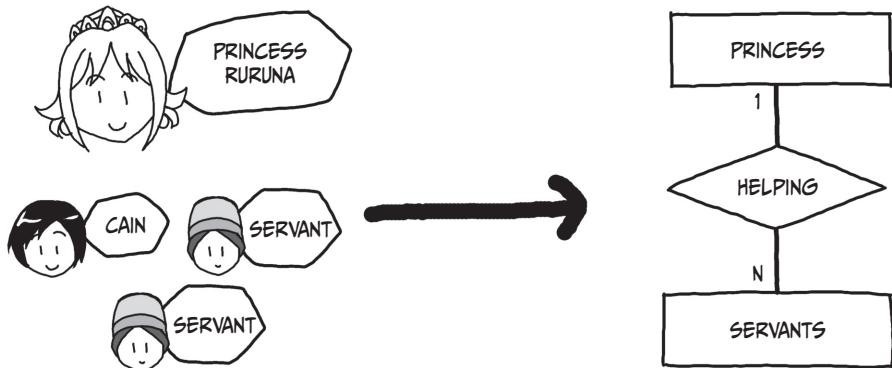
CASE 1: ONE-TO-ONE RELATIONSHIP

One export destination manages one piece of export history information. This kind of relationship is called a *one-to-one* relationship.



CASE 2: ONE-TO-MANY RELATIONSHIP

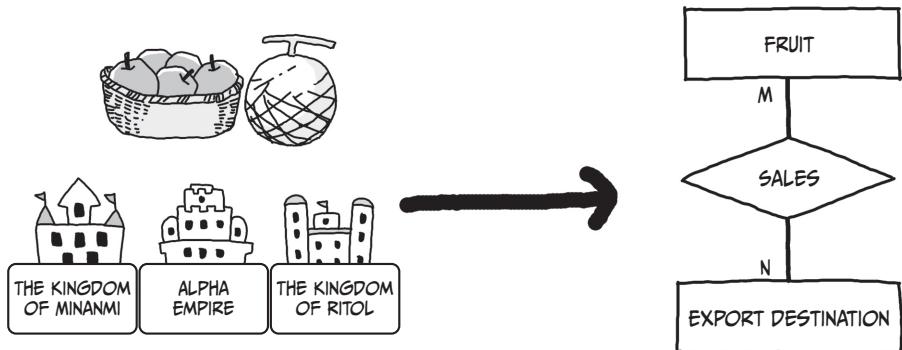
Multiple servants serve one princess. The servants do not serve any other princess or even the king.



This kind of relationship is called a *one-to-many* relationship.

CASE 3: MANY-TO-MANY RELATIONSHIP

Fruit is exported to multiple export destinations. The export destinations import multiple kinds of fruit.



This kind of relationship is called a *many-to-many* relationship.

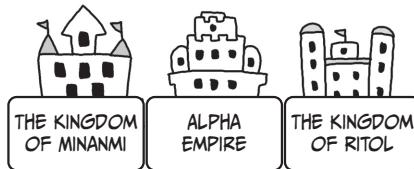


QUESTIONS

How well do you understand the E-R model? Analyze and draw an E-R model for each of the cases below. The answers are on page 82.

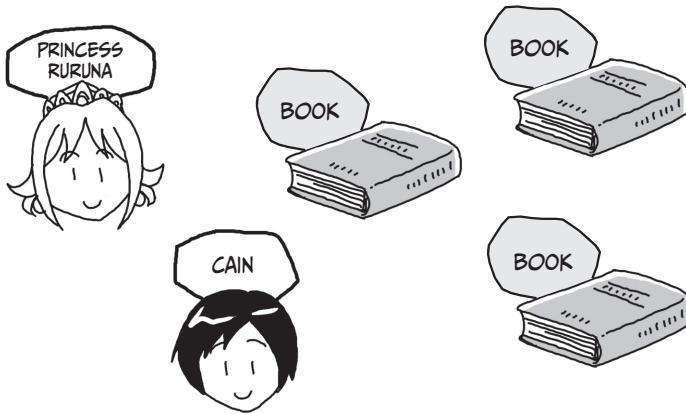
Q1

One staff member manages multiple customers. One customer will never be contacted by more than one staff member.



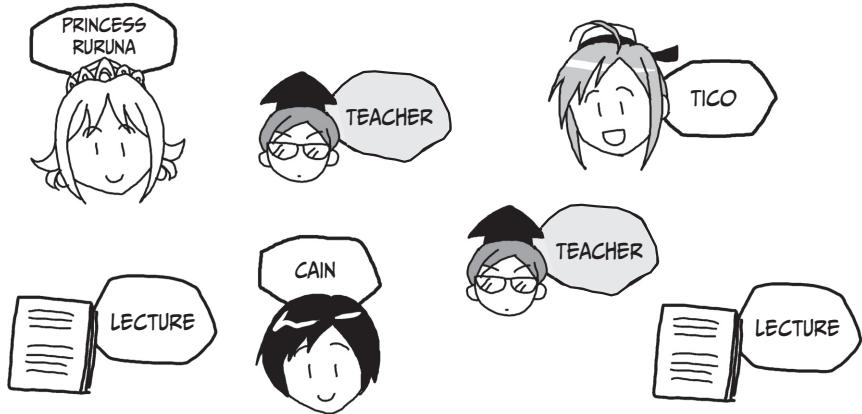
Q2

One person can check out multiple books. Books can be checked out to multiple students at different times.



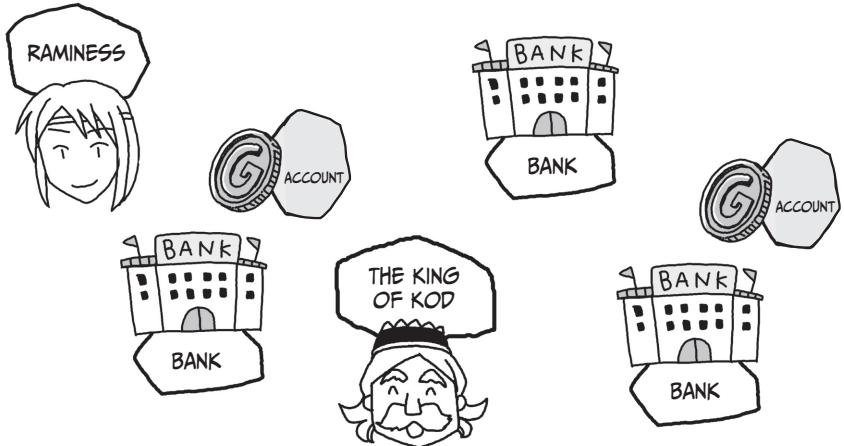
Q3

Each student attends multiple lectures. Each lecture is attended by multiple students. One teacher gives multiple lectures. Each lecture is given by one teacher.



Q4

Each customer can open multiple deposit accounts. Each deposit account is opened by one customer. Each bank manages multiple deposit accounts. Each deposit account is managed by one bank.



Keep in mind that E-R model-based analysis does not necessarily produce one “correct” result. There can be many ways to logically organize data to reflect real-world conditions.

NORMALIZING A TABLE



Princess Ruruna and Cain learned about normalization, the process of tabulating data from the real world for a relational database. It is necessary to normalize data in order to properly manage a relational database. Normalization is summarized here (the shaded fields are *primary keys*).

UNNORMALIZED FORM

Report code	Date	Export destination code	Export destination name	Product code	Product name	Unit price	Quantity
-------------	------	-------------------------	-------------------------	--------------	--------------	------------	----------

FIRST NORMAL FORM

Report code	Date	Export destination code	Export destination name	
Report code	Product code	Product name	Unit price	Quantity

SECOND NORMAL FORM

Report code	Date	Export destination code	Export destination name
Report code	Product code		Quantity
Product code		Product name	Unit price

THIRD NORMAL FORM

Report code	Date	Export destination code
Export destination code		Export destination name
Report code	Product code	Quantity
Product code	Product name	Unit price

The *unnormalized form* is a table in which items that appear more than once have not been removed. We've seen that you cannot manage data well using this kind of table for a relational database. Consequently, you need to divide the table.

The *first normal form* refers to a simple, two-dimensional table resulting from division of the original, unnormalized table. You can consider it to be a table with one item in each cell. The table is divided so that no items will appear more than once.

The *second normal form* refers to a table in which a key that can identify data determines values in other columns. Here, it is the *primary key* that determines values in other columns.

In a relational database, a value is called *functionally dependent* if that value determines values in other columns. In the second normal form, the table is divided so that values in other columns are functionally dependent on the primary key.

In the *third normal form*, a table is divided so that a value is not determined by any non-primary key. In a relational database, a value is called *transitively dependent* if that value determines values in other columns indirectly, which is part of functionally dependent operation. In the third normal form, the table is divided so that transitively dependent values are removed.



QUESTIONS

It is important to be able to design a relational database table for various situations, so let's look at some examples of normalizing tables. Determine how the table was normalized in each of the cases below. The answers are on page 82.

Q5

The following table manages book lending like the example in Q2. To what stage is it normalized?

Lending code	Date	Student code	Student name	Student address	Department	Entrance year
--------------	------	--------------	--------------	-----------------	------------	---------------

ISBN	Book name	Author name	Publication date	Total page count
------	-----------	-------------	------------------	------------------

Lending code	ISBN	Quantity
--------------	------	----------

Q6

The following table also shows a book lending situation. To what stage is it normalized?

Lending code	Date	Student code
--------------	------	--------------

Student code	Student name	Student address	Department	Entrance year
--------------	--------------	-----------------	------------	---------------

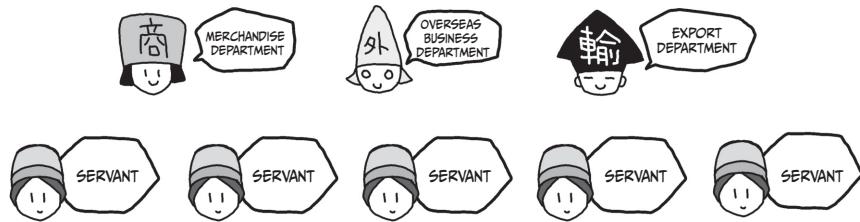
ISBN	Book name	Author name	Publication date	Total page count
------	-----------	-------------	------------------	------------------

Lending code	ISBN	Quantity
--------------	------	----------

Q7

The following table shows monthly sales for each staff member. Each department has multiple staff members. A staff member can only be part of one department. Normalize this table to the third normal form.

Staff member code	Staff member name	Month	Member's sales	Department code	Department name
-------------------	-------------------	-------	----------------	-----------------	-----------------

**Q8**

The following table represents an order-receiving system. Normalize it to the third normal form. However, process one customer per order-taking code. You can process multiple products based on one order-taking code. In addition, one order-taking code should correspond to only one representative.

Order-taking code	Date	Customer code	Customer name	Product code	Product name	Unit price	Representative code	Representative name	Quantity
-------------------	------	---------------	---------------	--------------	--------------	------------	---------------------	---------------------	----------

Q9

The following table represents an order-receiving system. Normalize it to the third normal form. Assume that products are classified by product code.

Order-taking code	Date	Customer code	Customer name	Product code	Product name	Unit price	Product classification code	Product classification name	Quantity
-------------------	------	---------------	---------------	--------------	--------------	------------	-----------------------------	-----------------------------	----------

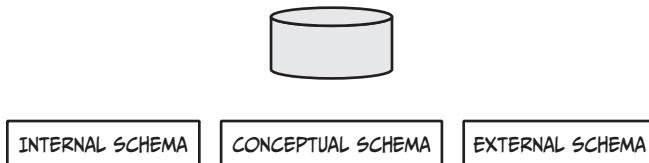
STEPS FOR DESIGNING a DATABASE

You have learned how to design a database! However, you have to do more than just that. You need to design a detailed file structure inside the database and devise methods for importing and exporting data. In general, you can divide the whole database design into three parts: conceptual schema, internal schema, and external schema.

The *conceptual schema* refers to a method that models the actual world. Namely, it is a way to determine the logical structure of a database. The conceptual schema is designed taking into consideration an E-R model-based understanding of the actual world and normalization of a table.

The *internal schema* refers to a database viewed from the inside of a computer. Namely, it is a way to determine the physical structure of a database. The internal schema is designed after creating a method to search the database at high speed.

The *external schema* refers to a database as viewed by users or applications. The external schema is designed after creating data required for application programs.



Princess Ruruna and Cain have designed a database with a focus on the conceptual schema in this chapter. They are in the midst of improving the database.

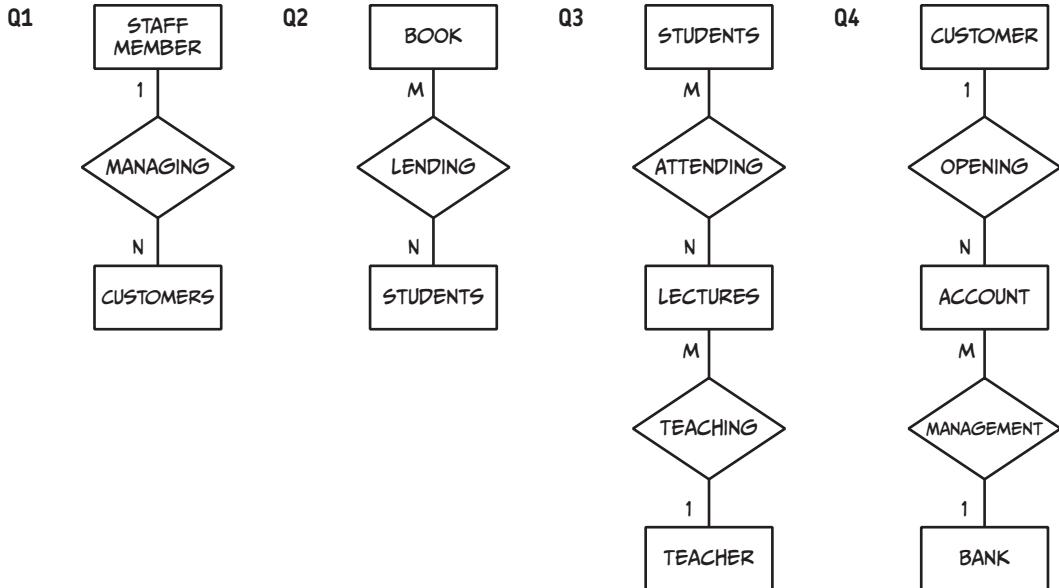
Now that you've completed the basic design of a database, we'll go straight to using the database in the next chapter.

SUMMARY



- An *E-R model* is used to analyze entities and relationships.
- Relationships between entities can be one-to-one, one-to-many, and many-to-many.
- The data in a table must be normalized before you can use it to create a relational database.
- The design of a database can be divided into three types: conceptual schema, internal schema, and external schema.

ANSWERS



Q5 Second normal form

Q6 Third normal form

Q7

Staff member code	Month	Member's sales
-------------------	-------	----------------

Staff member code	Staff member name	Department code
-------------------	-------------------	-----------------

Department code	Department name
-----------------	-----------------

Q8

Order-taking code	Date	Customer code	Representative code
-------------------	------	---------------	---------------------

Customer code	Customer name
---------------	---------------

Order-taking code	Product code	Quantity
-------------------	--------------	----------

Product code	Product name	Unit price
--------------	--------------	------------

Representative code	Representative name
---------------------	---------------------

Q9

Order-taking code	Date	Customer code
-------------------	------	---------------

Customer code	Customer name
---------------	---------------

Order-taking code	Product code	Quantity
-------------------	--------------	----------

Product code	Product classification code	Product name	Unit price
--------------	-----------------------------	--------------	------------

Product classification code	Product classification name
-----------------------------	-----------------------------

DESIGNING A DATABASE

In this chapter, you learned how to design a relational database. However, there are other database design methods. Usability and efficiency of a database depend on an analysis and design method. Therefore, it is important to create an appropriate database in the design stage.

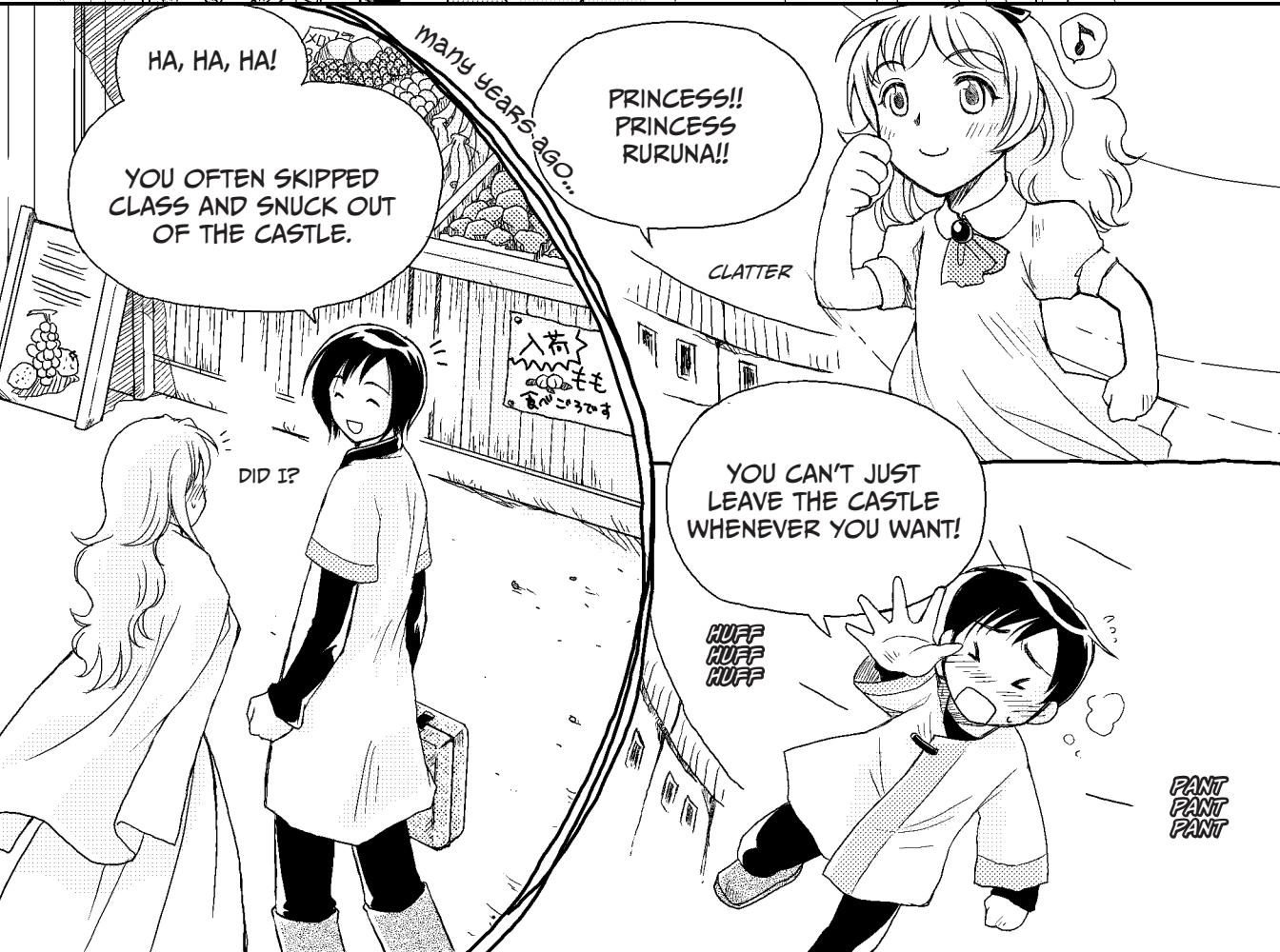
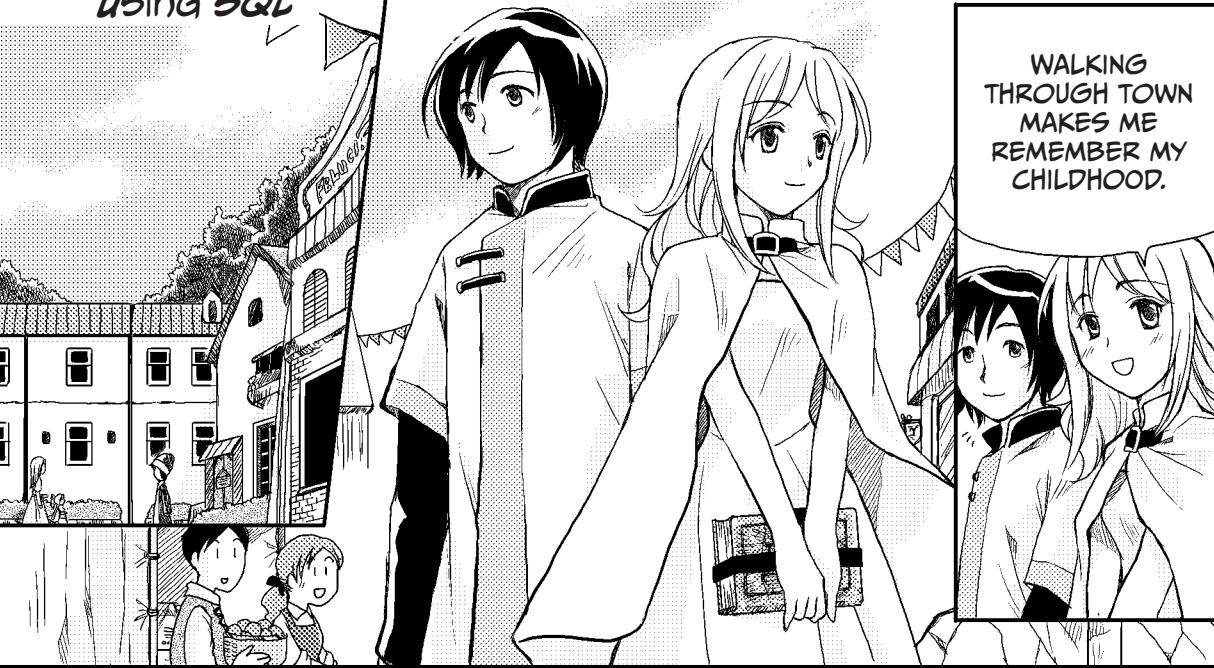
In the database design stage, you need to perform various tasks in addition to table design. For example, you need to consider a datatype to use in the table. You may also need to specify columns indicating numerical values, currencies, and character strings. In addition, you need to devise a search method so you can carry out fast searches. Sometimes, you must create a design while keeping physical file organization in mind. And you have to control which users can access the database to ensure security. There are many factors you need to think about when designing a database. We'll look at some of these factors in the following chapters.

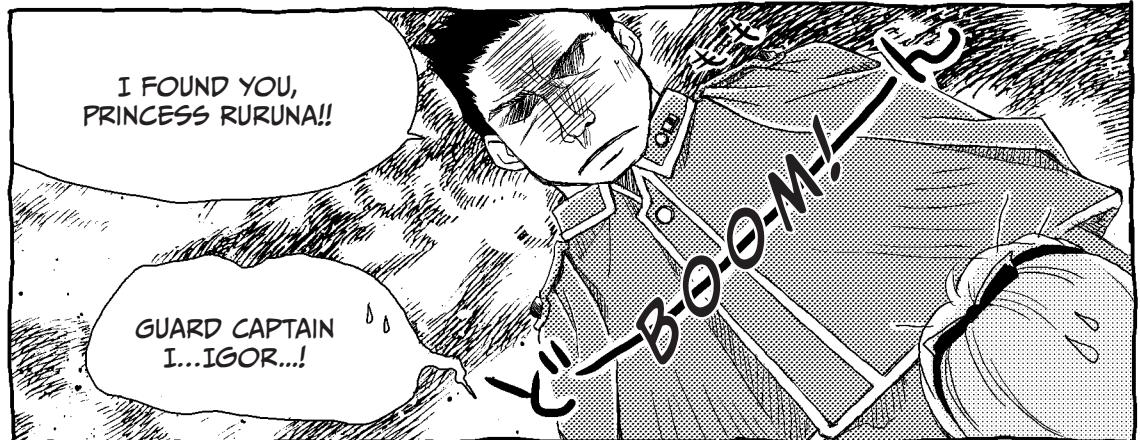
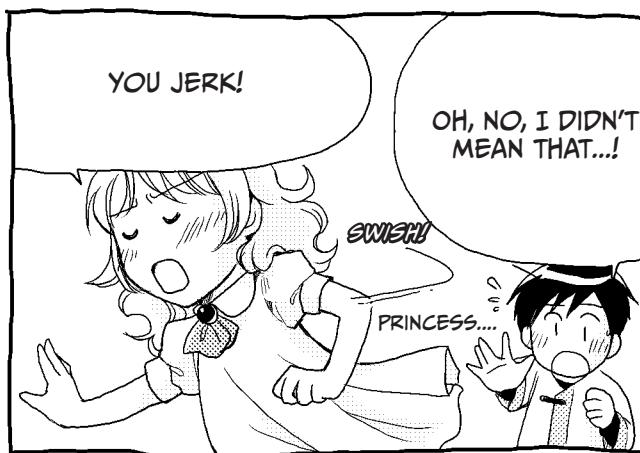
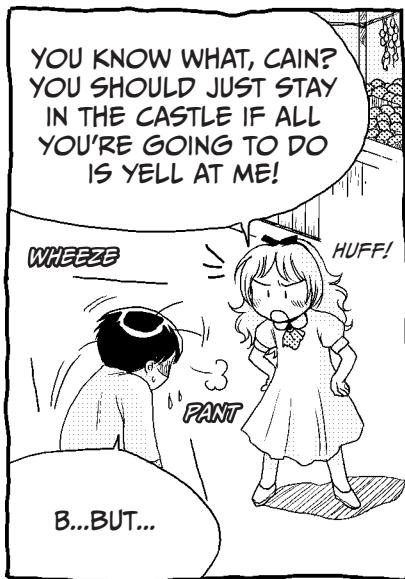
4

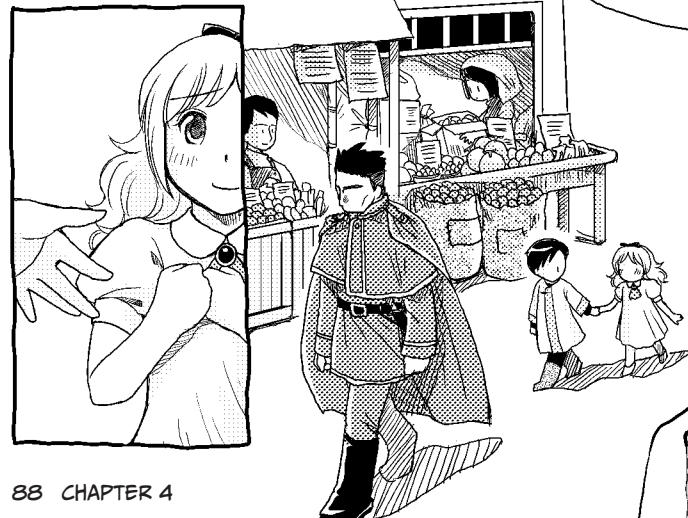
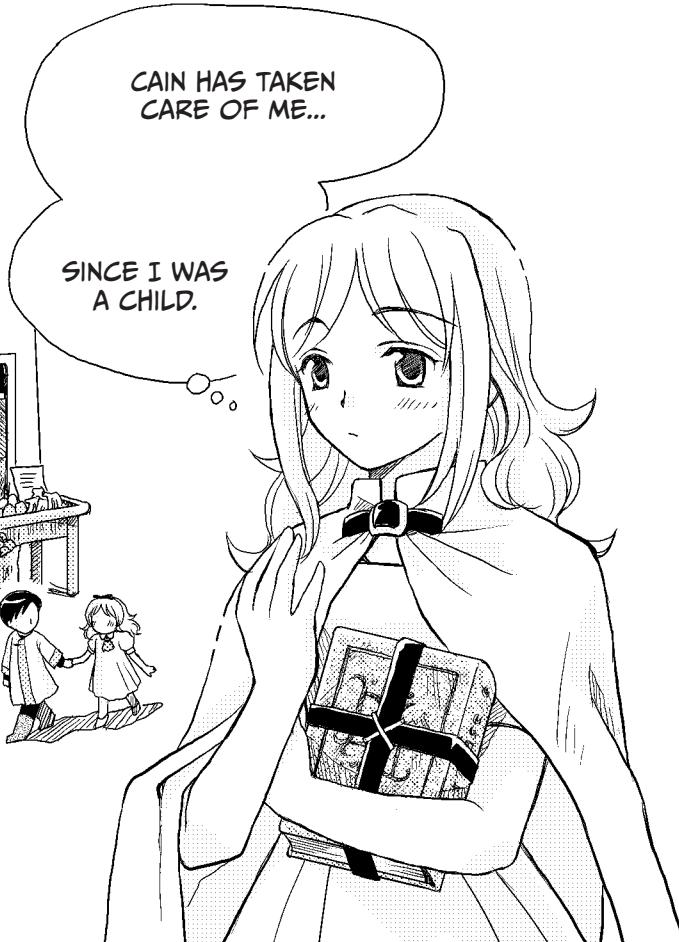
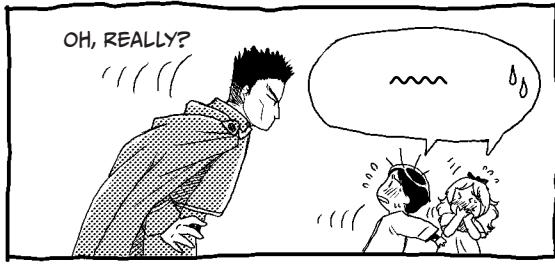
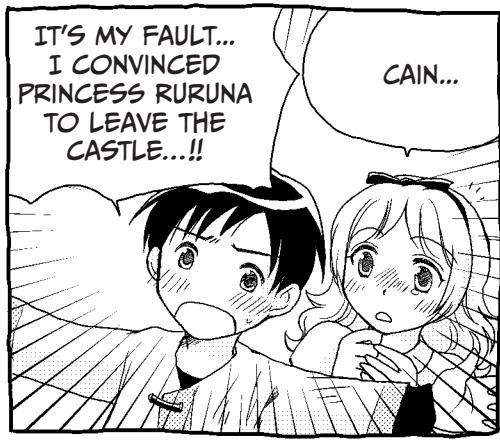
LET'S LEARN ABOUT SQL!

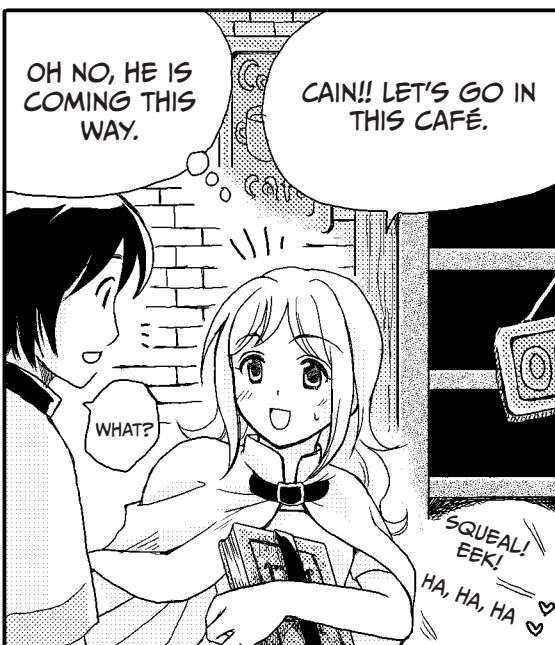
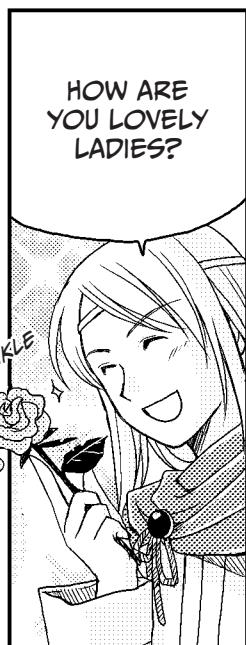
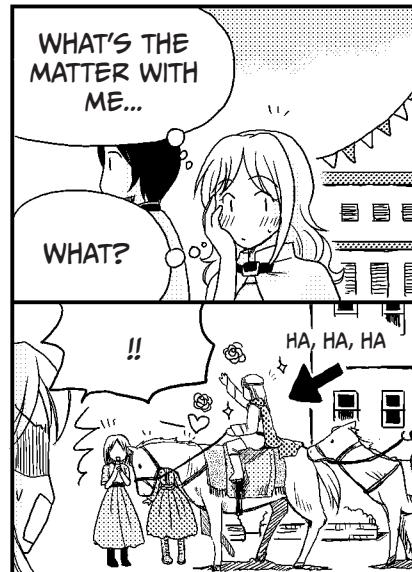


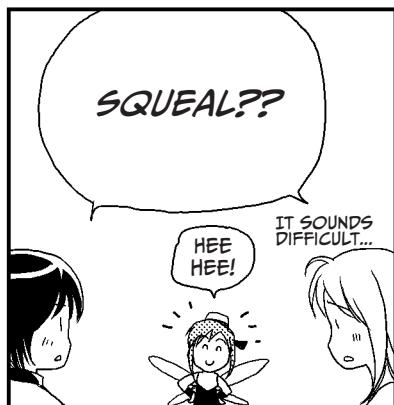
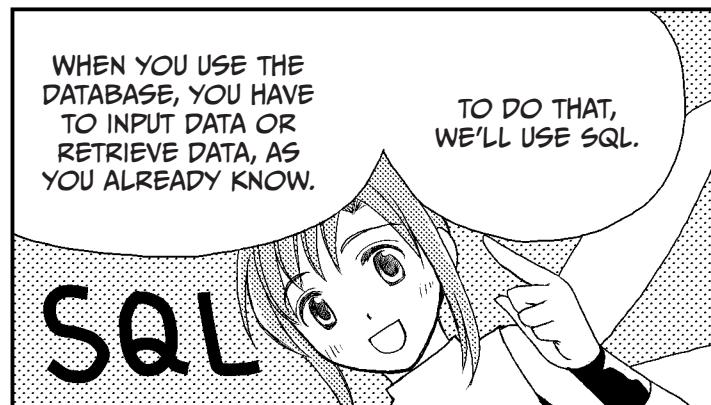
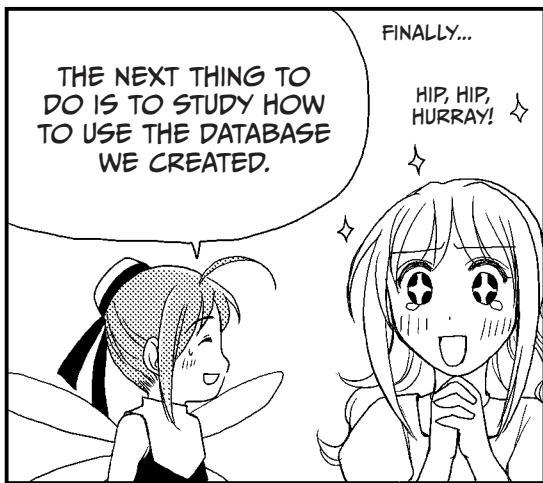
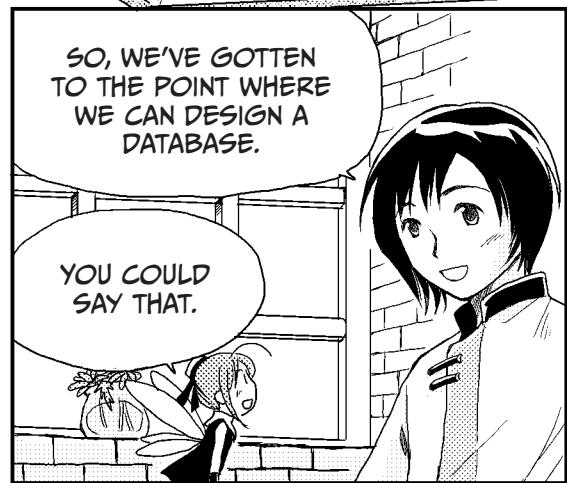
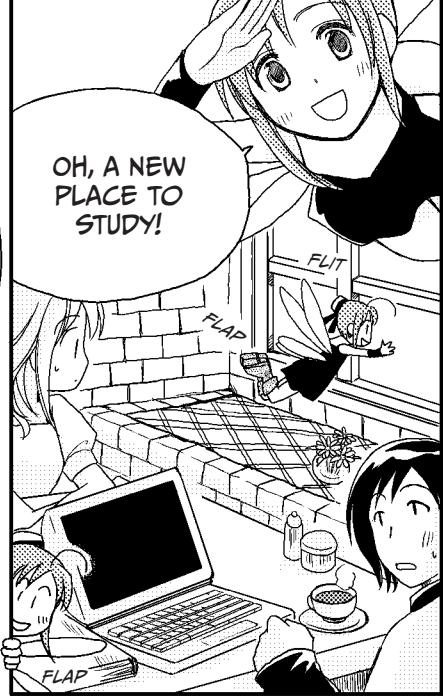
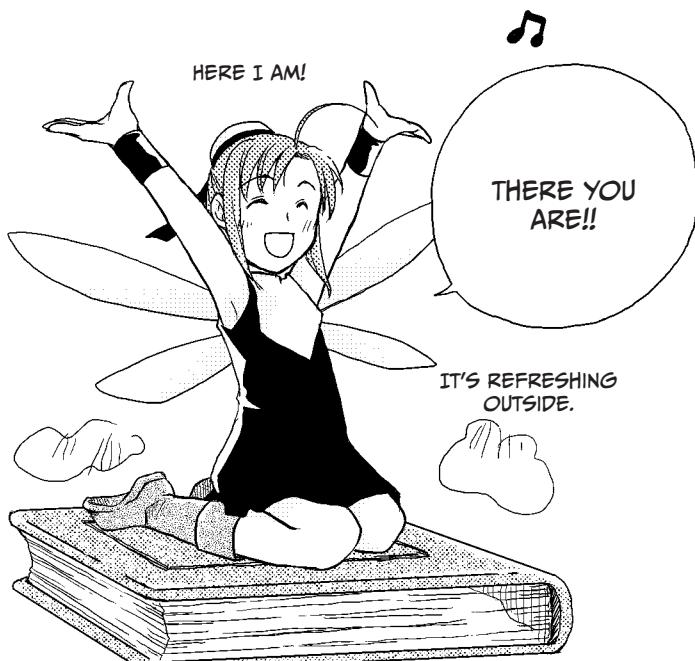
USING SQL

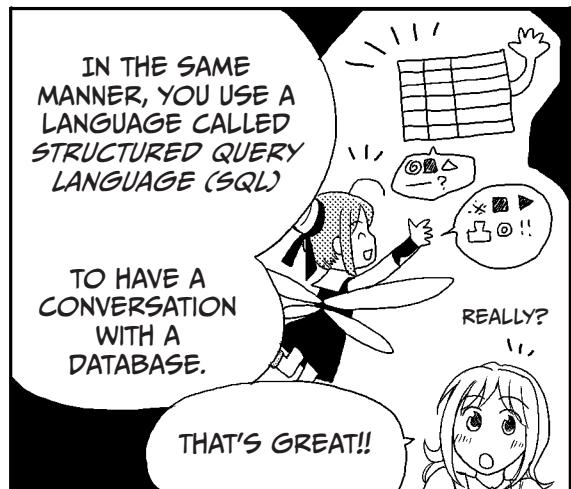
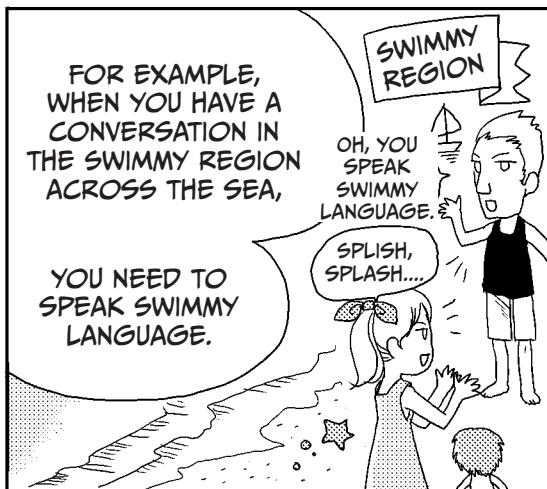




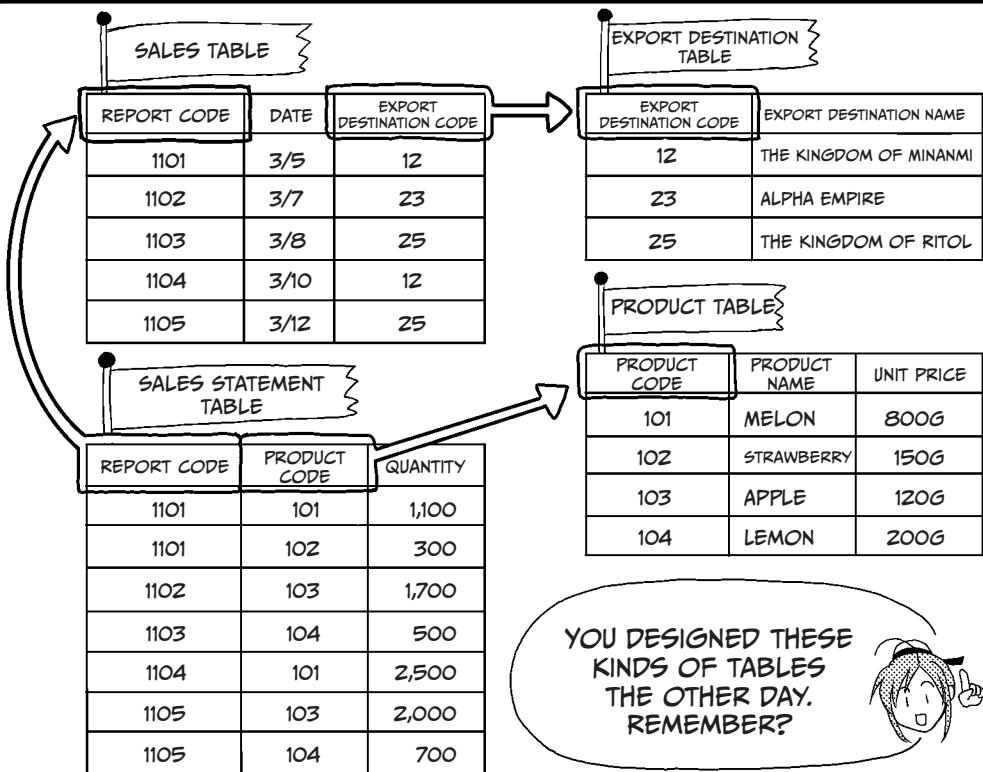


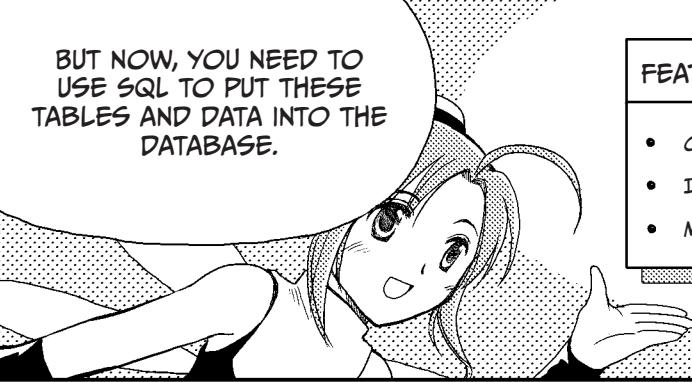






OF COURSE, I
KNOW THAT!





BUT NOW, YOU NEED TO USE SQL TO PUT THESE TABLES AND DATA INTO THE DATABASE.

FEATURES OF SQL

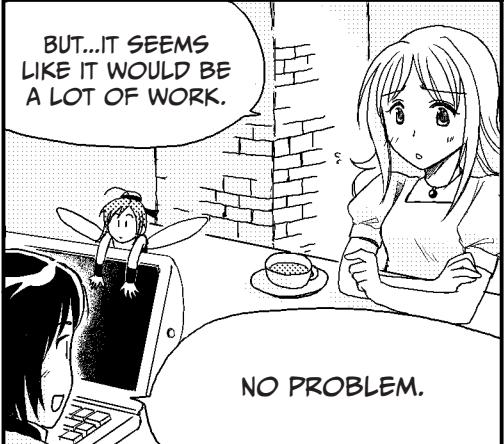
- CREATION OF TABLES
- INPUT AND RETRIEVAL OF DATA
- MANAGEMENT OF USERS



BY USING SQL, YOU CAN HAVE A CONVERSATION WITH THE DATABASE TO DO TASKS LIKE THESE...

IT SOUNDS LIKE WE CAN DO ALL SORTS OF THINGS!!

FEATURES
SQL
• CREATION OF TABLES
• INPUT AND RETRIEVAL
• MANAGEMENT OF USERS



BUT...IT SEEMS LIKE IT WOULD BE A LOT OF WORK.

NO PROBLEM.

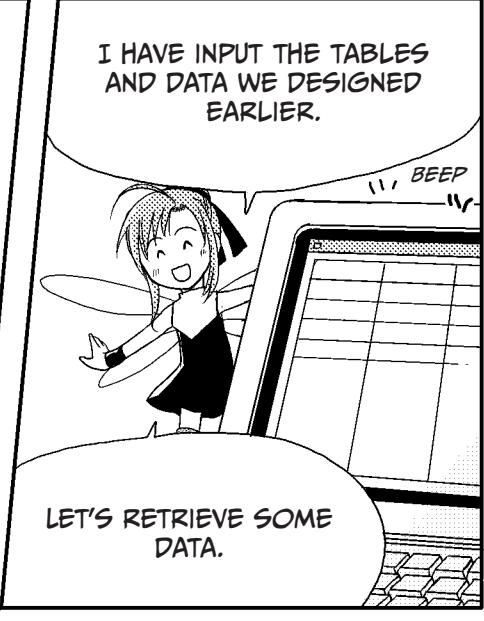


WE HAVE LEARNED SO MUCH ALREADY!



WELL...THAT'S TRUE... AND I WANT TO USE A DATABASE AS SOON AS POSSIBLE.

THAT'S THE SPIRIT!

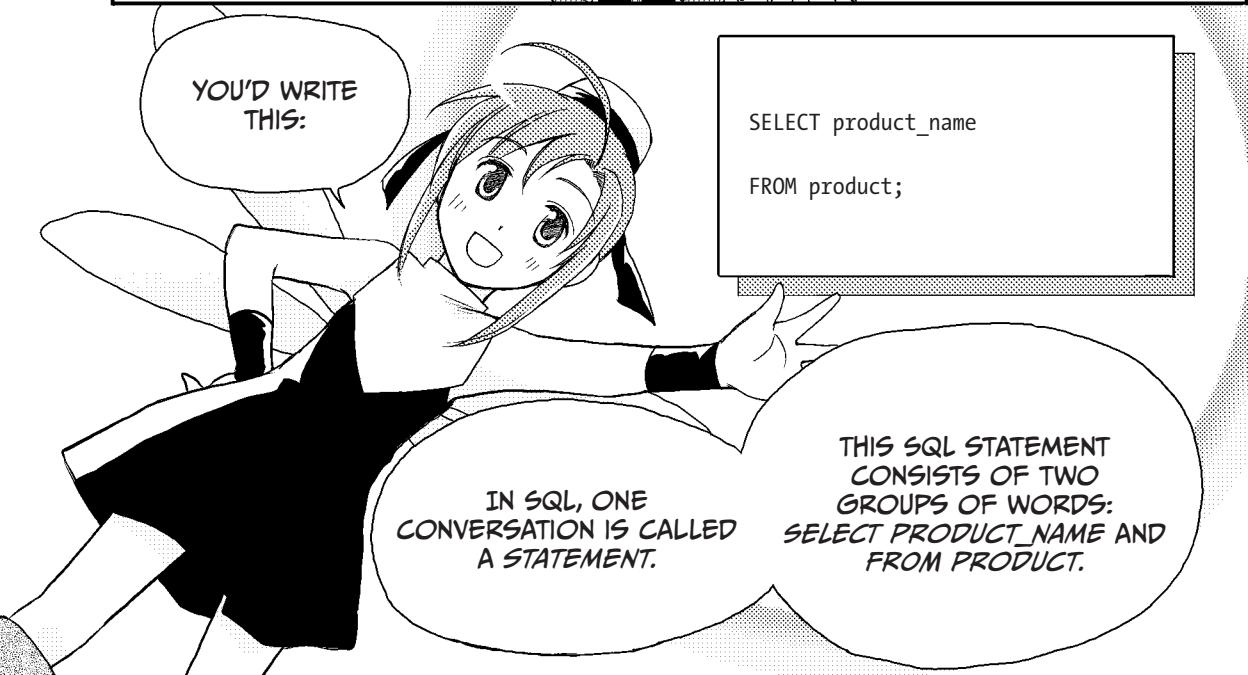
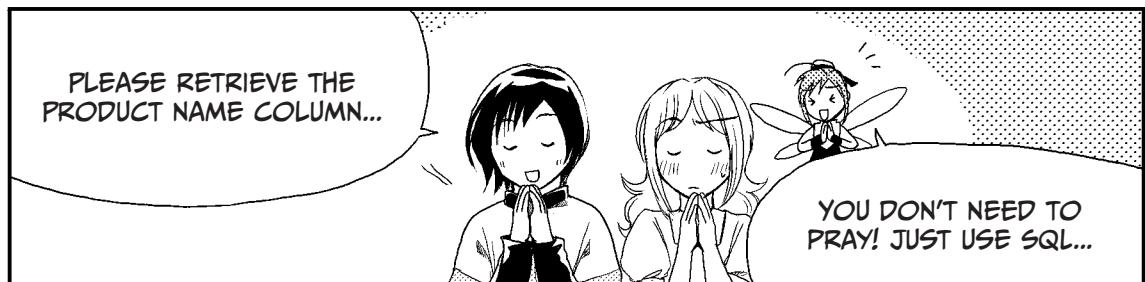
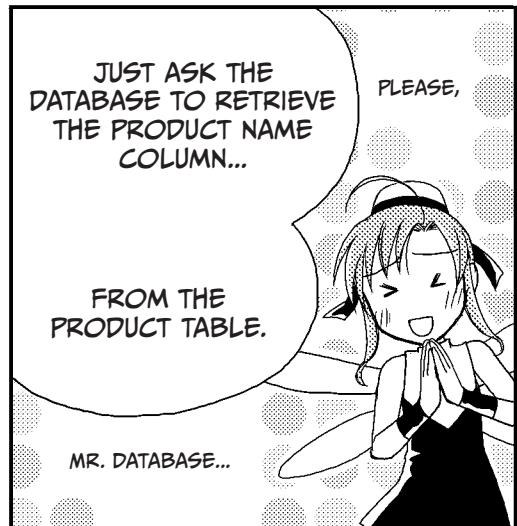


I HAVE INPUT THE TABLES AND DATA WE DESIGNED EARLIER.

„,BEEP“

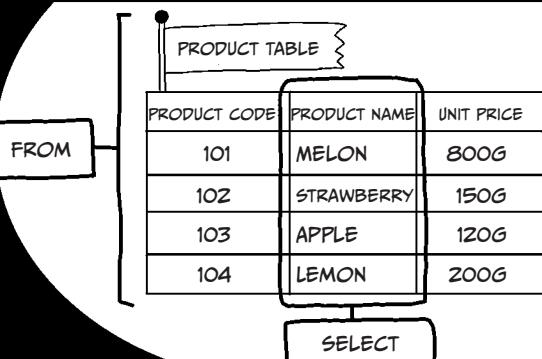
LET'S RETRIEVE SOME DATA.

SEARCHING FOR DATA USING A *SELECT* STATEMENT



THESE GROUPS OF WORDS ARE
CALLED PHRASES.

IN SQL, YOU SPECIFY A COLUMN NAME YOU WANT TO RETRIEVE WITH THE SELECT PHRASE AND THE TABLE NAME FROM WHICH YOU WANT TO RETRIEVE IT WITH THE FROM PHRASE.



HERE IS THE RETRIEVED
DATA.

THIS ALLOWS YOU TO
RETRIEVE ALL PRODUCT
NAMES FROM THE
PRODUCT TABLE.

PRODUCT NAME
MELON
STRAWBERRY
APPLE
LEMON

WE ARE HAVING A CONVERSATION WITH A DATABASE USING SQL.

THAT'S RIGHT.
YOU CAN RETRIEVE
NECESSARY
DATA BY USING
VARIOUS KINDS
OF PHRASES.

VARIOUS
KINDS...
HMM.

**WHAT ABOUT ASKING
FOR A LIST OF
PRODUCTS WHOSE
UNIT PRICE IS
GREATER THAN OR
EQUAL TO 200G?**

IN THAT CASE, YOU DON'T WANT ALL THE PRODUCT DATA.

YES, OF COURSE.

YOU ONLY NEED TO RETRIEVE PRODUCTS WHOSE UNIT PRICE IS GREATER THAN OR EQUAL TO 200G.

IN SUCH CASES, YOU SPECIFY CONDITIONS WITH THE WHERE PHRASE. FOR EXAMPLE,

WHERE unit_price>=200

YOU WRITE IT LIKE THIS.

I SEE... BUT...

IT IS INCONVENIENT TO SPECIFY A COLUMN NAME EACH TIME, ISN'T IT?

NO PROBLEM! TO SPECIFY ALL COLUMNS,

IT'S A PAIN!

HMM...

YOU CAN USE *!
IT CAN BE SUMMARIZED AS FOLLOWS.

BANG!!

```
SELECT *
FROM product
WHERE unit_price>=200
```

THIS STATEMENT RETRIEVES ALL THE DATA FROM THE PRODUCT TABLE...

HERE YOU ARE!

PRODUCTS THAT COST 200G OR MORE

PRODUCT CODE	PRODUCT NAME	UNIT PRICE
101	MELON	800G
104	LEMON	200G

SO,

THAT HAS A UNIT PRICE OF GREATER THAN OR EQUAL TO 200G.

SO, IF YOU CHANGE THE CONDITIONS, YOU CAN RETRIEVE PRODUCTS WHOSE UNIT PRICE IS BELOW 200G.

```
WHERE unit_price<200
```

THAT'S RIGHT—
LIKE THIS!

NOW WE NEED TO LEARN HOW TO MAKE CONDITIONS.

INDEED...

THEN, WHAT DO YOU DO TO RETRIEVE APPLE, FOR EXAMPLE?

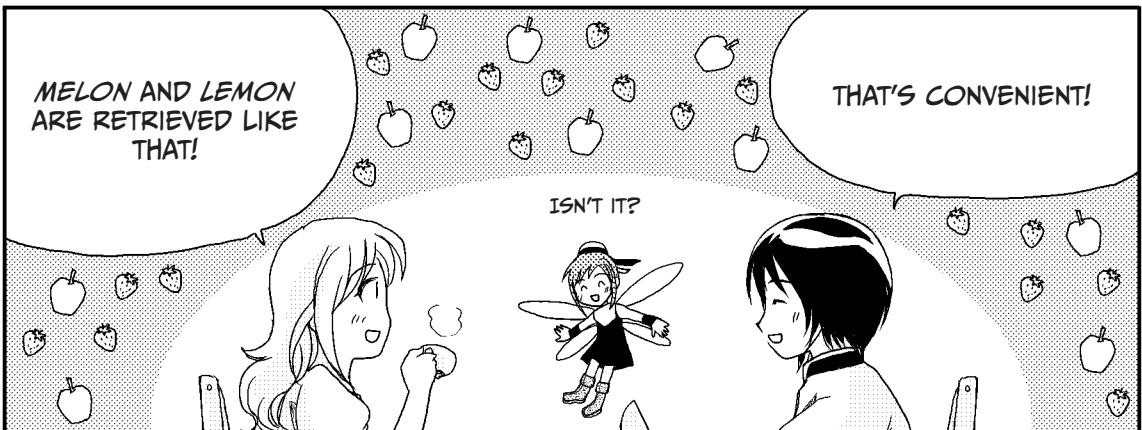
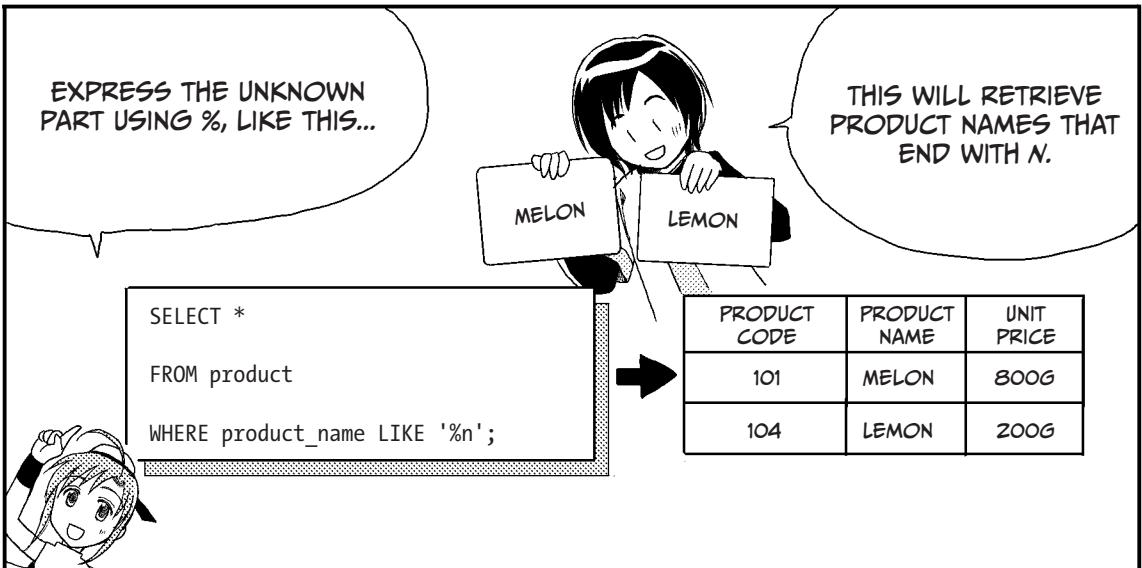
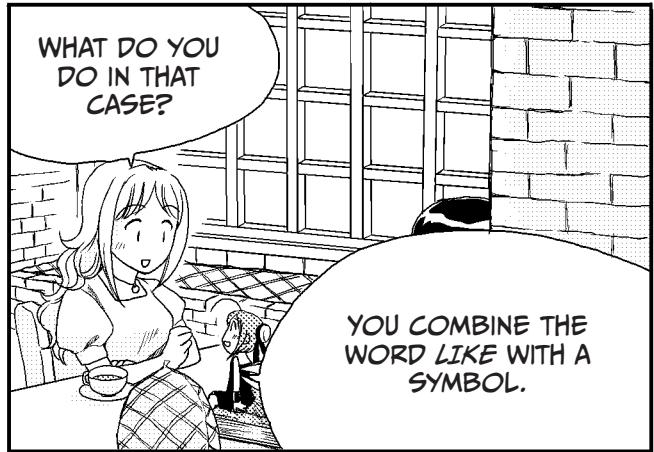
```
SELECT *  
FROM product  
WHERE product_name='apple';
```

WRITE IT LIKE THIS. WHEN USING CHARACTERS AS A CONDITION, ENCLOSE THEM WITHIN QUOTATION MARKS (').

PRODUCT CODE	PRODUCT NAME	UNIT PRICE
103	APPLE	120G

IF YOU DO THIS,
YOU CAN RETRIEVE
APPLE.

EXACTLY.



USING AGGREGATE FUNCTIONS

YOU CAN ALSO SORT RETRIEVED RESULTS WITH AN ORDER BY PHRASE.

TO SORT PRODUCTS IN ORDER OF ASCENDING PRICE, ADD A STATEMENT LIKE ORDER BY UNIT PRICE.

YOU CAN FIND OUT INFORMATION ABOUT PRODUCTS BY DOING THIS.



```
SELECT *  
FROM product  
WHERE unit_price < 200  
ORDER BY unit_price;
```

THAT'S GREAT!!

PRODUCT CODE	PRODUCT NAME	UNIT PRICE
103	APPLE	120G
102	STRAWBERRY	150G

I WANT TO KNOW MORE ABOUT SQL, TICO!

OH, REALLY?

I'M GLAD.

HOW ABOUT THIS ONE?

IN THE SELECT PHRASE, USE AVG (COLUMN NAME) TO OBTAIN THE AVERAGE OF EACH ROW.

```
SELECT AVG(unit_price)  
FROM product;
```

ばん!!

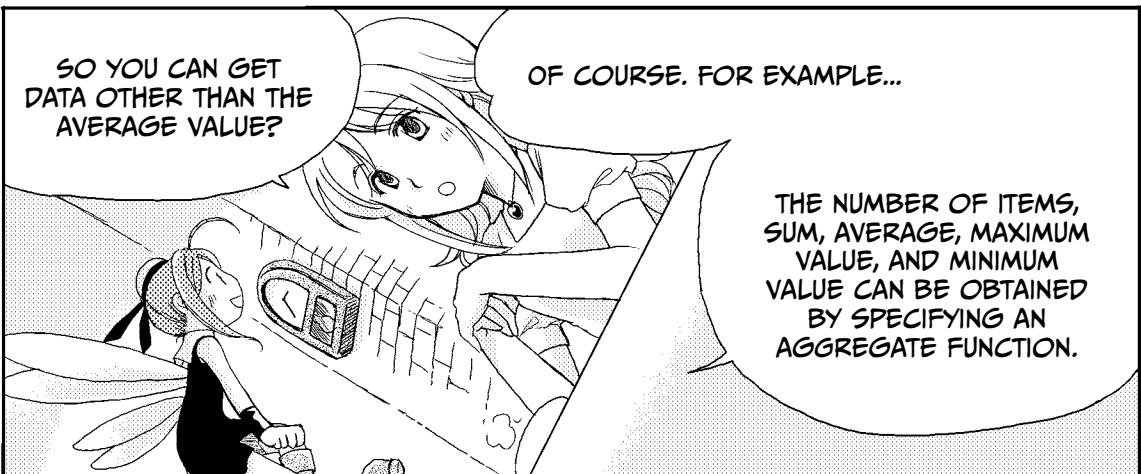
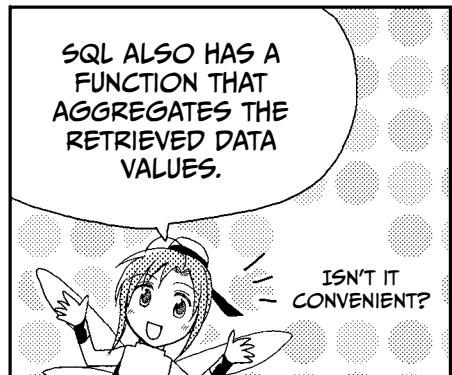
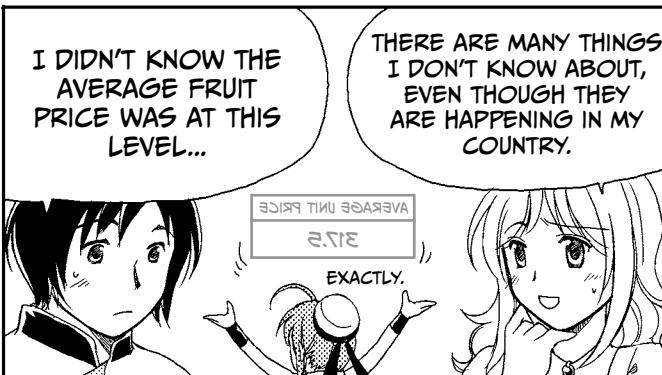
IT'S AMAZING.

ZZOOOP!!

AVERAGE UNIT PRICE

317.5

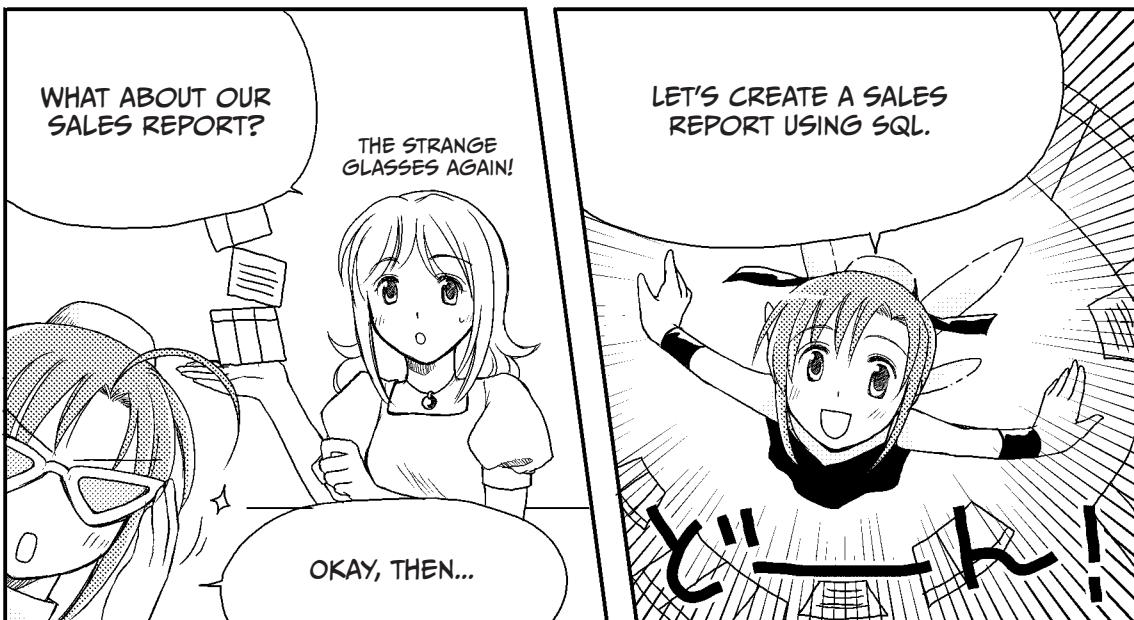
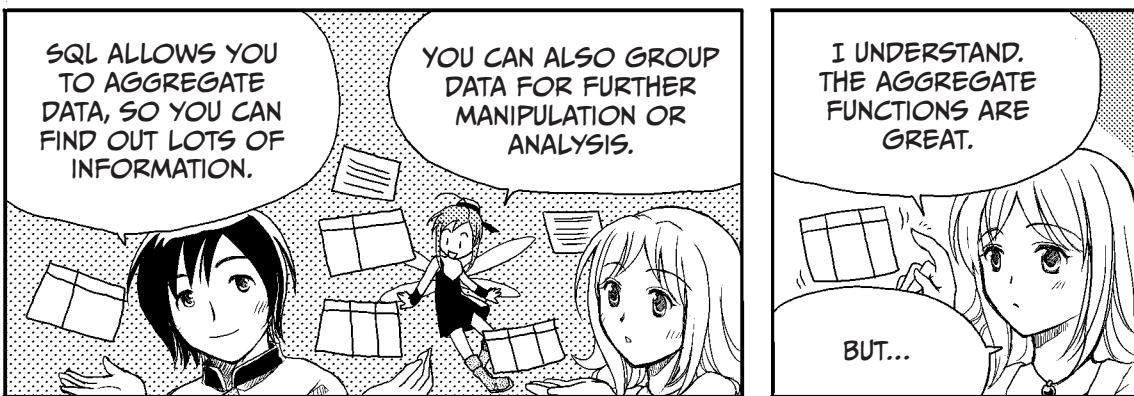
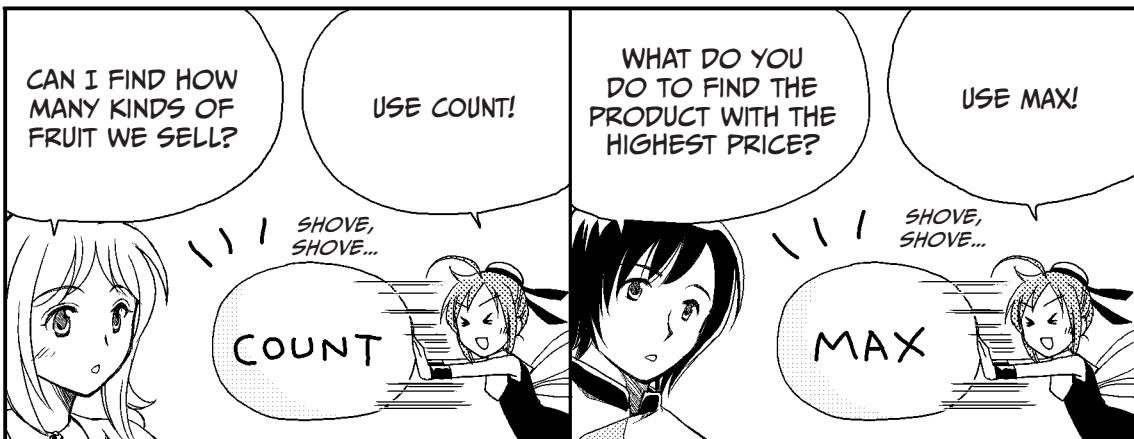
WE NOW HAVE THE AVERAGE UNIT PRICE OF PRODUCTS.



AGGREGATE FUNCTIONS IN SQL

Function	Description
COUNT(*)	Obtains the number of rows
COUNT(column_name)	Obtains the number of times the column is not null
COUNT(DISTINCT column_name)	Obtains the number of distinct values in the column
SUM(column_name)	Obtains the sum of the column's values in all rows
AVG(column_name)	Obtains the average of the column's values in all rows
MAX(column_name)	Obtains the maximum value of the column
MIN(column_name)	Obtains the minimum value of the column





JOINING TABLES

TO CREATE A SALES REPORT, YOU HAVE TO RETRIEVE DATA BY JOINING THE PRODUCT TABLE AND THE EXPORT DESTINATION, SALES, AND SALES STATEMENT TABLES.



RIGHT. THERE WAS JUST ONE TABLE BEFORE NORMALIZATION.

UH-OH!

IN ORDER TO JOIN TABLES, SQL REQUIRES A CONDITION THAT...

THE PRIMARY KEY IS EQUAL TO THE FOREIGN KEY WHICH REFERS TO THE PRIMARY KEY.

HOW CAN YOU SPECIFY THAT?

JOIN TABLES BY PLACING A COMMA BETWEEN THEM.

IF THE SAME COLUMN NAME APPEARS IN MULTIPLE TABLES, JUST SPECIFY IT AS TABLE_NAME.COLUMN_NAME.

```
SELECT sales.report_code, date, sales.export_destination_code,  
       export_destination_name, sales_statement.product_code,  
       product_name, unit_price, quantity  
  
FROM   sales, sales_statement, product, export_destination  
  
WHERE  sales.report_code = sales_statement.report_code  
       AND  
       sales_statement.product_code = product.product_code  
       AND  
       export_destination.export_destination_code =  
           sales.export_destination_code
```

HAVING JOINED THESE FOUR TABLES, WE THEN RESTRICT OUR RESULTS USING WHERE.



THIS WAY, YOU CAN RETRIEVE SALES REPORT DATA FROM TABLES, EVEN IF THEY ARE DIVIDED.

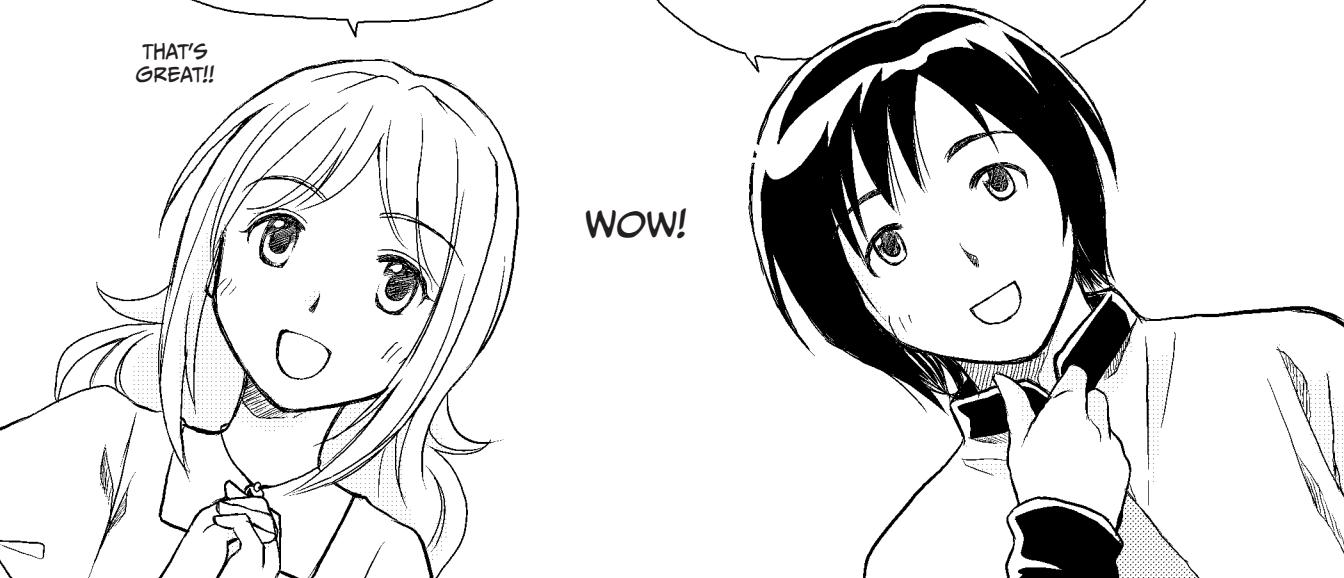
REPORT CODE	DATE	EXPORT DEST. CODE	EXPORT DEST. NAME	PRODUCT CODE	PRODUCT NAME	UNIT PRICE	QUANTITY
1101	3/5	12	THE KINGDOM OF MINANMI	101	MELON	800G	1,100
1101	3/5	12	THE KINGDOM OF MINANMI	102	STRAWBERRY	150G	300
1102	3/7	23	ALPHA EMPIRE	103	APPLE	120G	1,700
1103	3/8	25	THE KINGDOM OF RITOL	104	LEMON	200G	500
1104	3/10	12	THE KINGDOM OF MINANMI	101	MELON	800G	2,500
1105	3/12	25	THE KINGDOM OF RITOL	103	APPLE	120G	2,000
1105	3/12	25	THE KINGDOM OF RITOL	104	LEMON	200G	700

THIS IS THE SAME AS THE TABLE WE HAVE BEEN USING. WE RECREATED IT!

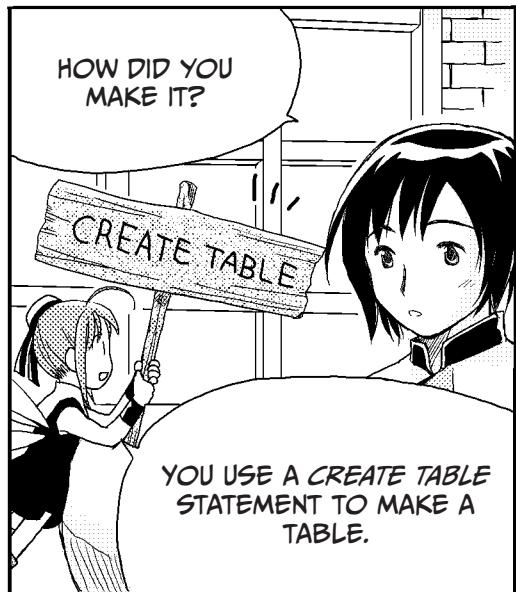
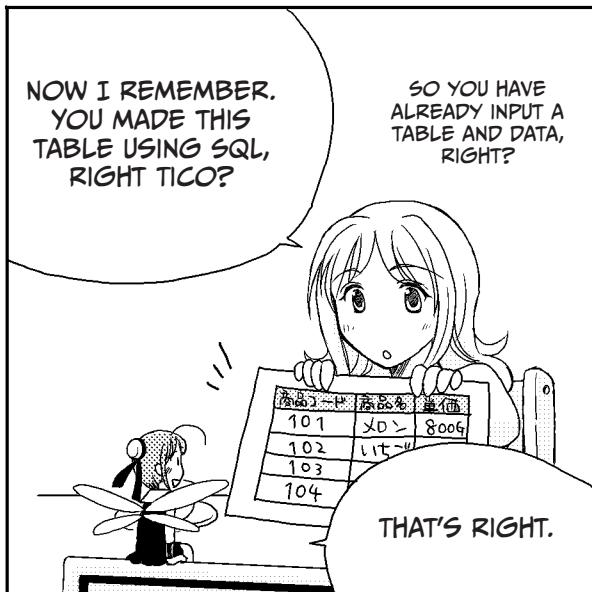
THAT'S GREAT!!

WOW!

YOU CAN RETRIEVE DATA RELATING TO THE SALES REPORT EVEN IF YOU MANAGE PRODUCTS, EXPORT DESTINATIONS, AND SALES INDEPENDENTLY.



CREATING A TABLE



CREATE TABLE product
(
product_code int NOT NULL,
product_name varchar(255),
unit_price int,
PRIMARY KEY(product_code)
);

PRODUCT CODE	PRODUCT NAME	UNIT PRICE

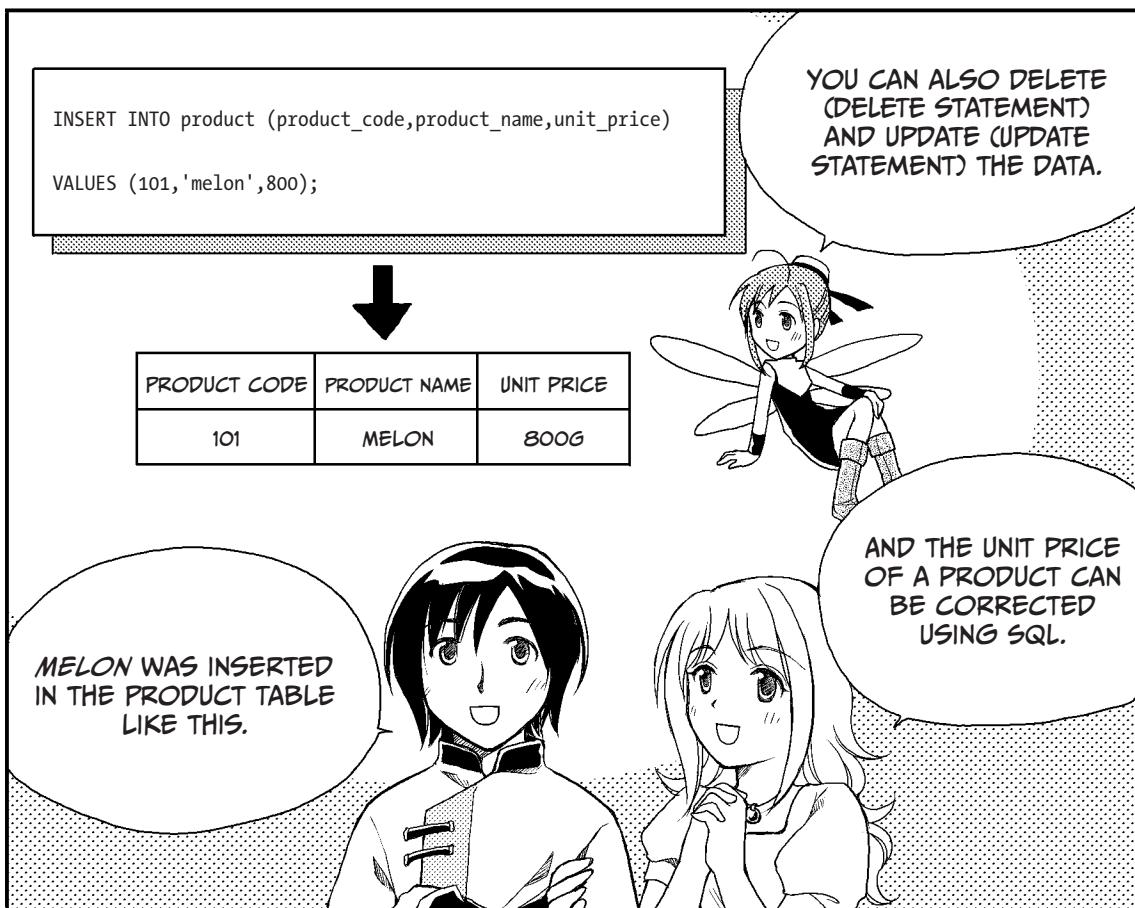
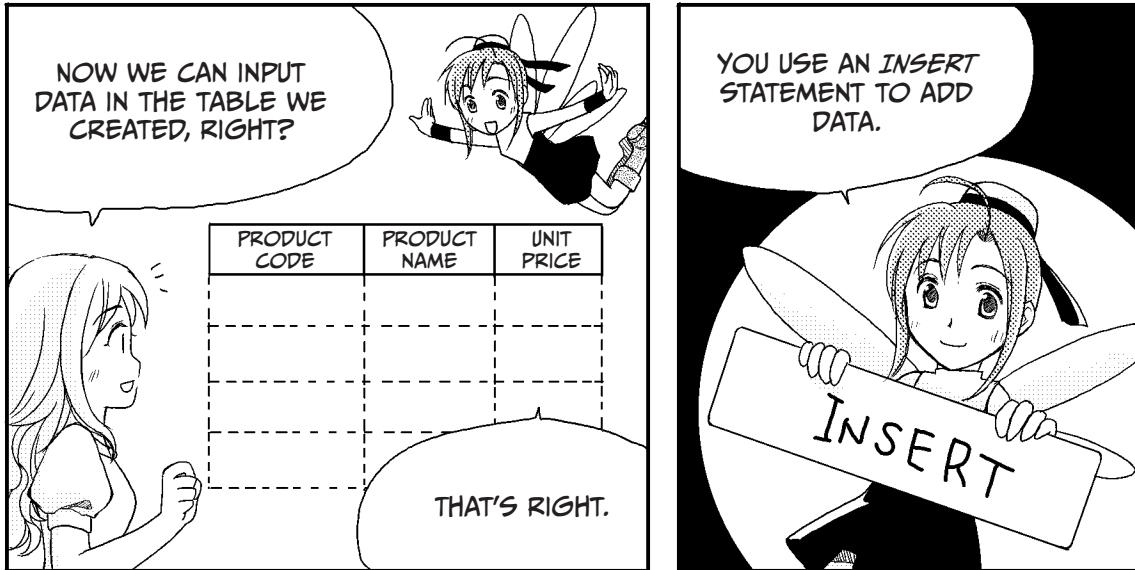
YOU MUST SPECIFY THE PRIMARY KEY, AS WELL.
I USED THE PRODUCT CODE AS A PRIMARY KEY.*

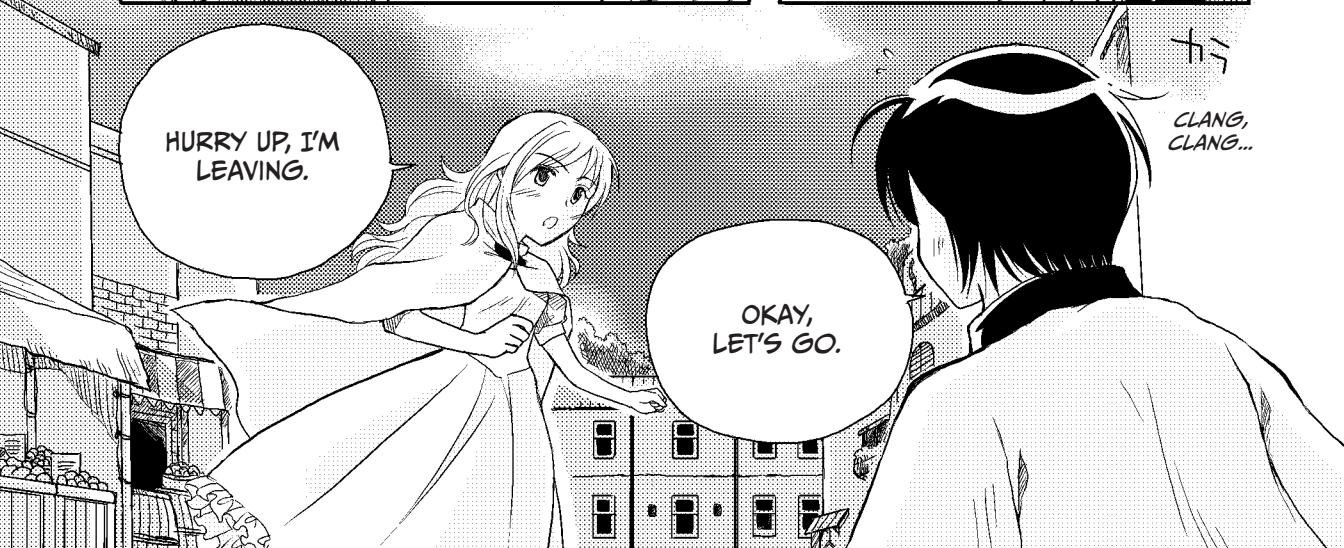
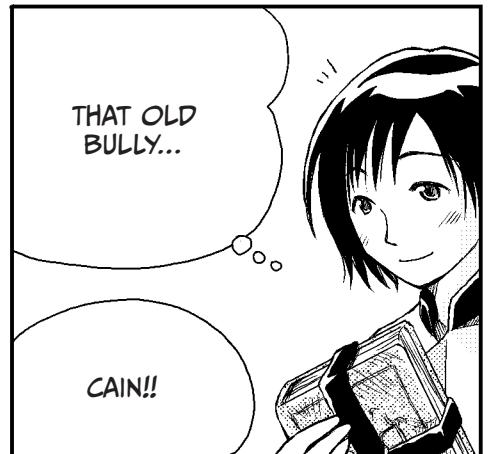
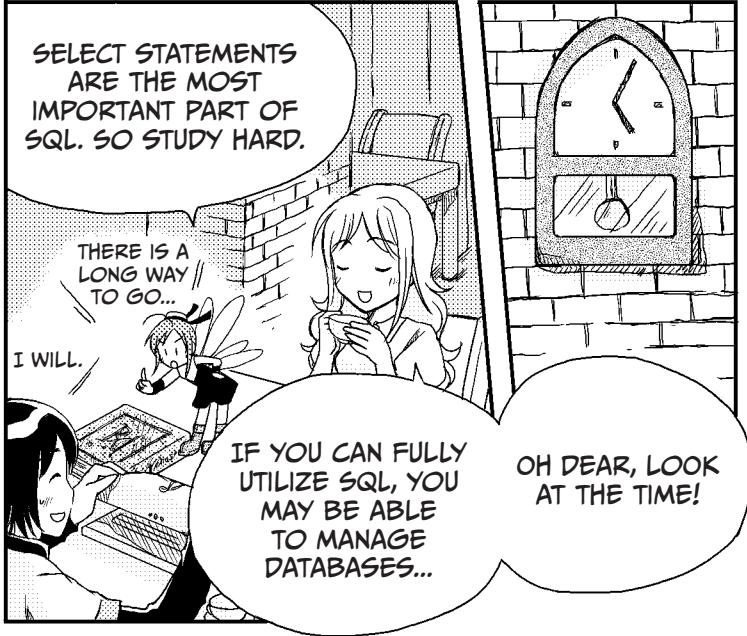
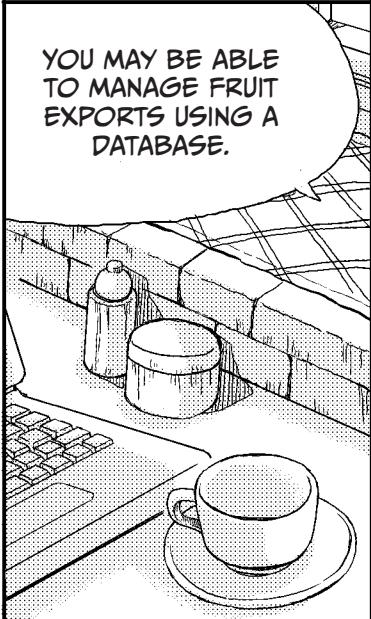
WE'VE ALSO SET THE DATATYPE OF EACH COLUMN. YOU CAN SEE THAT PRODUCT AND UNIT_PRICE ARE INTEGERS (INT). VARCHAR MEANS THAT THE DATABASE EXPECTS TEXT, AND (255) LIMITS THE PRODUCT_NAME TO 255 CHARACTERS.

LIKE THIS...

THIS PREVENTS YOU FROM ENTERING INCORRECT VALUES.

* SEE PAGE 115 FOR A COMPLETE EXPLANATION OF CREATE TABLE STATEMENTS.





SQL OVERVIEW



In this chapter, Princess Ruruna and Cain learned about *SQL*, or *Structured Query Language*, a language used to operate a relational database. SQL's commands can be broken down into three distinct types:

Data Definition Language (DDL) Creates a table

Data Manipulation Language (DML) Inputs and retrieves data

Data Control Language (DCL) Manages user access

SQL has commands that create the framework of a database, and a command that creates a table within a database. You can use this language to change and delete a table as well. The database language that has these functions is called the *Data Definition Language (DDL)*.

SQL also has commands that manipulate data in a database, such as inserting, deleting, and updating data. It also has a command that allows you to search for data. The database language with these functions is called the *Data Manipulation Language (DML)*.

In addition, SQL offers the capability to control a database, so that data conflicts will not occur even if multiple people use the database at the same time. The database language associated with these functions is called the *Data Control Language (DCL)*.

SEARCHING FOR DATA USING A SELECT STATEMENT

Princess Ruruna and Cain started learning SQL by using a basic data search function. SQL searches for data when one *statement* (a combination of phrases) is input. To search for a certain product with a unit price of 200G, for example, you would use the following SQL statement.

```
SELECT *
FROM product
WHERE unit_price=200
```

Create an SQL statement
by combining phrases.

A SELECT statement is the most basic SQL statement. It specifies *which column, from which table* (FROM), and *matching which conditions* (WHERE). You can combine these phrases to make intuitive, query-type statements in SQL—even a user unfamiliar with databases can use them to search for data.

CREATING CONDITIONS

Cain said earlier, “Now we need to learn how to make conditions.” Let’s look at some ways to create conditions using SQL.

COMPARISON OPERATORS

One way to express conditions is by using *comparison operators* like `>=` and `=`. For example, the condition “A is greater than or equal to B” is expressed using `>=`, and the condition “A is equal to B” is expressed using `=`. More examples of comparison operators are shown in the table below.

COMPARISON OPERATORS

Comparison operator	Description	Example	Description of example
<code>A = B</code>	A is equal to B.	<code>unit_price=200</code>	Unit price is 200G.
<code>A > B</code>	A is greater than B.	<code>unit_price>200</code>	Unit price is greater than 200G.
<code>A >= B</code>	A is greater than or equal to B.	<code>unit_price>=200</code>	Unit price is greater than or equal to 200G.
<code>A < B</code>	A is less than B.	<code>unit_price<200</code>	Unit price is less than 200G.
<code>A <= B</code>	A is less than or equal to B.	<code>unit_price<=200</code>	Unit price is less than or equal to 200G.
<code>A <> B</code>	A is not equal to B.	<code>unit_price<>200</code>	Unit price is not 200G.

LOGICAL OPERATORS

In some cases, you need to express conditions that are more complex than simple comparisons. You can use *logical operators* (`AND`, `OR`, and `NOT`) to combine operator-based conditions and create more complicated conditions, as shown in the table below.

LOGICAL OPERATORS

Logical operator	Description	Example	Description of example
<code>AND</code>	A and B	<code>Product code >= 200 AND unit price = 100</code>	The product code is greater than or equal to 200 and the unit price is 100G.
<code>OR</code>	A or B	<code>Product code >= 200 OR unit price = 100</code>	The product code is greater than or equal to 200 or the unit price is 100G.
<code>NOT</code>	Not A	<code>NOT unit price = 100</code>	The unit price is not 100G.

PATTERNS

When you don't know exactly what to search for, you can also use pattern matching in conditions by using wildcard characters. When using pattern matching, use characters such as % or _ in a LIKE statement; this will search for a character string that matches the pattern you specify. You can search for a value that corresponds to a partially specified character string using %, which indicates a character string of any length, and _, which specifies only one character.

An example of a query using wild cards is shown below. This example statement searches for a character string that has *n* at the end of the product name.

The diagram illustrates the use of wild cards in an SQL query. On the left, a code box contains the SQL statement:

```
SELECT *  
FROM product  
WHERE product_name LIKE '%n';
```

An arrow points from this statement to a callout box containing the text: "This statement matches patterns using a wild card." To the right of the arrow is a table titled "Product code" showing two rows of data:

Product code	Product name	Unit price
101	Melon	800G
104	Lemon	200G

The wild cards you can use in an SQL statement are explained below.

WILD CARDS

Wild card	Description	Example of pattern	Matching character string
%	Matches any number of characters	%n n%	Lemon Melon Nut Navel orange
_	Matches one character	_t t_	it to

SEARCHES

There are also many other search methods. For example, you can specify BETWEEN X AND Y for a value range. If you specify a range as shown below, you can extract products with unit prices greater than or equal to 150G or less than 200G.

The diagram shows a SQL query with a search condition using the BETWEEN operator:

```
SELECT *  
FROM product  
WHERE unit_price  
BETWEEN 150 AND 200;
```

A callout box to the right of the WHERE clause contains the text: "Specifies a search range".

In addition, you can specify IS NULL when searching for rows. If you use the search shown below, you can extract products with null unit prices.

The diagram shows a SQL query with a search condition using the IS NULL operator:

```
SELECT *  
FROM product  
WHERE unit_price is NULL;
```

A callout box to the right of the WHERE clause contains the text: "Searches for a null".



QUESTIONS

Now, let's create SQL statements using various kinds of conditions. Let's use the Export Destination Table below (assuming the unit for population is 10,000). Answer the questions below using SQL statements. The answers are on page 119.

EXPORT DESTINATION TABLE

Export destination code	Export destination name	Population
12	The Kingdom of Minanmi	100
23	Alpha Empire	120
25	The Kingdom of Ritol	150
30	The Kingdom of Sazanna	80

Q1

To find countries in which the population is greater than or equal to 1 million, extract the table below.

Export destination code	Export destination name	Population
12	The Kingdom of Minanmi	100
23	Alpha Empire	120
25	The Kingdom of Ritol	150

Q2

To find countries in which the population is less than 1 million, extract the table below.

Export destination code	Export destination name	Population
30	The Kingdom of Sazanna	80

Q3

Find countries in which the export destination code is less than 20 and the population is greater than or equal to 1 million.

Q4

Find countries in which the export destination code is greater than or equal to 30 and the population is greater than 1 million.

Q5

What is the population of the Kingdom of Ritol?

Q6

Find countries whose names contain the letter *n*.

AGGREGATE FUNCTIONS



Princess Ruruna and Cain have learned about various aggregate functions. Aggregate functions are also known as *set functions*. You can use these functions to aggregate information such as maximum and minimum values, number of items, and sum.

If you specify a WHERE phrase along with an aggregate function, you can obtain an aggregated value for just the specified rows. If you specify a phrase like the one shown below, you can figure out the number of products with unit prices greater than or equal to 200G.

```
SELECT COUNT(*)  
FROM product  
WHERE unit_price>=200;
```



COUNT(*)
2

AGGREGATING DATA BY GROUPING

If you group data, you can obtain aggregated values easily. For example, if you want to obtain the number of products and average unit price based on district, you can use the grouping function.

To group data, combine the aggregate function and the GROUP BY phrase. Let's use the Product Table shown below.

PRODUCT TABLE

Product code	Product name	Unit price	District
101	Melon	800G	South Sea
102	Strawberry	150G	Middle
103	Apple	120G	North Sea
104	Lemon	200G	South Sea
201	Chestnut	100G	North Sea
202	Persimmon	160G	Middle
301	Peach	130G	South Sea
302	Kiwi	200G	South Sea

To obtain the average unit price for each district in the Product Table, specify the District column and the AVG function for the GROUP BY phrase. This will group data based on district and give you the average unit value of the products in each district.

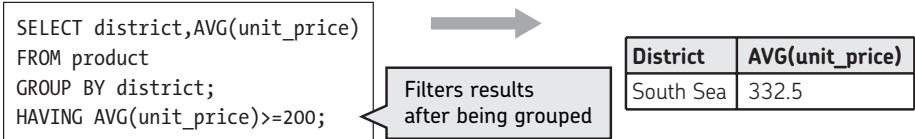
```
SELECT district,AVG(unit_price)  
FROM product  
GROUP BY district;
```

Enables grouping



District	AVG(unit_price)
South Sea	332.5
North Sea	110
Middle	155

What if you wanted to further restrict your results, based on a particular property of the data? Assume that you want to find products with regional average unit prices greater than or equal to 200G. In this case, do not specify a condition in the WHERE phrase, but use a HAVING phrase instead. This allows you to extract only districts in which the average unit price is greater than or equal to 200G.



QUESTIONS



Answer the questions below using this Export Destination Table (assuming the unit for population is 10,000). The answers are on page 120.

EXPORT DESTINATION TABLE

Export destination code	Export destination name	Population	District
12	The Kingdom of Minamni	100	South Sea
15	The Kingdom of Paronu	200	Middle
22	The Kingdom of Tokanta	160	North Sea
23	Alpha Empire	120	North Sea
25	The Kingdom of Ritol	150	South Sea
30	The Kingdom of Sazanna	80	South Sea
31	The Kingdom of Taharu	240	North Sea
33	The Kingdom of Mariyon	300	Middle

Q7

What is the smallest population?

Q8

What is the largest population?

Q9

What is the total population of all countries included in the Export Destination Table?

Q10

What is the total population of the countries in which the export destination code is greater than 20?

Q11

How many countries are there in which the population is greater than or equal to 1 million?

Q12

How many countries are in the North Sea district?

Q13

Which country in the North Sea district has the largest population?

Q14

What is the total population of every country excluding the Kingdom of Ritol?

Q15

Find the districts in which the average population is greater than or equal to 2 million.

Q16

Find the districts that contain at least three countries.

SEARCHING FOR DATA



There are more complicated query methods available in SQL, in addition to the ones we've already discussed.

USING A SUBQUERY

For example, you can embed one query in another query. This is called a *subquery*. Let's look at the tables below.

PRODUCT TABLE

Product code	Product name	Unit price
101	Melon	800G
102	Strawberry	150G
103	Apple	120G
104	Lemon	200G

SALES STATEMENT TABLE

Report code	Product code	Quantity
1101	101	1,100
1101	102	300
1102	103	1,700
1103	104	500
1104	101	2,500
1105	103	2,000
1105	104	700

You can use these two tables to search for the names of products for which the sales volume is greater than or equal to 1,000. The following SQL statement will conduct that search.

```
SELECT * FROM product
WHERE product_code IN
(SELECT product_code
FROM sales_statement
WHERE quantity>=1000);
```

This statement contains a subquery.

In this SQL statement, the SELECT statement in parentheses is performed first: The product code in the Sales Statement Table is searched for first, and product codes 101 and 103 are found (as these are the only reports with sales volume greater than 1,000). These product codes are used as a part of the condition for the SELECT statement outside the parentheses. For IN, the condition is satisfied when a row matches any value enclosed within parentheses. Thus, products that correspond to the product codes 101 and 103 will be returned.

In other words, in the case of a subquery, the result of the SELECT statement within parentheses will be sent to the other SELECT statement for searching. The following information will be the result of the whole query.

Product code	Product name	Unit price
101	Melon	800G
103	Apple	120G

USING A CORRELATED SUBQUERY

Let's consider a subquery as being *contained inside* another query. Such a subquery may refer to data from the outer query. This is called a *correlated subquery*. In the query below, the *sales_statement* table in the outer query is temporarily given the new name U so the subquery can refer to it unambiguously. The syntax U.product_code indicates which product_code column is intended, since there are two sources for that column inside the subquery.

Because the subquery refers to data from the outer query, the subquery is not independent of the outer query as in previous examples. This dependency is called a *correlation*.

```

① SELECT *
FROM sales_statement U
② WHERE quantity>
③ (SELECT AVG(quantity)
FROM sales_statement
WHERE product_code=U.product_code);

```

Report code	Product code	Quantity
1104	101	2,500
1105	103	2,000
1105	104	700

This query extracts statements with sales volume greater than the product's average.

Let's look at how this correlated subquery is processed. In the correlated subquery, the query outside is implemented first.

```

① SELECT *
FROM sales_statement U

```

This result is sent to the query inside to be evaluated row by row. Let's explore the evaluation of the first row, product code 101.

③ (SELECT AVG(quantity)
FROM sales_statement
WHERE product_code=101)

The product code for the first row is 101, or melons—the average sales quantity of melons is 1,800. This result is then sent as a condition for the query outside.

② WHERE quantity>(1,800)

This process continues for all rows in the sales statement—steps ② and ③ are performed for all possible product codes. In other words, this query extracts reports in which the sales volume of a fruit is greater than that particular fruit's average sales quantity. Consequently, only the fifth, sixth, and seventh rows of ① are extracted.



QUESTIONS

Now, answer the following questions based on the Product Table and the Sales Statement Table. The answers are on page 122.

Q17

Find the sales statement for fruit with unit prices greater than or equal to 300G, and extract the table below.

Report code	Product code	Quantity
1101	101	1,100
1104	101	2,500

Q18

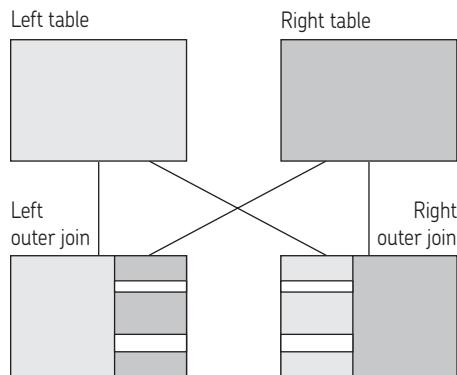
Obtain the average sales volume by product, and find items that have sales volumes that are less than the average.

JOINING TABLES

After conducting an SQL-based search, Princess Ruruna and Cain created a sales report by combining tables. Joining tables by combining columns with the same names is called an *equi join*. For an equi join, rows with the same value are designated as join conditions for joining tables. Joining columns with the same name into one is called a *natural join*.

The join method in which only rows having a common value like equi join are selected is called *inner join*.

In contrast, the join method that keeps all rows of one table and specifies a null for rows not included in another table is called an *outer join*. If you place a table created from an outer join on the right or left of an SQL statement, it is called a *left outer join* or a *right outer join*, depending on which rows are kept.



CREATING a TABLE



Finally, Princess Ruruna and Cain learned about the statement syntax that creates a table, CREATE TABLE. The statement syntax inside a CREATE TABLE statement often depends on the particular kind of database you use. An example is shown below.

```
CREATE TABLE product
(
product_code int NOT NULL,
product_name varchar(255),
unit_price int,
PRIMARY KEY(product_code)
);
```

This statement creates a table.

When you create a table, you must specify its column names. Additionally, you can specify a primary key and a foreign key for each column. In this example, the product code is specified as a PRIMARY KEY and product code is not allowed to be null. When creating a table, you may need to include the following specifications.

CONSTRAINTS ON A TABLE

Constraint	Description
PRIMARY KEY	Sets a primary key
UNIQUE	Should be unique
NOT NULL	Does not accept a NULL value
CHECK	Checks a range
DEFAULT	Sets a default value
FOREIGN KEY REFERENCES	Sets a foreign key

These specifications are called *constraints*. Giving constraints when creating a table helps to prevent data conflicts later on and allows you to correctly manage the database.

INSERTING, UPDATING, OR DELETING ROWS

You can use the INSERT, UPDATE, and DELETE statements to insert, update, or delete data from a table created by the CREATE TABLE statement. Let's insert, update, and delete some data using SQL.

```
INSERT INTO product  
(product_code,product_name,unit_price)  
VALUES (200,'cherry',200);
```

This statement adds cherry.

```
UPDATE product  
SET product_name='cantaloupe'  
WHERE product_name='melon';
```

This statement updates
melon to cantaloupe.

```
DELETE FROM product  
WHERE product_name='apple';
```

This statement deletes apple.

Product code	Product name	Unit price
101	Cantaloupe	800G
102	Strawberry	150G
103	Apple	120G
104	Lemon	200G
200	Cherry	200G

Updated to cantaloupe.

Apple is deleted.

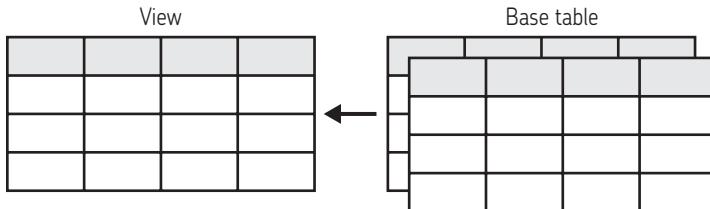
Cherry is added.

When inserting, updating, or deleting a row, you cannot violate the constraints set by the CREATE TABLE statement. If a product with product code 200 already exists, you cannot add cherry, since you cannot add duplicated data as a primary key. When you insert, update, or delete data in a database, you must consider the database's constraints.



CREATING A VIEW

Based on the table you created with the CREATE TABLE statement, you can also create a virtual table that exists only when it is viewed by a user. This is called a *view*. The table from which a view is derived is called a *base table*.



Use the SQL statement shown below to create a view.

```
CREATE VIEW expensive_product
(product_code,product_name,unit_price)
AS SELECT *
FROM product
WHERE unit_price>=200;
```

This statement creates a view.

The Expensive Product Table is a view based on the Product Table, which is a base table. It was created by extracting data with unit prices greater than or equal to 200G from the Product Table.

EXPENSIVE PRODUCT TABLE

Product code	Product name	Unit price
101	Melon	800G
104	Lemon	200G
202	Persimmon	200G

Once you create the Expensive Product view, you can search for data in it the same way you would search for data in a base table.

```
SELECT *
FROM expensive_product
WHERE unit_price>=500;
```

Allows the view to be used
in the same manner as a
base table

It is convenient to create a view when you want to make part of the data in a base table public.

There are also SQL statements for deleting a base table or view. The statement used to delete a base table or view is shown below.

```
DROP VIEW expensive_product;
```

```
DROP TABLE product;
```



QUESTIONS

Create SQL statements for the following questions (assuming the unit for population is 10,000). The answers are on page 123.

Q19

The following Export Destination Table was created using a CREATE TABLE statement.
Add the data below.

EXPORT DESTINATION TABLE

Export destination code	Export destination name	Population	District
12	The Kingdom of Minanmi	100	South Sea
15	The Kingdom of Paronu	200	Middle
22	The Kingdom of Tokanta	160	North Sea
23	Alpha Empire	120	North Sea

Q20

From the Export Destination Table in Q19, create a view titled *North Sea Country* that shows countries belonging to the North Sea district.

EXPORT DESTINATION TABLE

Export destination code	Export destination name	Population
22	The Kingdom of Tokanta	160
23	Alpha Empire	120

Q21

Change the population of the Kingdom of Tokanta in the Export Destination Table to 1.5 million.

Q22

In the Export Destination Table, delete all data for the Kingdom of Paronu.

SUMMARY



You can use SQL functions to define, operate, and control data.
To search for data, use a SELECT statement.
To specify a condition, use a WHERE phrase.
To insert, update, and delete data, use INSERT, UPDATE, and DELETE statements.
To create a table, use a CREATE TABLE statement.

ANSWERS

Q1

```
SELECT *
FROM export_destination
WHERE population>=100;
```

Q2

```
SELECT *
FROM export_destination
WHERE population<100;
```

Q3

```
SELECT *
FROM export_destination
WHERE export_destination_code<20
AND population>=100;
```

Export destination code	Export destination name	Population
12	The Kingdom of Minanmi	100

Q4

```
SELECT *
FROM export_destination
WHERE export_destination_code>=30
AND population>100;
```

None of the countries meet this criteria, so this query returns an empty set.

Q5

```
SELECT population
FROM export_destination
WHERE export_destination_name='the Kingdom of Ritol';
```

Population
150

Q6

```
SELECT *
FROM export_destination
WHERE export_destination_name LIKE '%n%';
```

Export destination code	Export destination name	Population
12	The Kingdom of Minanmi	100
25	The Kingdom of Ritol	150
30	The Kingdom of Sazanna	80

Q7

```
SELECT MIN(population)
FROM export_destination;
```

MIN(population)
80

Q8

```
SELECT MAX(population)
FROM export_destination;
```

MAX(population)
300

Q9

```
SELECT SUM(population)
FROM export_destination;
```

SUM(population)
1,350

Q10

```
SELECT SUM(population)
FROM export_destination
WHERE export_destination_code>20;
```

SUM(population)
1,050

Q11

```
SELECT COUNT(*)
FROM export_destination
WHERE population>=100;
```

COUNT(*)
7

Q12

```
SELECT COUNT(*)
FROM export_destination
WHERE district='north sea';
```

COUNT(*)
3

Q13

```
SELECT MAX(population)
FROM export_destination
WHERE district='north sea';
```

MAX(population)
240

Q14

```
SELECT SUM(population)
FROM export_destination
WHERE NOT(export_destination_name='the Kingdom of Ritol');
```

SUM(population)
1,200

Q15

```
SELECT district, AVG(population)
FROM export_destination
GROUP BY district
HAVING AVG(population)>=200;
```

District	AVG(population)
Middle	250

Q16

```
SELECT district, COUNT(*)
FROM export_destination
GROUP BY district
HAVING COUNT(*)>=3;
```

District	COUNT(*)
North Sea	3
South Sea	3

Q17

```
SELECT *
FROM sales_statement
WHERE product_code IN
(SELECT product_code
FROM product
WHERE unit_price>=300);
```

Q18

```
SELECT *
FROM sales_statement U
WHERE quantity<
(SELECT AVG(quantity)
FROM sales_statement
WHERE product_code=U.product_code);
```

Report code	Product code	Quantity
1101	101	1,100
1102	103	1,700
1103	104	500

Q19

```
INSERT INTO export_destination(export_destination_
code,export_destination_name,population,district)
VALUES(12,'the Kingdom of Minanmi',100,'south sea');
INSERT INTO export_destination(export_destination_
code,export_destination_name,population,district)
VALUES(15,'the Kingdom of Paronu',200,'middle');
INSERT INTO export_destination(export_destination_
code,export_destination_name,population,district)
VALUES(22,'the Kingdom of Tokanta',160,'north sea');
INSERT INTO export_destination(export_destination_
code,export_destination_name,population,district)
VALUES(23,'Alpha Empire',120,'north sea');
```

Q20

```
CREATE VIEW north_sea_country(export_destination_
code,export_destination_name,population)
AS SELECT export_destination_code,export_destination_name,population
FROM export_destination_name
WHERE district='north sea';
```

Q21

```
UPDATE export_destination
SET population=150
WHERE export_destination_name='the Kingdom of Tokanta';
```

Q22

```
DELETE FROM export_destination
WHERE export_destination_name='the Kingdom of Paronu';
```

STANDARDIZATION OF SQL

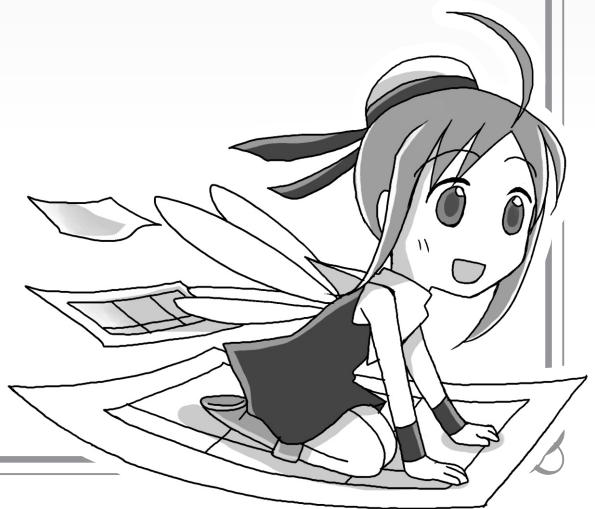
SQL is standardized by the International Organization for Standardization (ISO). In Japan, it is standardized by JIS (Japanese Industrial Standards).

Other SQL standards include SQL92, established in 1992, and SQL99, established in 1999. Relational database products are designed so that queries can be made in accordance with these standards.

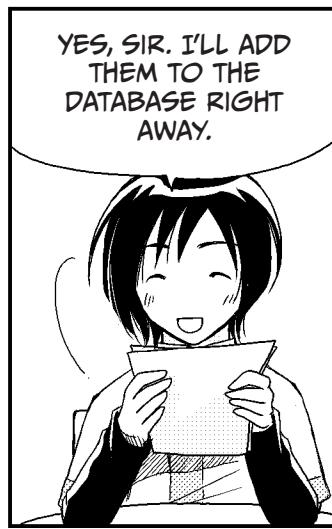
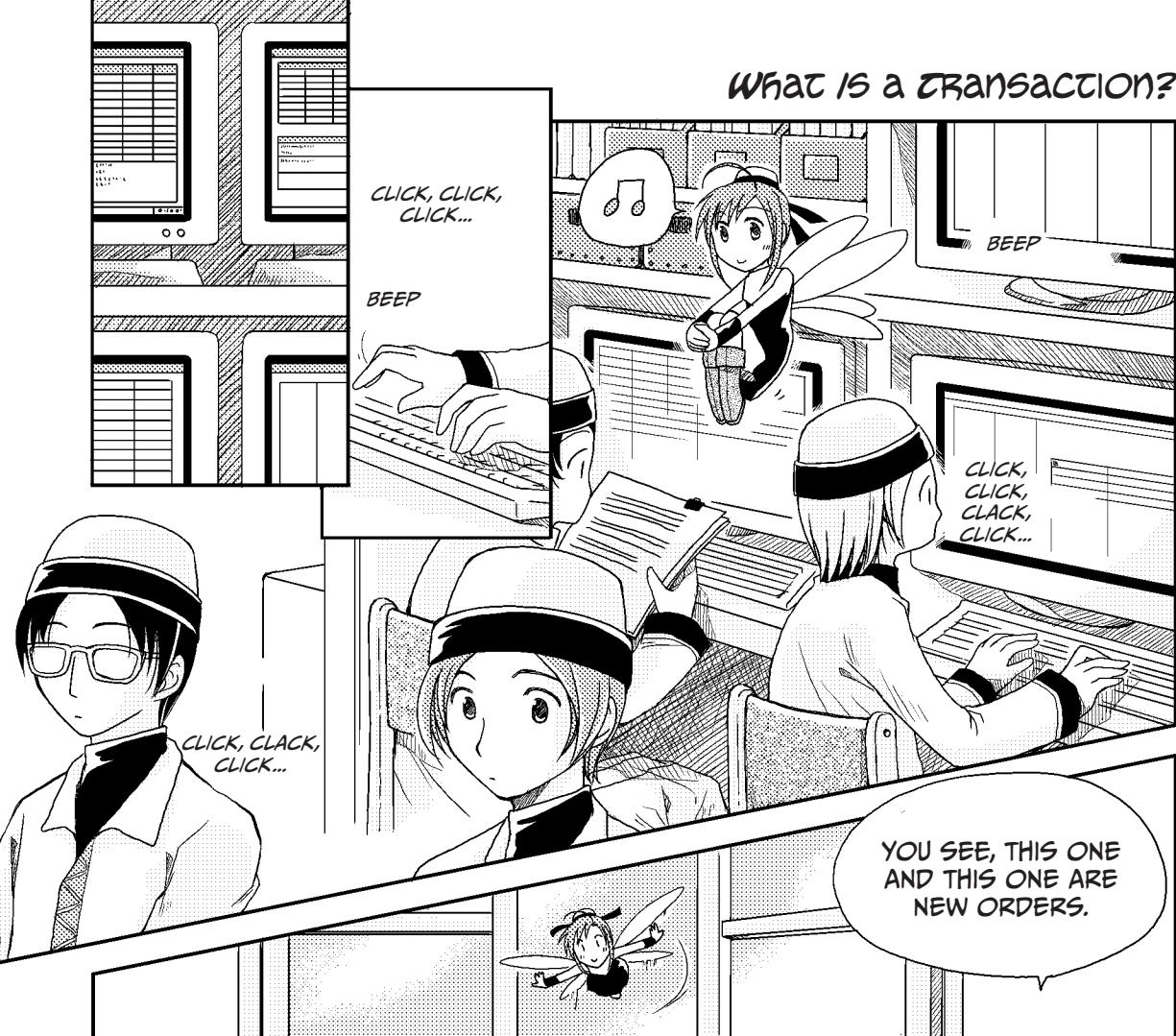
Some relational database products have their own specifications. Refer to the operation manual for your database product for further information.

5

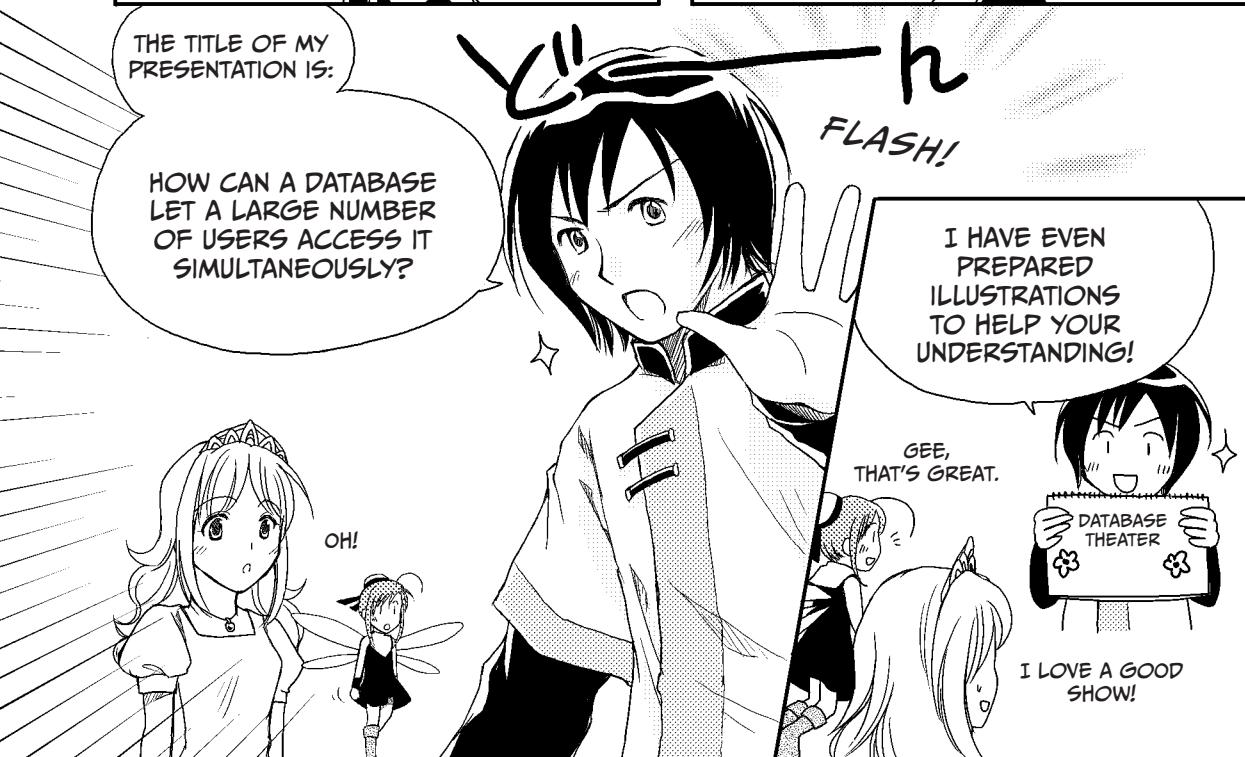
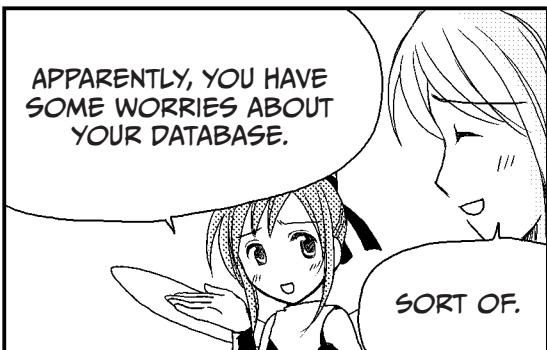
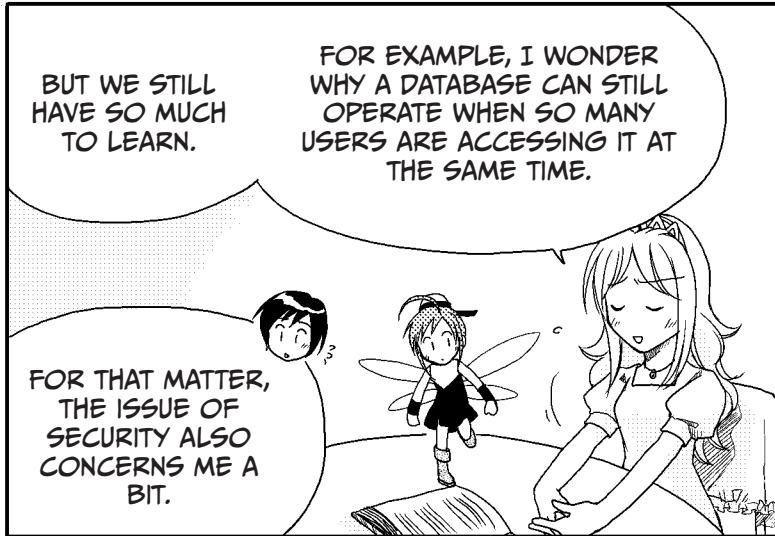
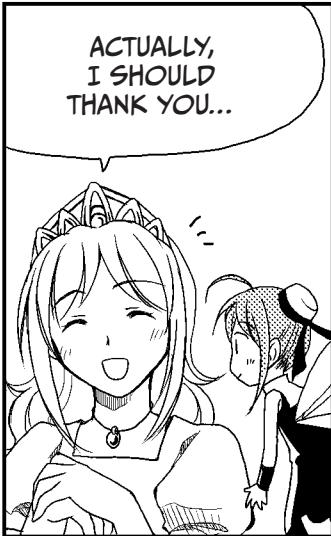
LET'S OPERATE a Database!

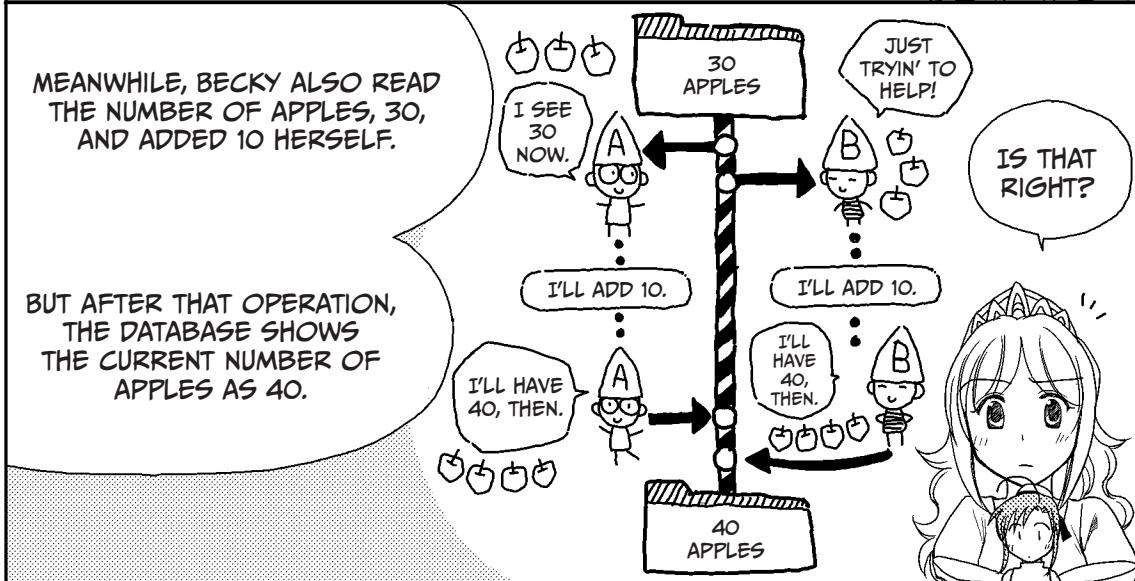
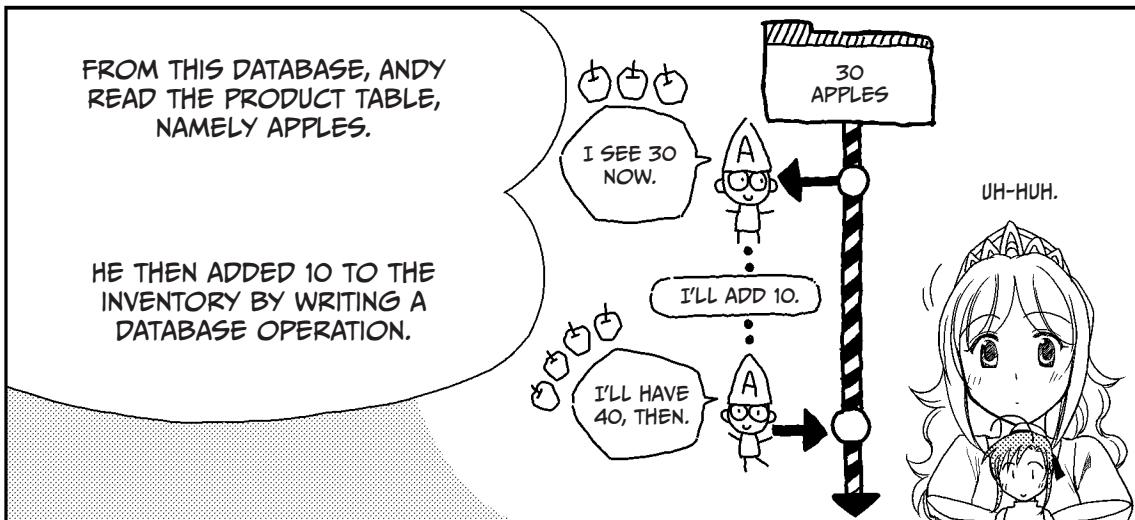
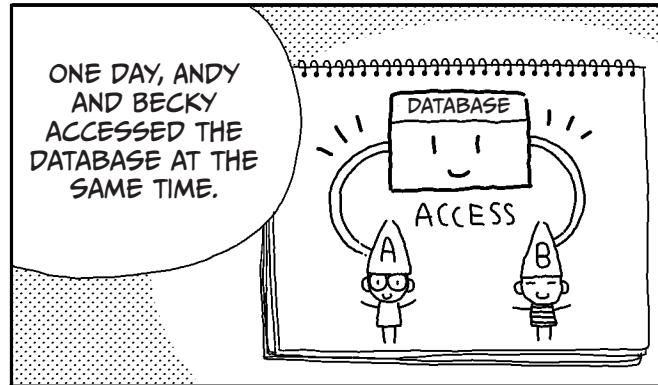
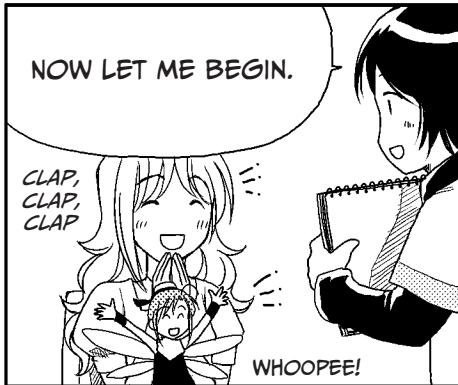


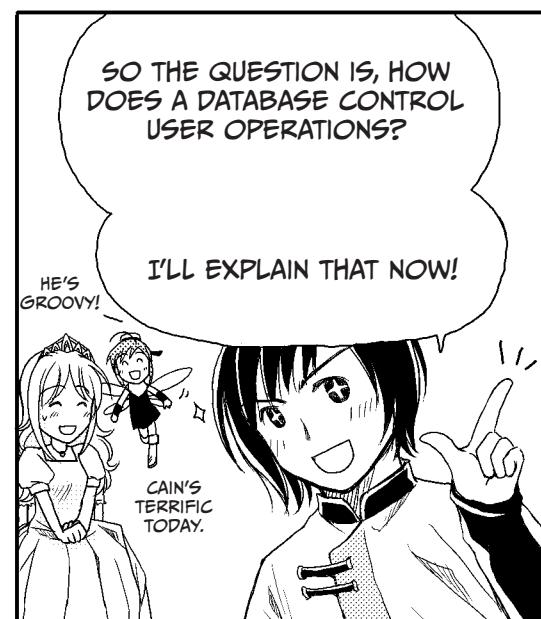
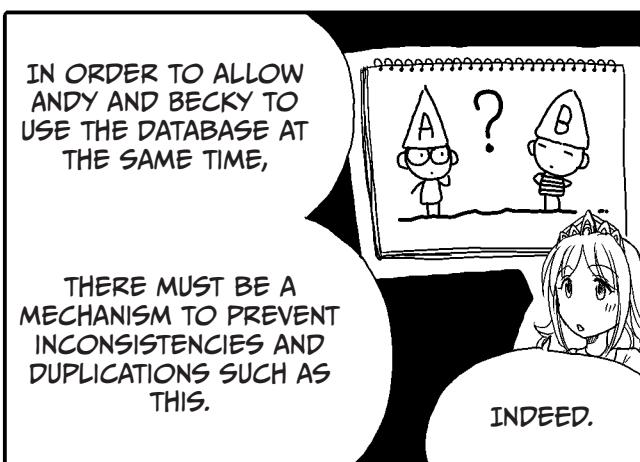
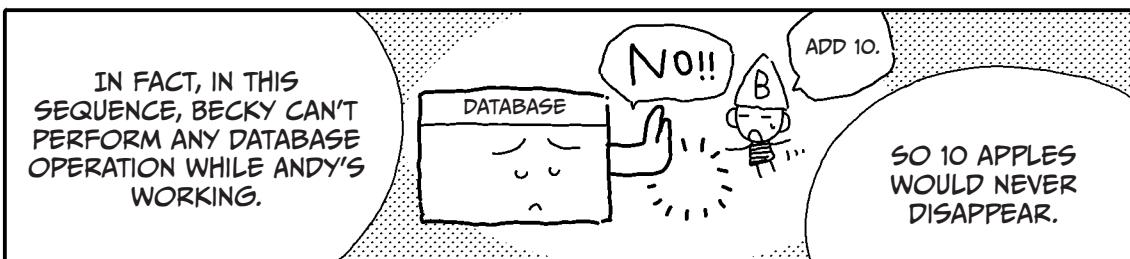
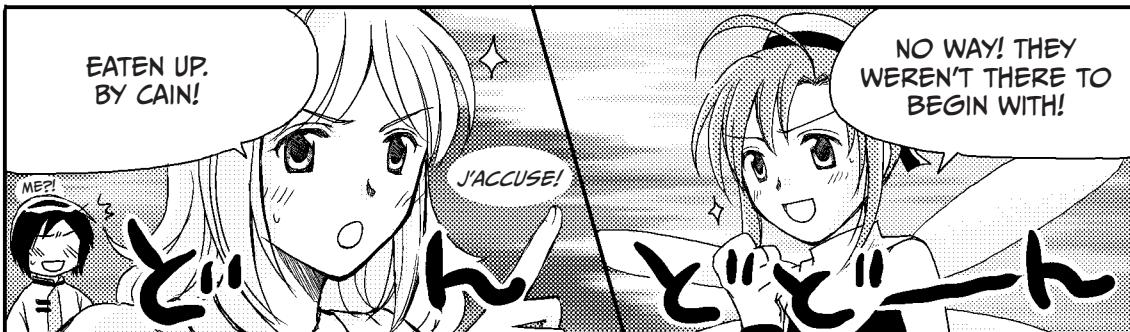
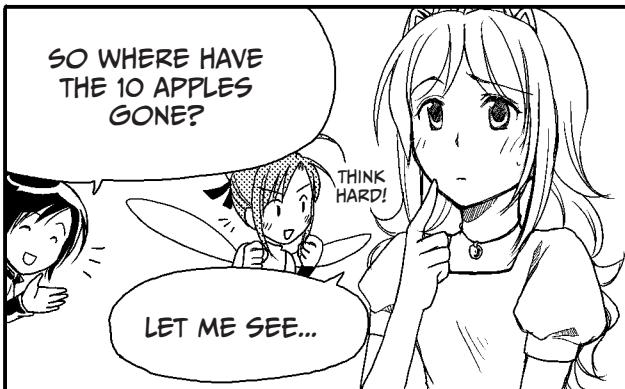
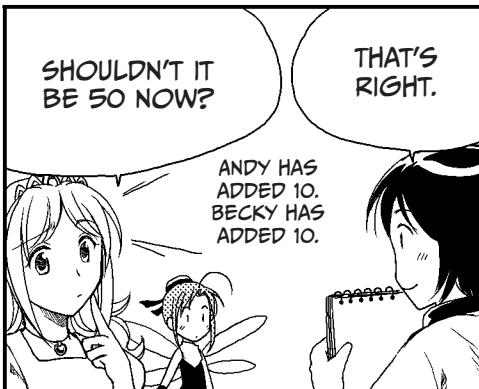
WHAT IS A TRANSACTION?

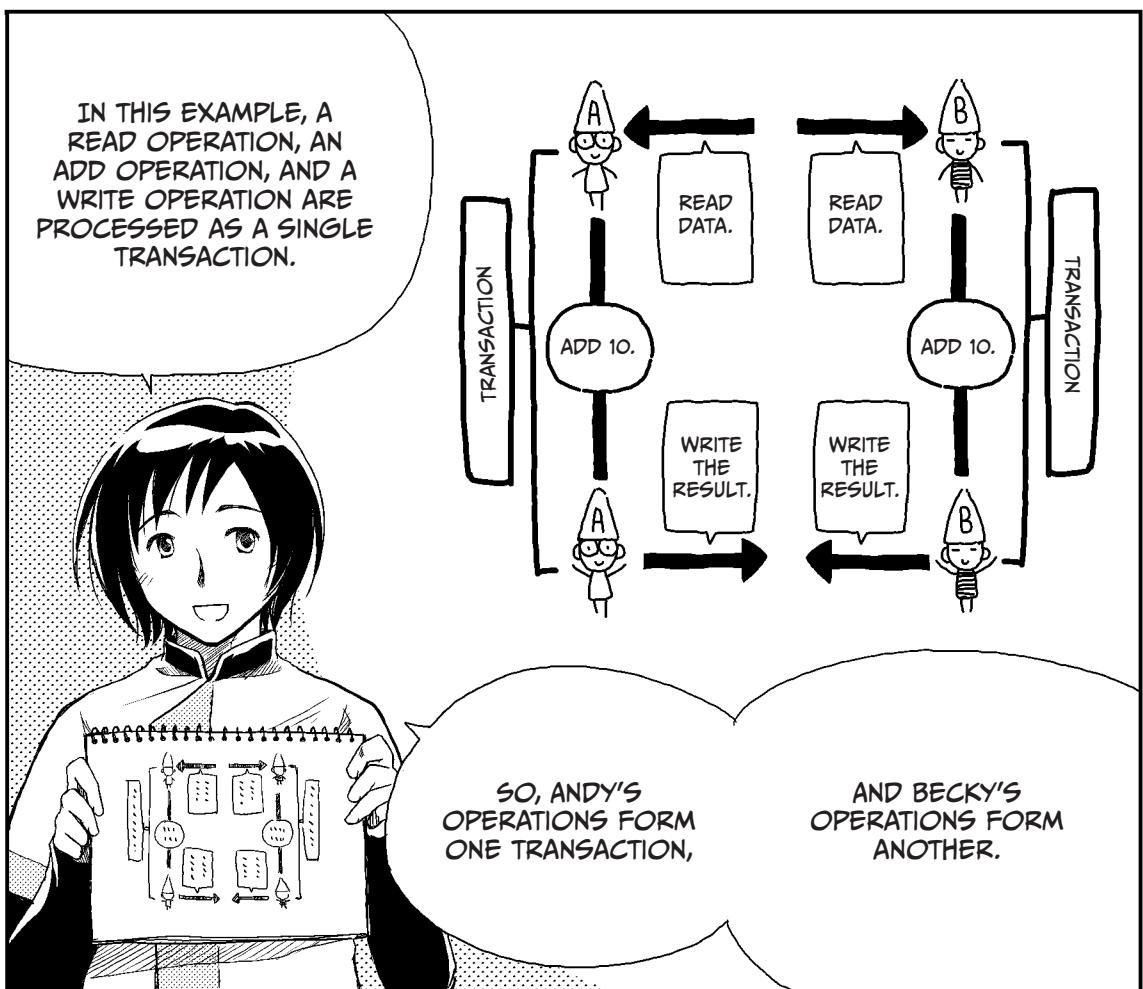
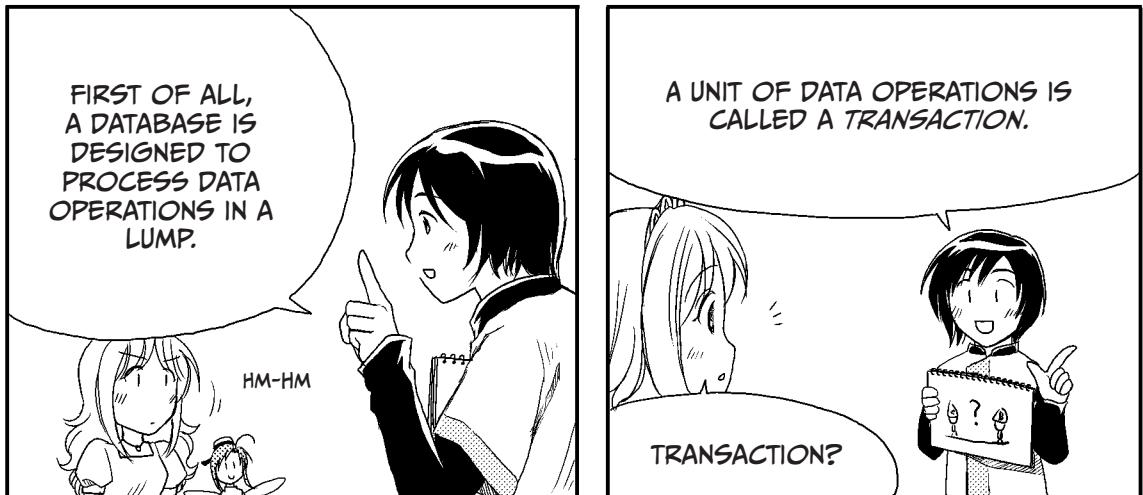


*HOP!

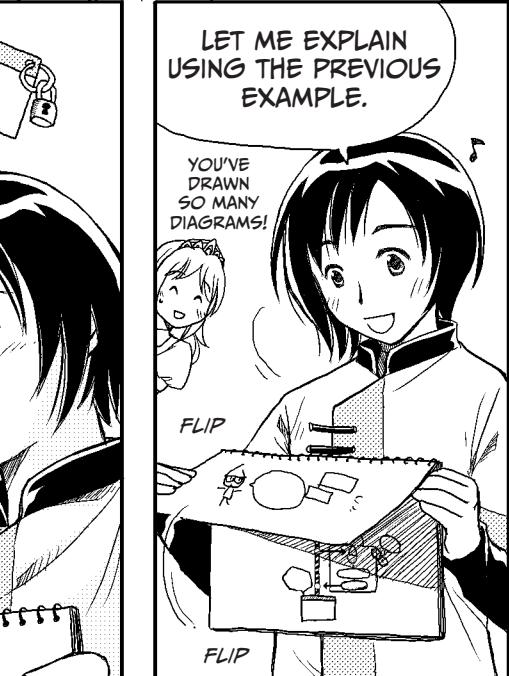
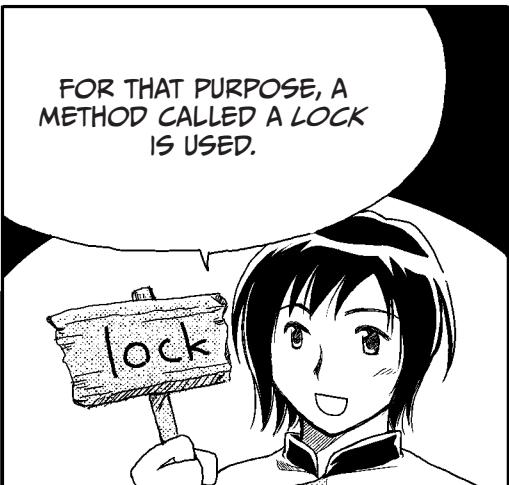
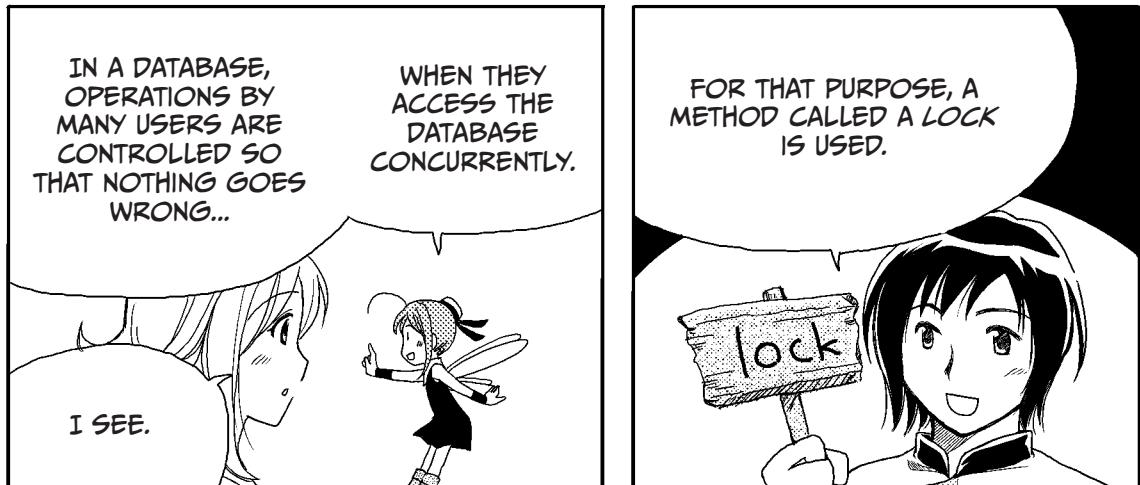






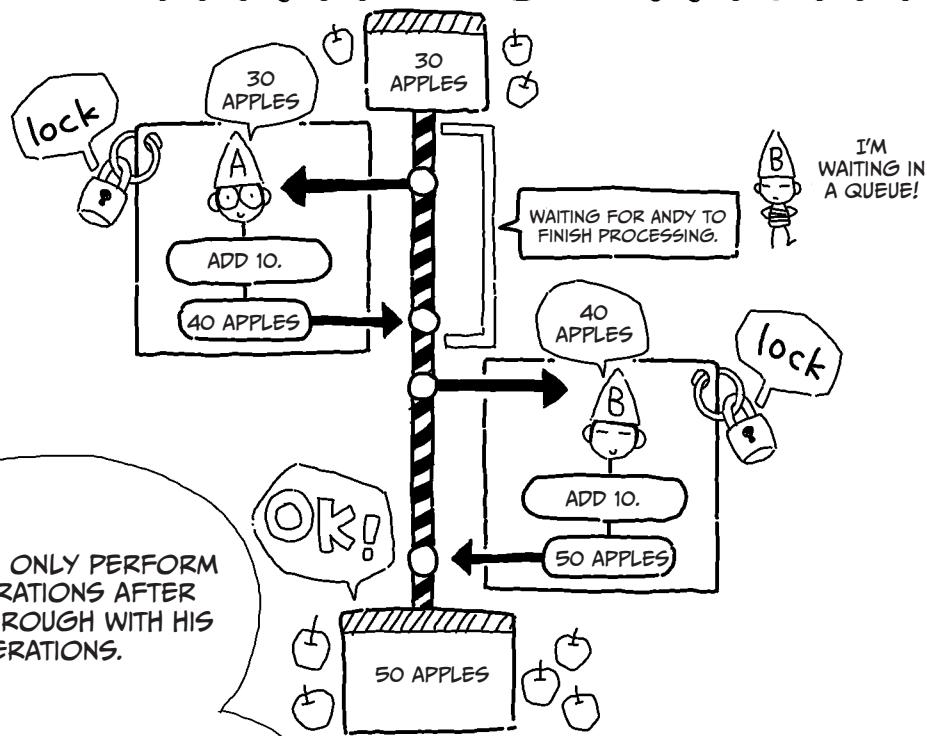


WHAT IS A LOCK?



ANDY LOCKS THE DATA BEFORE PERFORMING A SERIES OF OPERATIONS.

WHEN BECKY TRIES TO PERFORM HER OPERATIONS, SHE MUST WAIT UNTIL ANDY IS FINISHED.

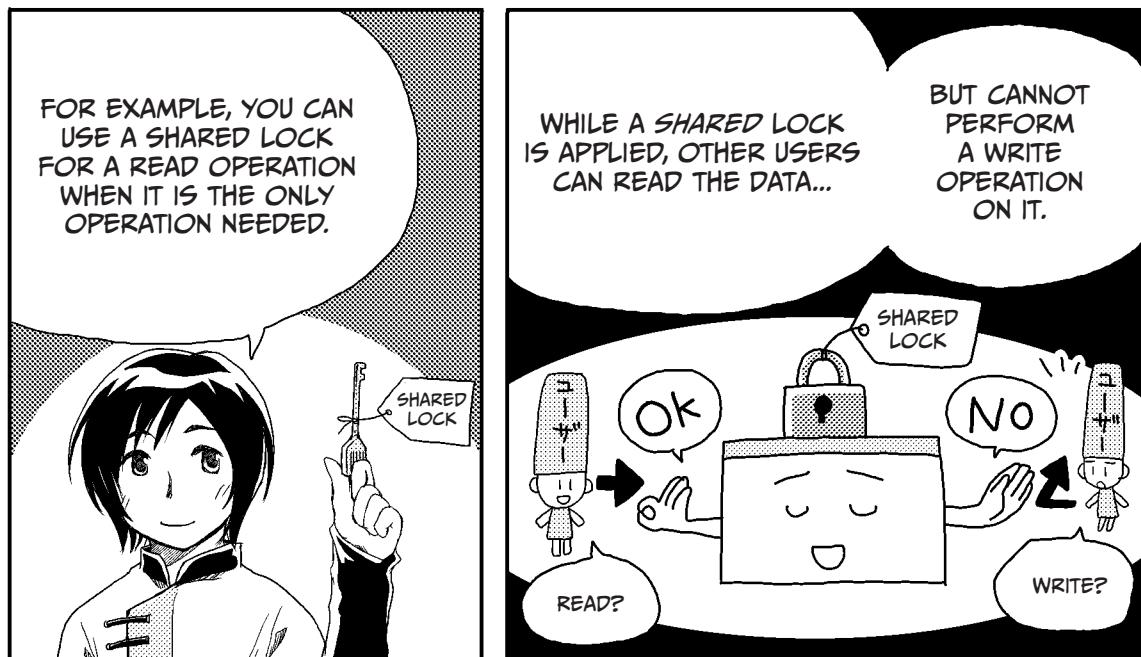
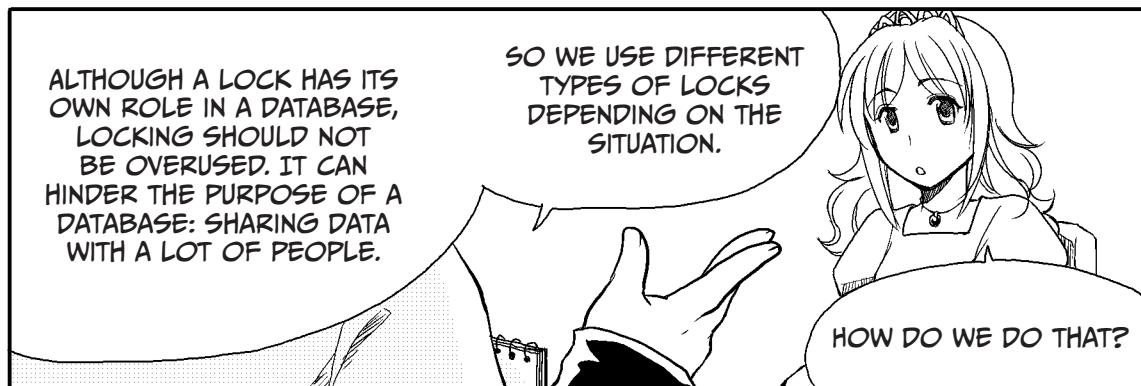
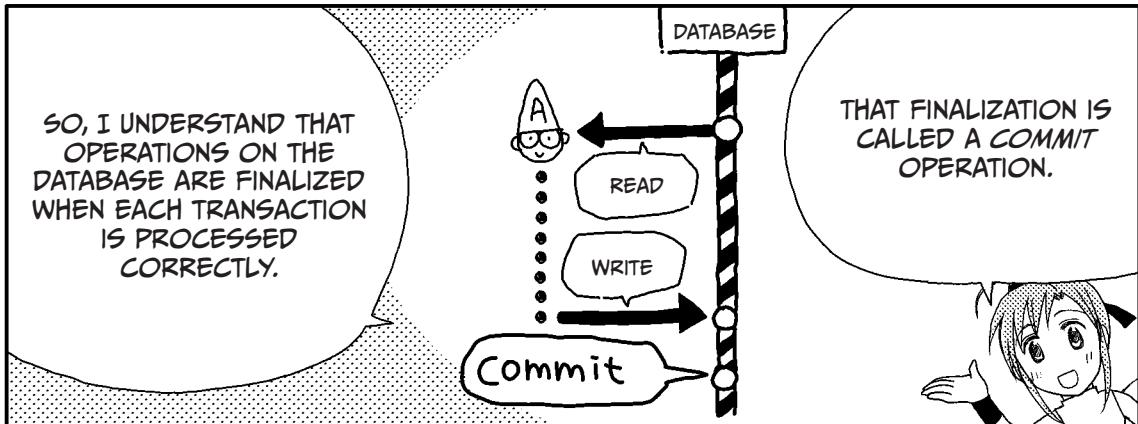


BECKY CAN ONLY PERFORM HER OPERATIONS AFTER ANDY IS THROUGH WITH HIS OPERATIONS.

AS A RESULT, THE DATABASE YIELDS THE VALUE 50, AS IT SHOULD.

OH, BOY! CAIN,
I'M IMPRESSED.

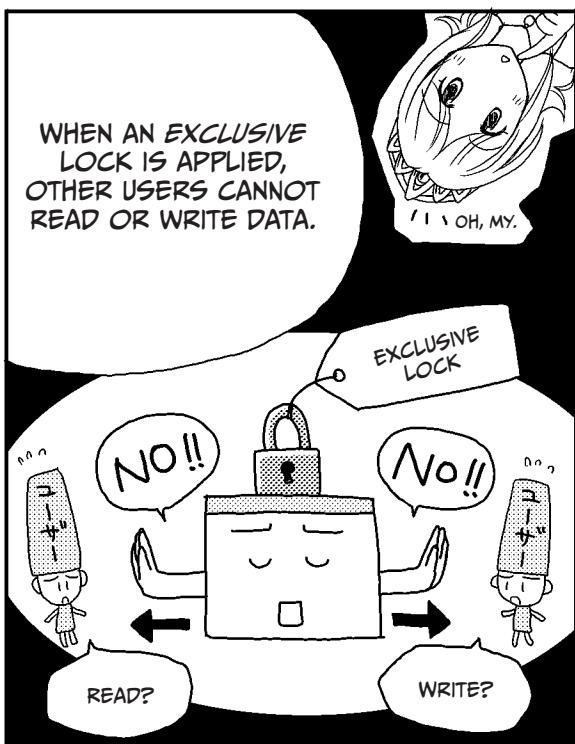
YEAH, YEAH.



WHEN PERFORMING A WRITE OPERATION, A USER APPLIES AN EXCLUSIVE LOCK.



WHEN AN EXCLUSIVE LOCK IS APPLIED, OTHER USERS CANNOT READ OR WRITE DATA.



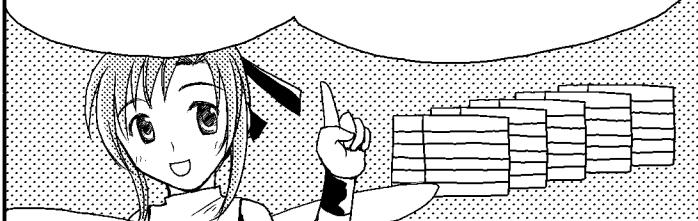
I SEE THAT THERE ARE DIFFERENT TYPES OF LOCKS.

THAT MAKES SENSE.

WHEN A LOCK IS USED TO CONTROL TWO OR MORE TRANSACTIONS, THAT IS CALLED CONCURRENCY CONTROL.

IN A DATABASE, CONCURRENCY CONTROL ALLOWS

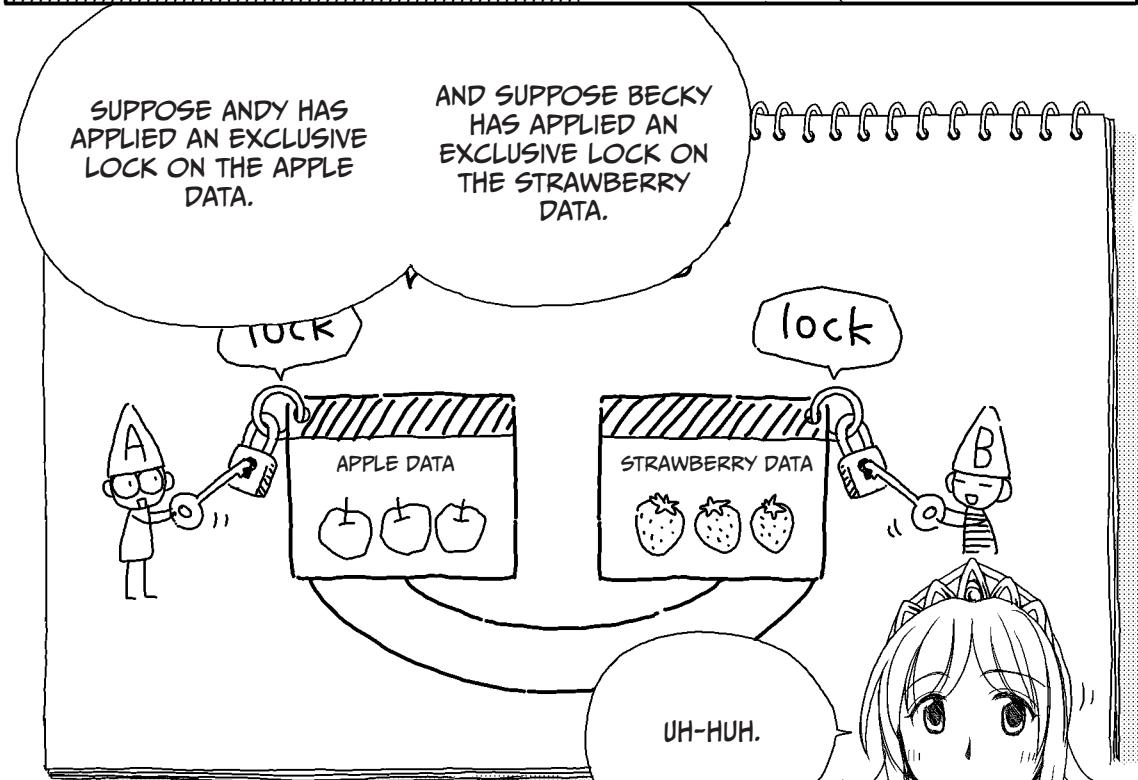
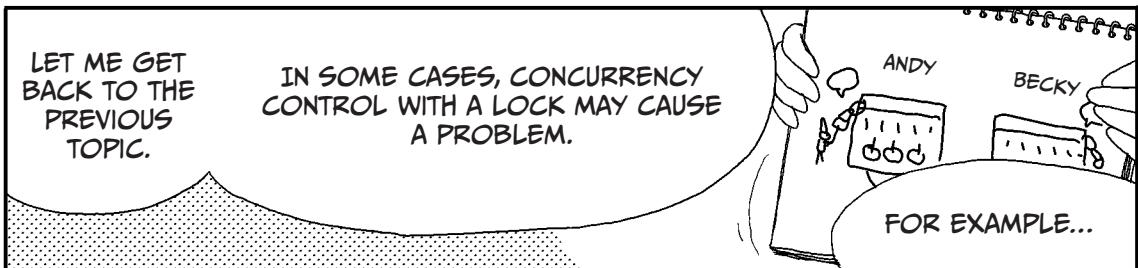
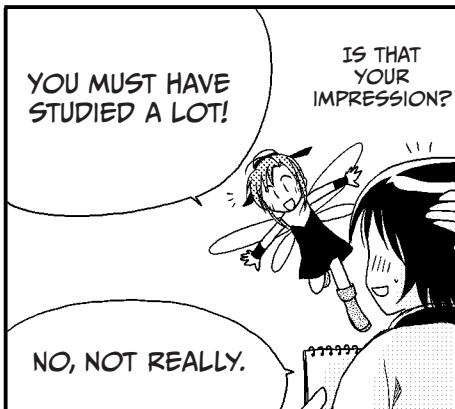
AS MANY USERS AS POSSIBLE TO USE A DATABASE AT ONE TIME WHILE PREVENTING DATA CONFLICTS FROM OCCURRING.



BY THE WAY, CAIN!

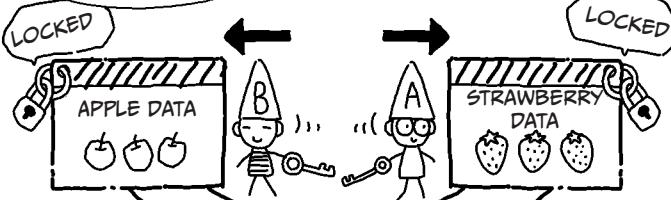
YES, TICO?





NEXT, ANDY MAY TRY TO APPLY AN EXCLUSIVE LOCK ON STRAWBERRY DATA,

AND BECKY MAY TRY TO APPLY AN EXCLUSIVE LOCK ON APPLE DATA.



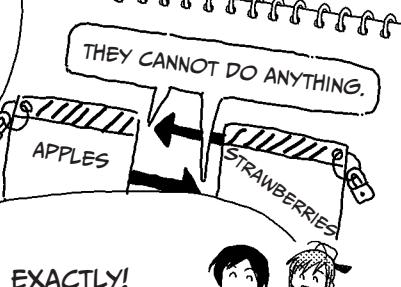
WHAT WOULD HAPPEN THEN?

WELL...

SINCE EACH OF THEM MUST WAIT FOR THE LOCK APPLIED BY THE OTHER USER TO BE RELEASED,

NEITHER ONE CAN PROCEED WITH ANY OPERATION. IS THAT IT?

LET ME THINK...



EXACTLY!

THIS SITUATION, WHICH IS CALLED A DEADLOCK, CANNOT BE SOLVED UNLESS ONE OF THE LOCKS IS RELEASED.

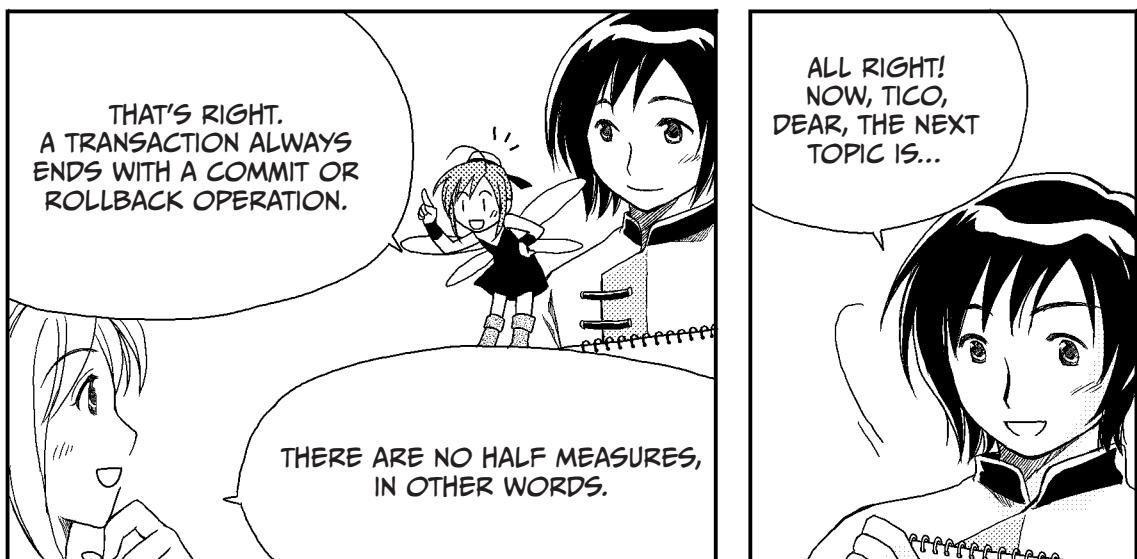
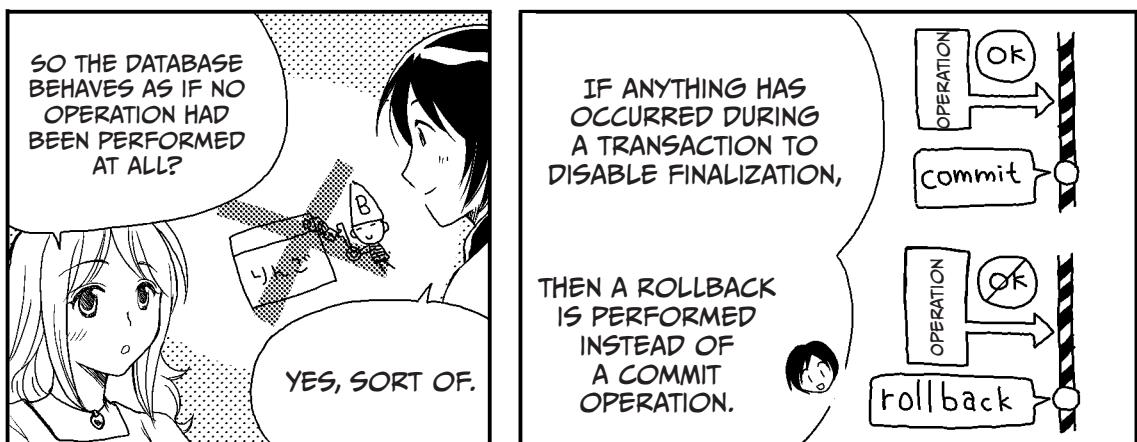
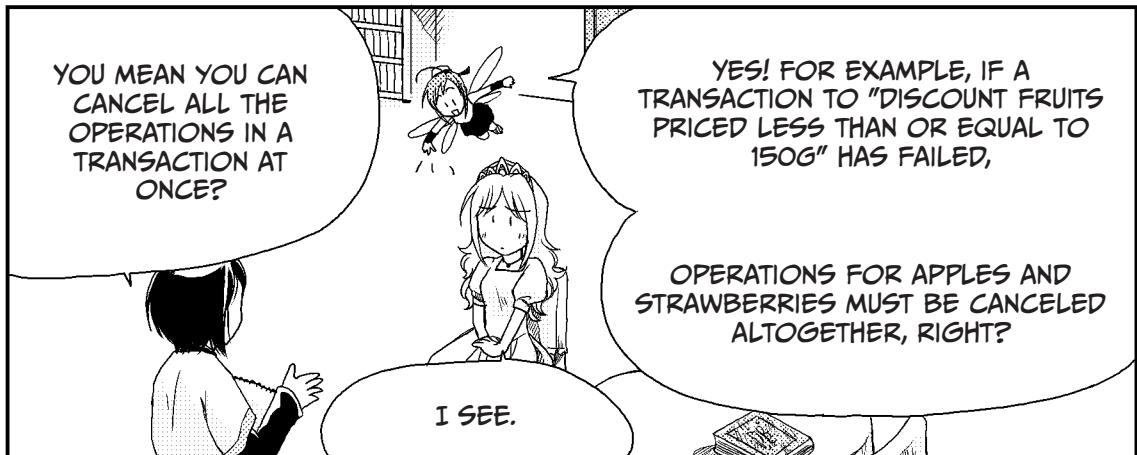
FOR EXAMPLE, YOU CAN LOOK FOR TRANSACTIONS THAT HAVE BEEN QUEUED FOR A CERTAIN LENGTH OF TIME...

CANCEL

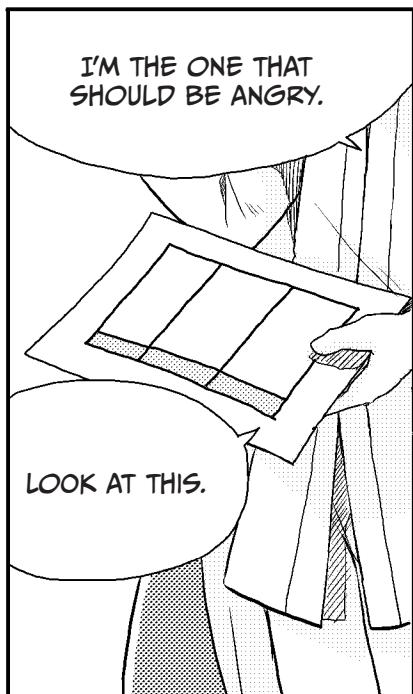
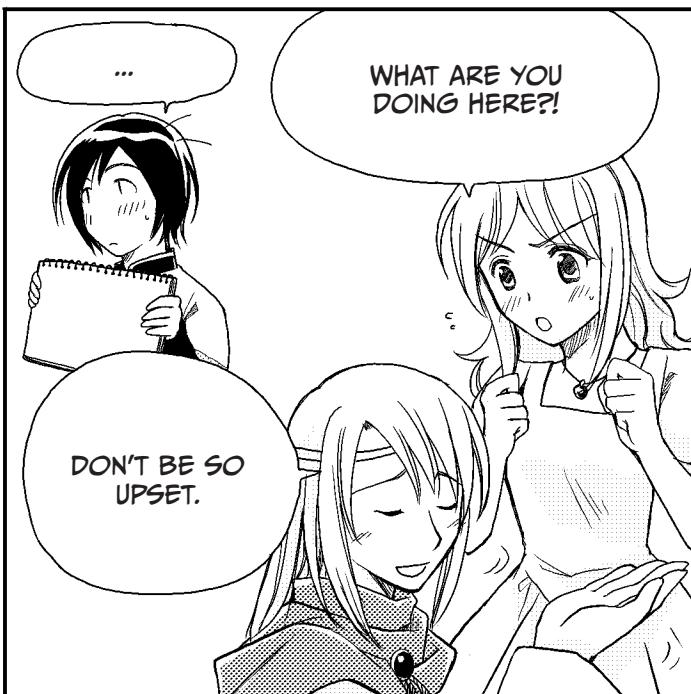
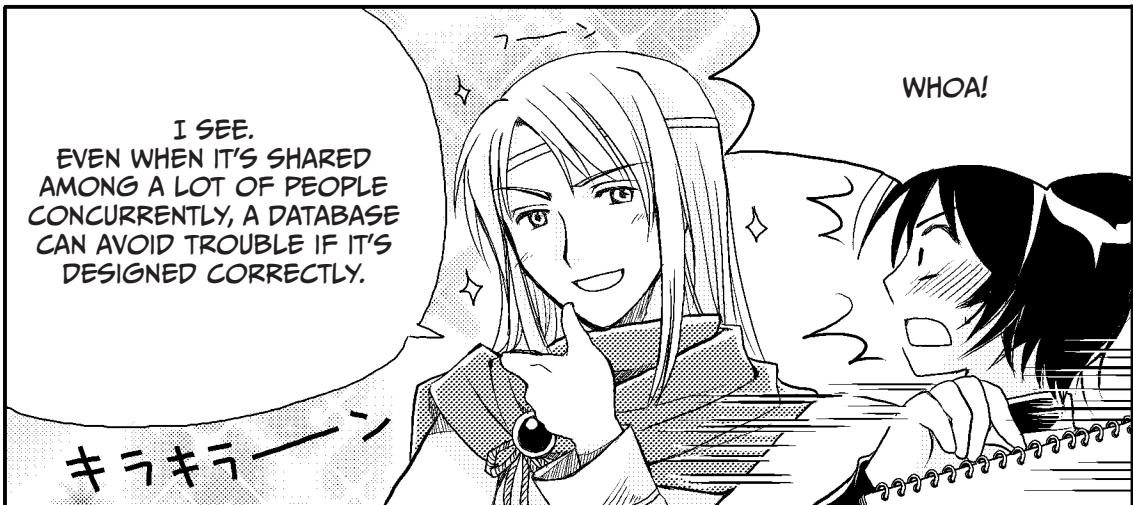
AND CANCEL THEM!

WHEN YOU CANCEL A TRANSACTION, IT'S CALLED A ROLLBACK.

ROLLBACK?



DATABASE SECURITY



OUR PRODUCT TABLE.



PRODUCT CODE	PRODUCT NAME	UNIT PRICE
101	MELON	10,000G
102	STRAWBERRY	12,500G
103	APPLE	8,000G
104	LEMON	6,000G
201	CHESTNUT	9,000G
202	PERSIMMON	12,400G
301	PEACH	5,000G
302	KIWI	6,000G

WHAT'S WRONG WITH IT?

THE PRICES.
THE PRICES!

?, THE PRICES?

THE FIGURES
IN THE UNIT PRICE
COLUMN ARE ALL
MESSED UP!

OH NO!!!

HOW COME?!

THAT'S ONE
EXPENSIVE
MELON!!

BECUSE YOUR INVOICES
ARE SUCH A MESS, MY
COUNTRY, AN IMPORTER, IS
EXPERIENCING HAVOC.

A DATABASE IS A
NASTY THING.

AS PART OF THE
COMPENSATION FOR
THIS TROUBLE...

SOMEONE WITH
MALICIOUS INTENT MIGHT
HAVE PERFORMED AN
UNAUTHORIZED DATA
OVERWRITE.

HOW
AWFUL!

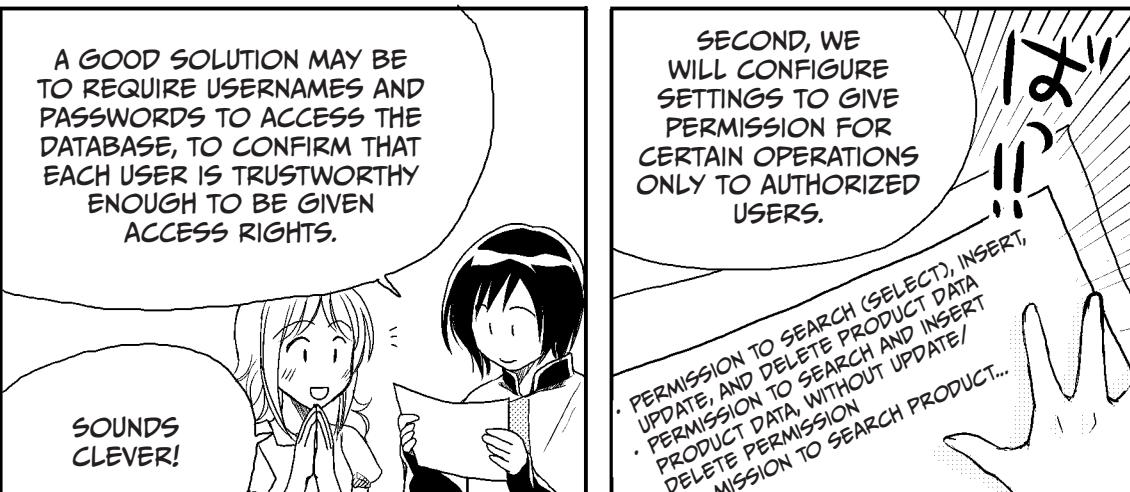
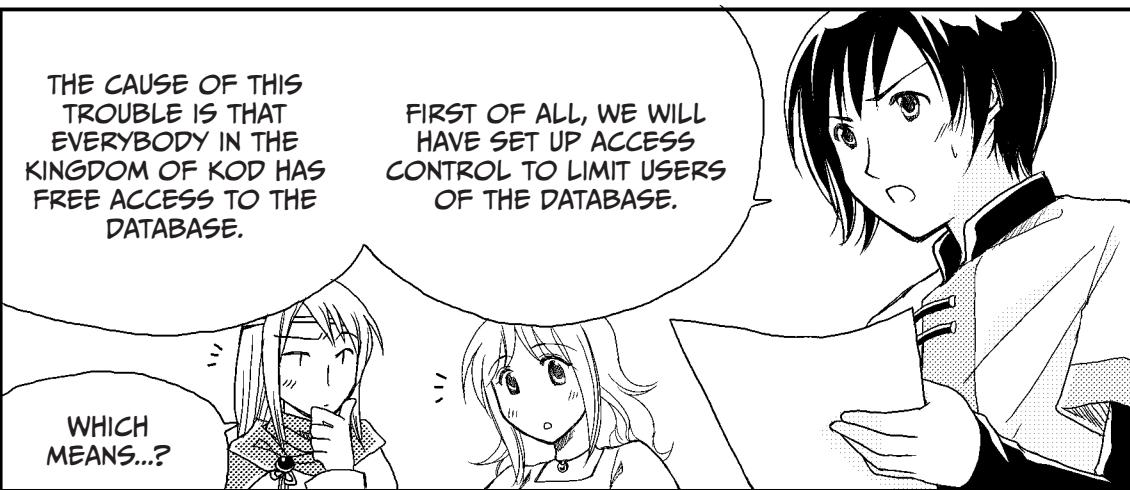
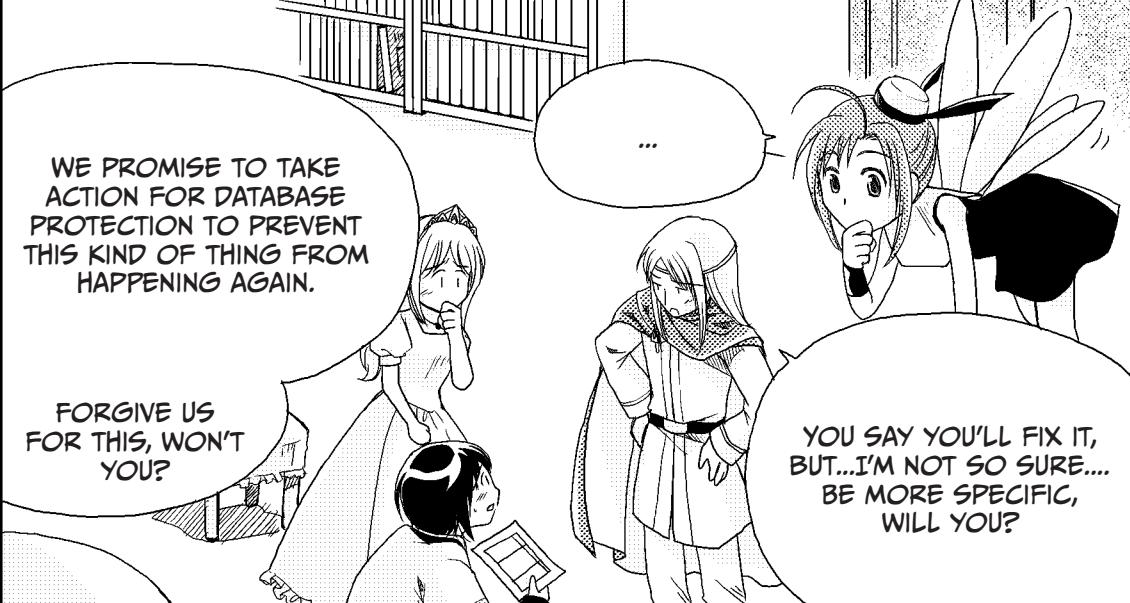
YOU ARE
BEHAVING
AS IF I
WERE NOT
HERE.

WHY DON'T YOU ACCEPT MY
PROPOSAL, PRINCESS RURUNA?
COME OVER TO MY COUNTRY
AND BE MY BRIDE.

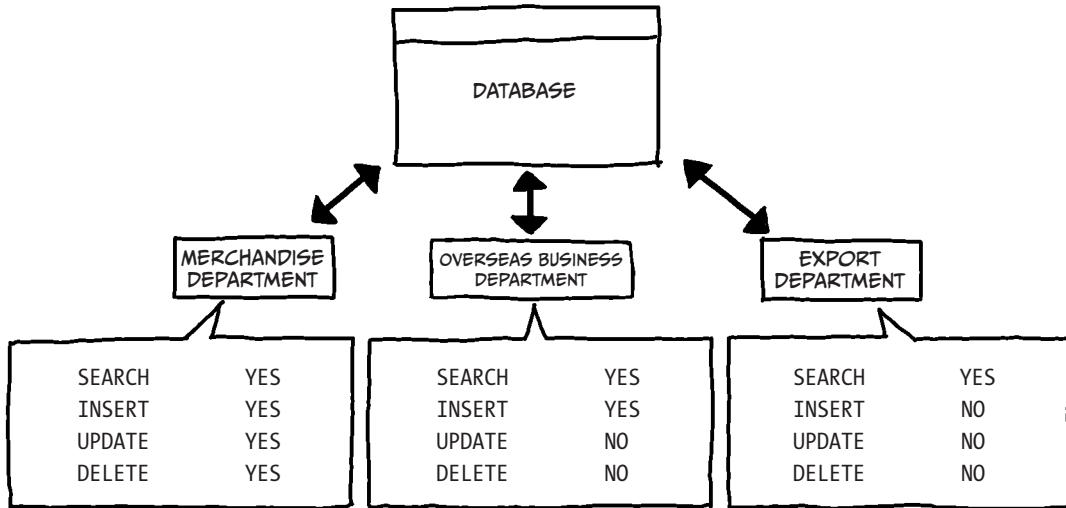
IGNORING

PRINCE RAMINESS,

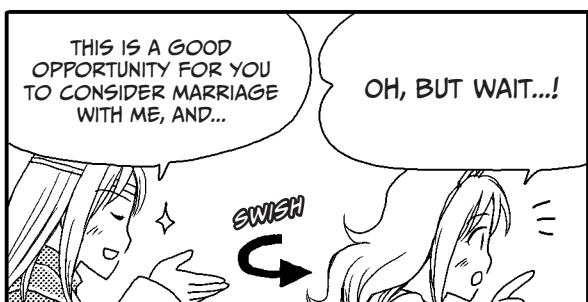
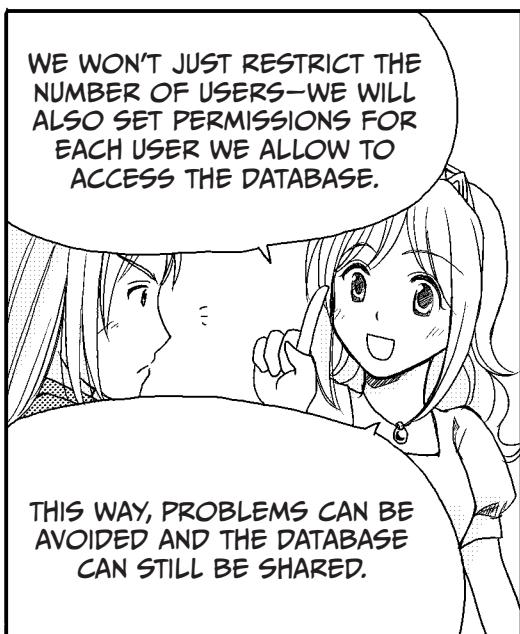
WE ARE VERY
SORRY.



- MERCHANDISE DEPARTMENT PERSONNEL MAY SEARCH, INSERT, UPDATE, AND DELETE PRODUCT DATA.
- OVERSEAS BUSINESS DEPARTMENT PERSONNEL MAY SEARCH AND INSERT PRODUCT DATA, BUT THEY ARE NOT ALLOWED TO UPDATE OR DELETE IT.
- EXPORT DEPARTMENT PERSONNEL MAY SEARCH PRODUCT DATA, BUT THEY ARE NOT ALLOWED TO INSERT, UPDATE, OR DELETE IT.



はーん!!



SPEEDING THINGS UP WITH INDEXING



SUPPOSE WE ARE GOING TO FIND THE MEANING OF THE TERM TRANSACTION BY CONSULTING THIS BOOK ON DATABASES.

A BLIND SEARCH FOR THE TERM WOULD BE PAINSTAKING, SO WE WOULD CHECK THE INDEX, INSTEAD.

UNDER THE ENTRY FOR TRANSACTION, PAGES THAT DISCUSS TRANSACTIONS ARE LISTED.

INDEX

REFERENCE PAGE

BY USING AN INDEX, WE CAN QUICKLY FIND THE PAGE WE'RE LOOKING FOR!

YOU'RE RIGHT.

IT'S JUST THE SAME FOR A DATABASE INDEX. FOR EXAMPLE...

IF YOU CREATE INDEXES FOR PRODUCT CODES,

IT TELLS YOU WHERE ON THE DISK THAT PRODUCT DATA IS LOCATED.

YOU CAN INSTANTLY LEARN WHERE PRODUCT DATA IS STORED FOR A PRODUCT ASSIGNED PRODUCT CODE 101.



INDEXING HELPS SPEED UP THE SEARCH.

UH-HUH.

WELL, IT'S NOT SO EASY FOR ME TO FOLLOW.... ANYWAY...

IT IS VERY TIME CONSUMING TO BROWSE ALL ROWS WHEN SEARCHING FOR CERTAIN DATA.

SHALL I REPEAT THE EXPLANATION FROM THE START? ♫

NO, NO...

USING INDEXES, WE CAN REDUCE THE DISK ACCESS COUNT.

REDUCE THE DISK ACCESS COUNT, AND OUR SEARCH WILL BE MUCH FASTER!

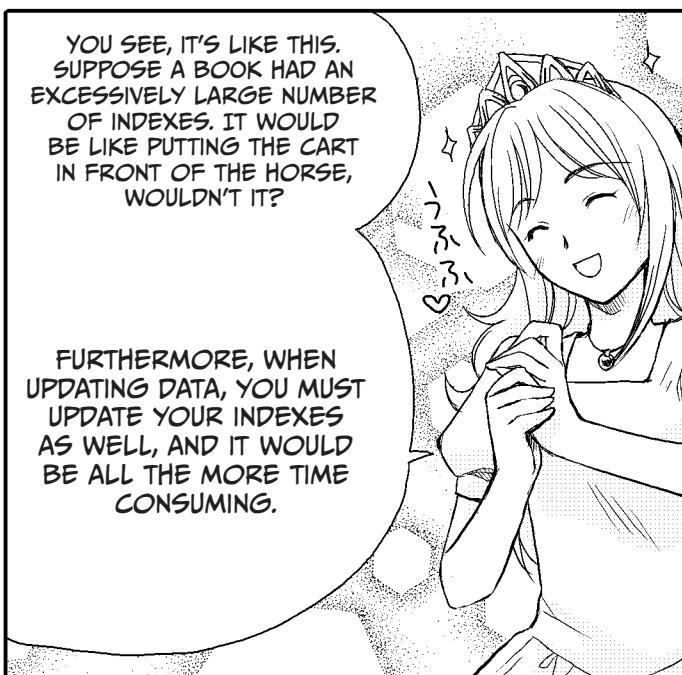
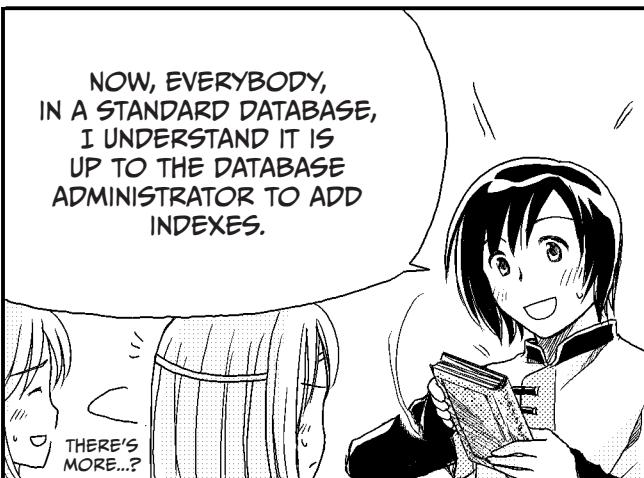
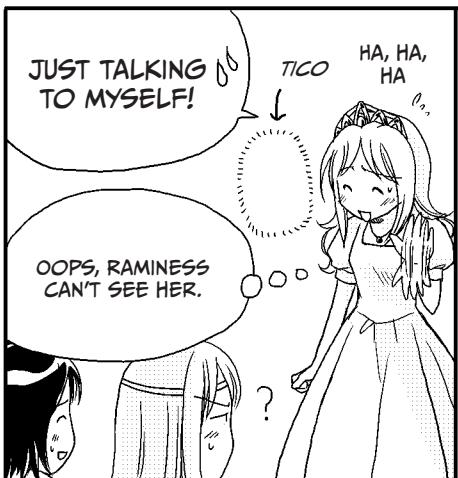
WHAT'S THAT?

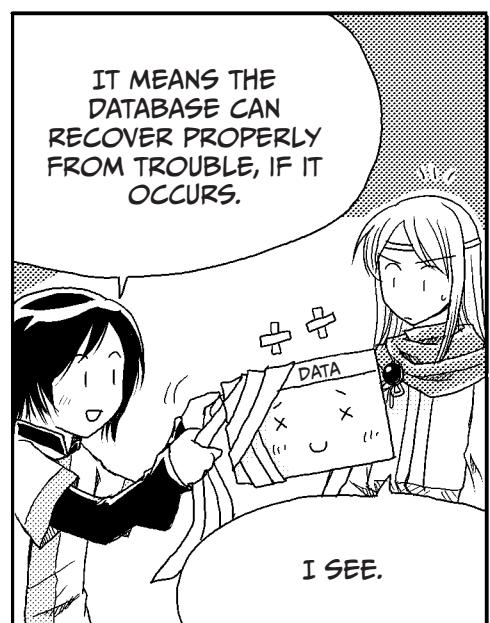
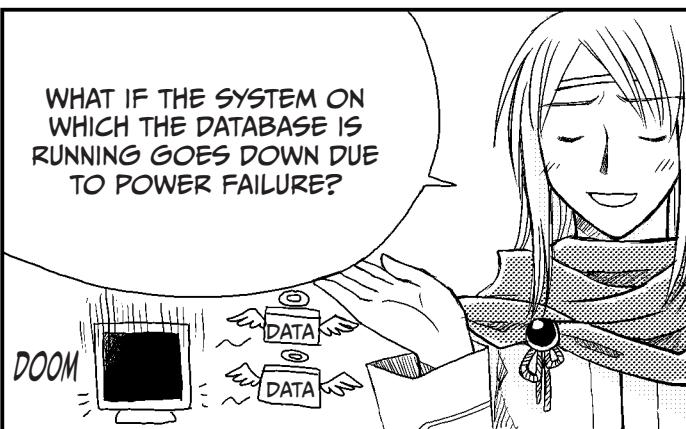
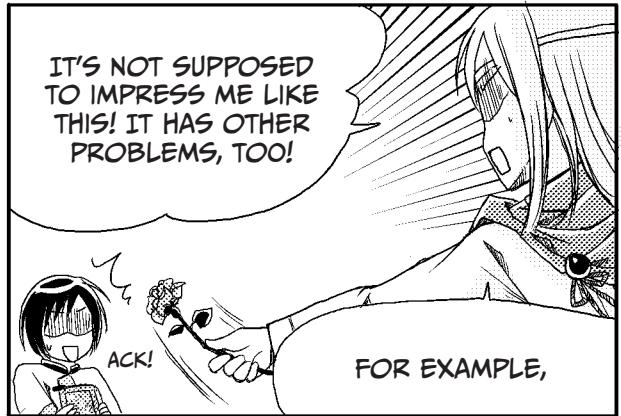
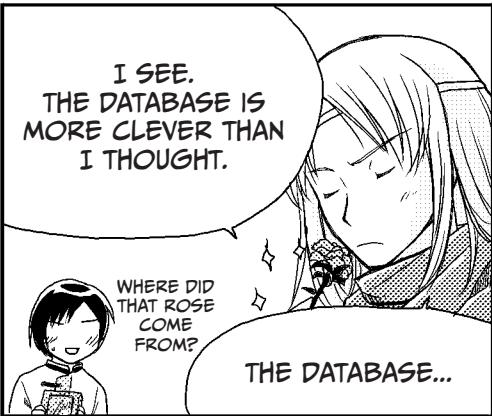
UH-OH.

HEY! WHO ARE YOU TALKING TO?

IS SOMEONE HERE?

PRINCESS, BE CAREFUL!



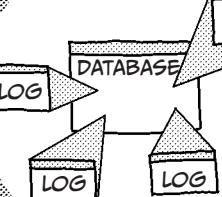


DISASTER RECOVERY

INSIDE A DATABASE,
RECORDS CALLED LOGS
ARE KEPT WHENEVER
A DATA OPERATION IS
PERFORMED, AREN'T THEY?



EXACTLY.



THAT'S HOW CHANGES TO
THE DATABASE'S CONTENTS
ARE RECORDED.

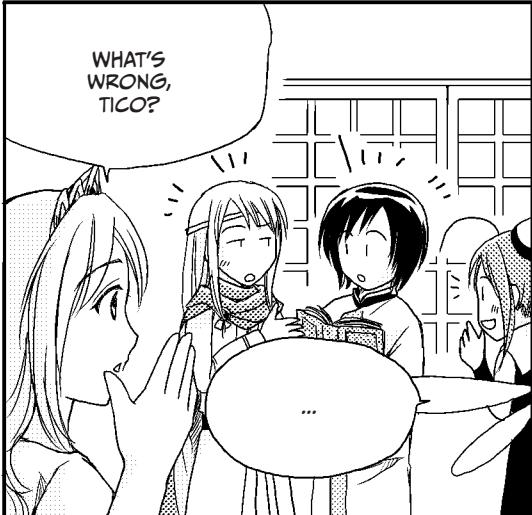
LOGS, HUH...



MOST IMPORTANT ARE RECORDS
OF THE VALUES BEFORE AND
AFTER A DATABASE UPDATE.

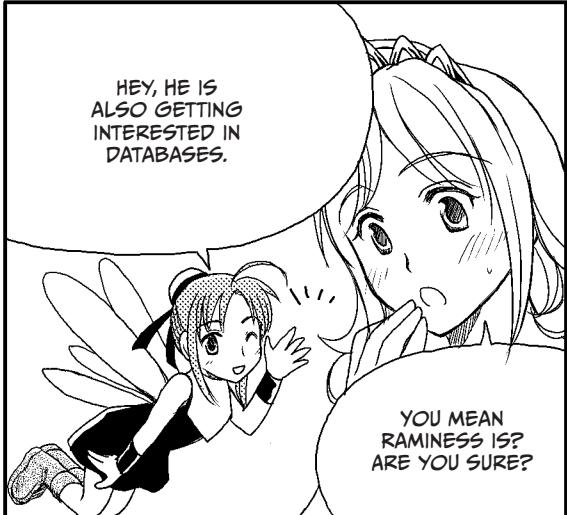
HMM...

WHAT'S
WRONG,
TICO?



...

HEY, HE IS
ALSO GETTING
INTERESTED IN
DATABASES.



YOU MEAN
RAMINESS IS?
ARE YOU SURE?

WHEN A PROBLEM HAS OCCURRED IN THE SYSTEM, FIRST YOU RESTART THE SYSTEM.

THE RECOVERY METHOD VARIES DEPENDING ON WHETHER OR NOT THE TRANSACTION HAS BEEN COMMITTED.

THEN YOU UTILIZE LOGS TO RECOVER THE DATABASE.

TRANS-
WHAT?

BLAB,
BLAB

APPARENTLY,
HE IS NOT SO
KNOWLEDGEABLE...
THAT'S A PITY.

YEAH.

I DON'T
REALLY
NEED
IT...

I'LL
ILLUSTRATE
IT...

SQUEAK,
SQUEAK

IF THE PROBLEM OCCURRED AFTER A TRANSACTION HAD ALREADY BEEN COMMITTED, THAT MEANS THE OPERATIONS HAVE BEEN FINALIZED FOR THAT TRANSACTION.

SO, YOU CAN RECOVER THE DATA BY REAPPLYING THE OPERATIONS TO THE DATABASE.

IN THIS RECOVERY, THE DATABASE REFERENCES THE VALUE AFTER THE UPDATE.

DATA A



ROLL
FORWARD

DATA A
AFTER AN UPDATE
OPERATION



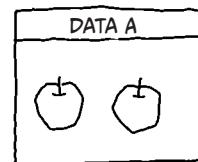
THIS RECOVERY METHOD IS CALLED ROLLING FORWARD.

WHAT IF THE TRANSACTION HASN'T BEEN COMMITTED YET WHEN THE PROBLEM OCCURS?



DON'T WORRY!
IN THAT CASE, A ROLLBACK TAKES PLACE.

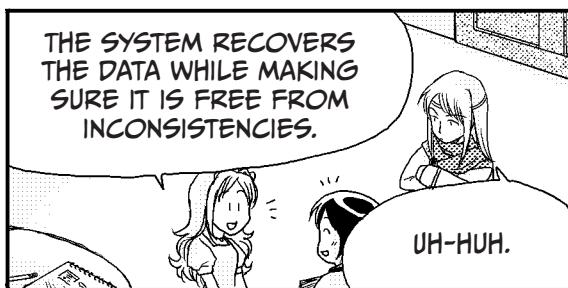
IN A ROLLBACK OPERATION, THE VALUE BEFORE THE UPDATE IS REFERENCED TO CANCEL THE TRANSACTION.



IN OTHER WORDS, IT RESTORES THE STATE OF THE DATABASE BEFORE THE TRANSACTION WAS STARTED.



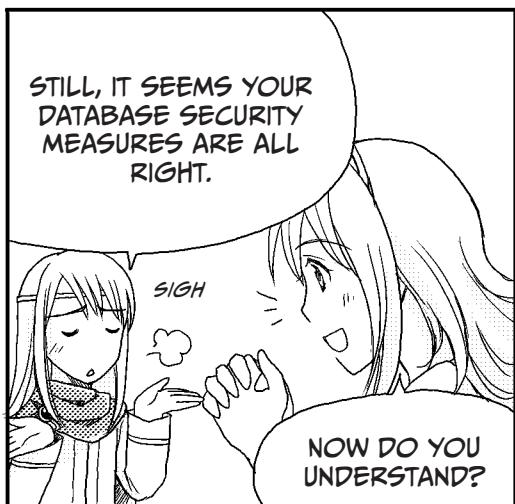
THE SYSTEM RECOVERS THE DATA WHILE MAKING SURE IT IS FREE FROM INCONSISTENCIES.



I AM NOT FAMILIAR WITH TERMS LIKE COMMIT AND TRANSACTION.

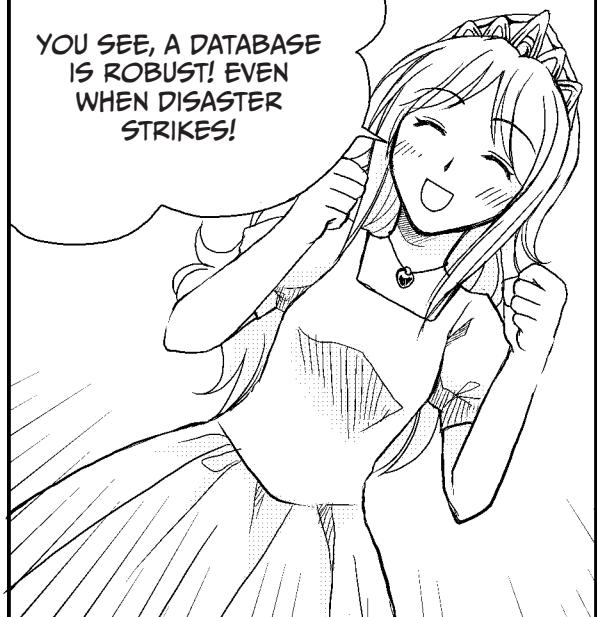
HMM

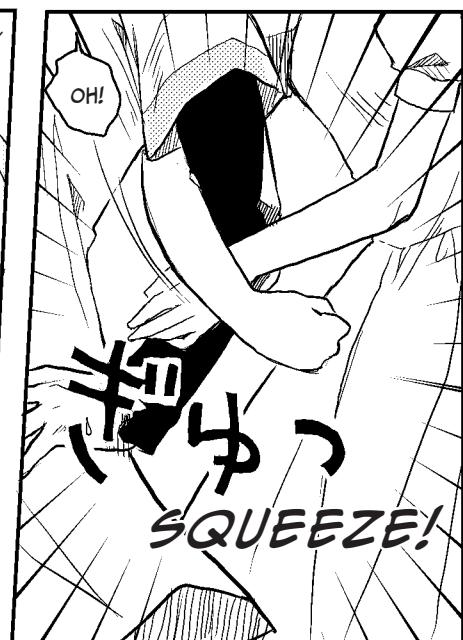
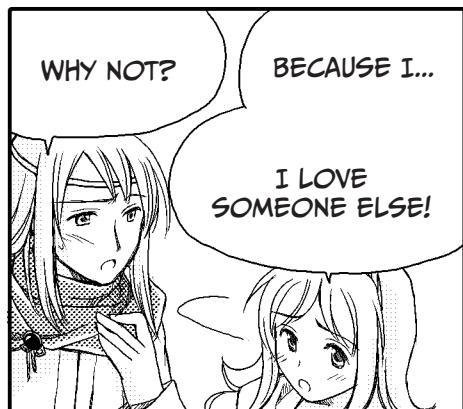
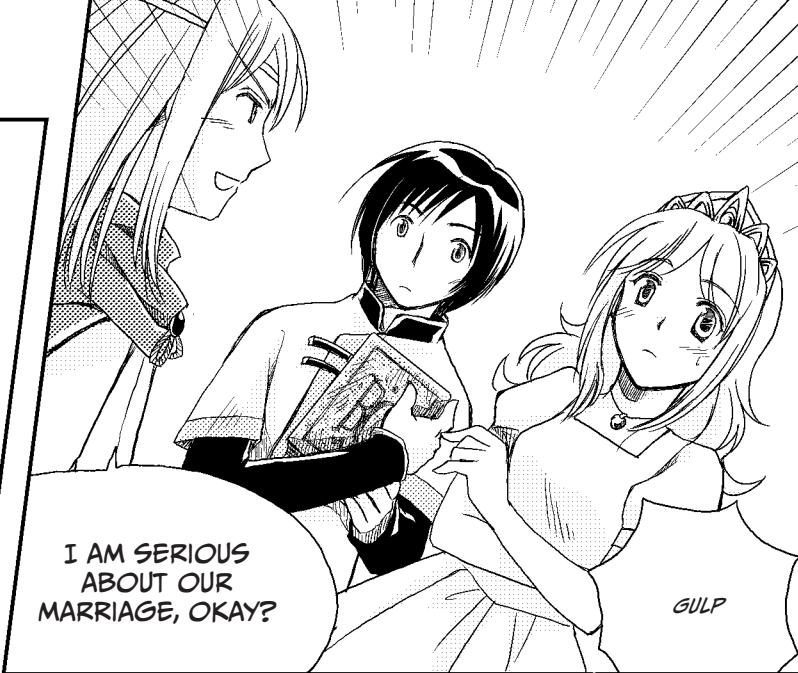
STILL, IT SEEMS YOUR DATABASE SECURITY MEASURES ARE ALL RIGHT.

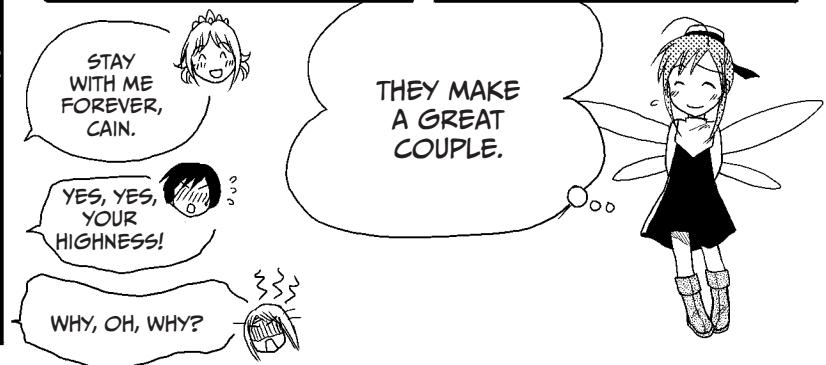
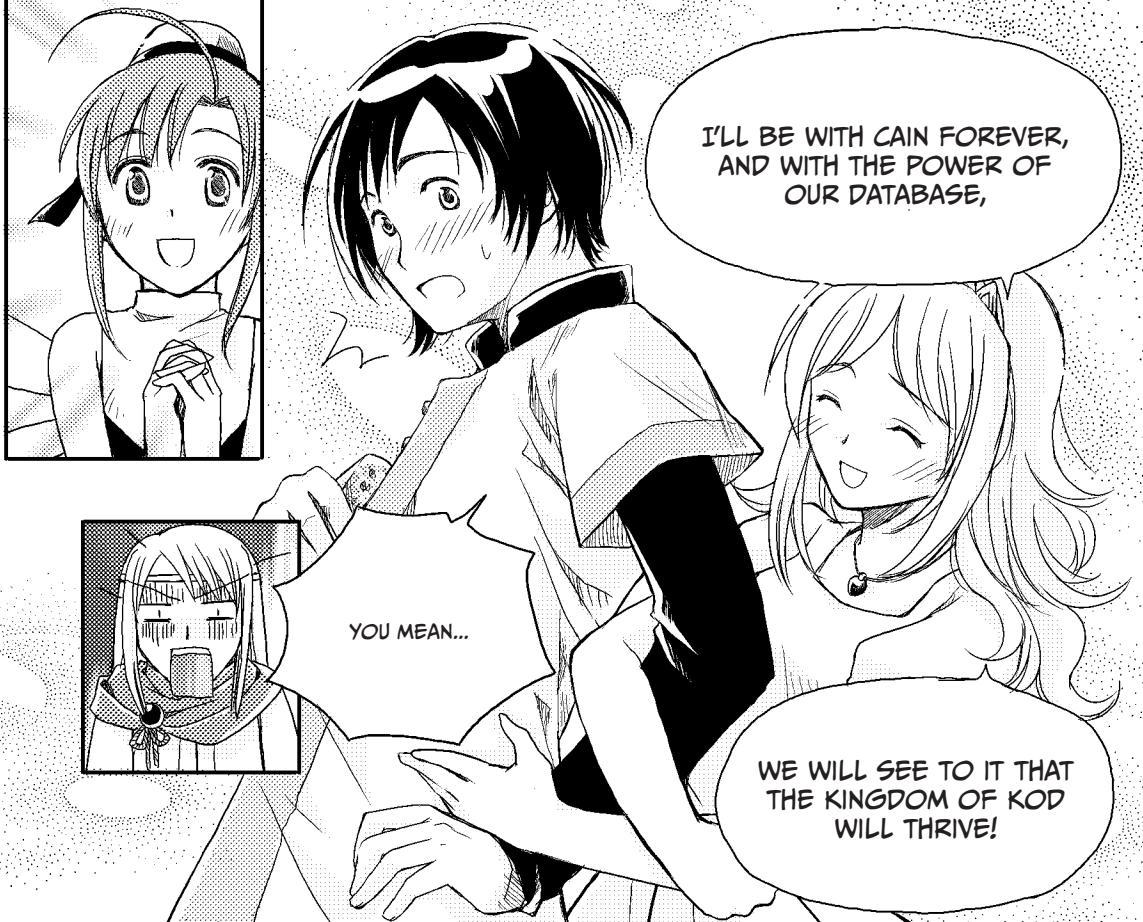


NOW DO YOU UNDERSTAND?

YOU SEE, A DATABASE IS ROBUST! EVEN WHEN DISASTER STRIKES!



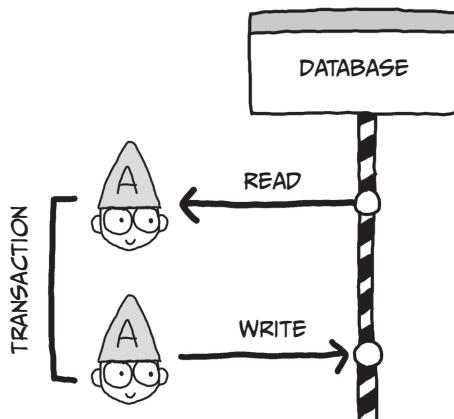




PROPERTIES OF TRANSACTIONS



Cain's research showed that users of a database can search for, insert, update, and delete data. A set of successful operations performed by a single user is called a *transaction*.



When users share a database, it is important to ensure that multiple transactions can be processed without causing conflicting data. It is also important to protect data from inconsistencies in case a failure occurs while a transaction is being processed. To that end, the following table lists the properties required for a transaction, which memorably spell the word *ACID*.

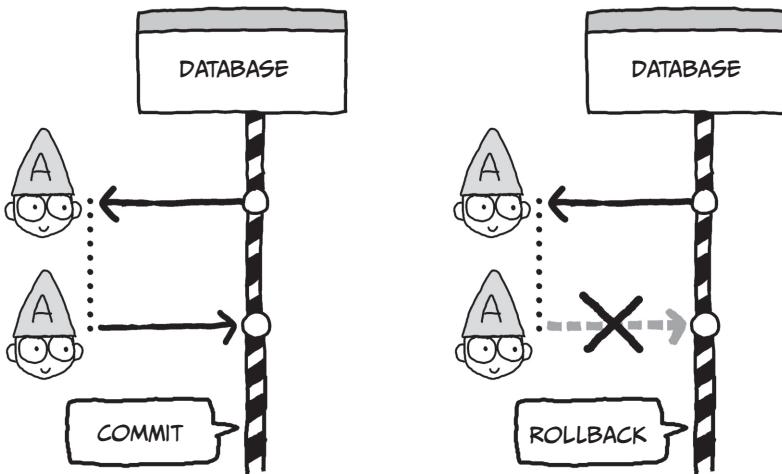
PROPERTIES REQUIRED FOR A TRANSACTION

Property	Stands for	Description
A	Atomicity	A transaction must either end with a commit or rollback operation.
C	Consistency	Processing a transaction never results in loss of consistency of the database.
I	Isolation	Even when transactions are processed concurrently, the results must be the same as for sequential processing.
D	Durability	The contents of a completed transaction should not be affected by failure.

Let's examine each of these properties in depth.

ATOMICITY

The first property required for a transaction, *atomicity*, means that a transaction must end with either a commit or rollback in order to keep a database free of inconsistencies. In short, either all actions of a transaction are completed or all actions are canceled. A *commit* finalizes the operation in the transaction. A *rollback* cancels the operation in the transaction.



In some cases, a commit or rollback is performed automatically. You can also specify which one should be carried out. For example, you can specify a rollback if an error occurs. You can use the SQL statements COMMIT and ROLLBACK to perform these operations.

`COMMIT;`

Use this statement to commit a transaction.

`ROLLBACK;`

Use this statement to roll back a transaction.

QUESTIONS



Answer these questions to see how well you understand atomicity. The answers are on page 167.

Q1

Write an SQL statement that can be used to finalize a transaction.

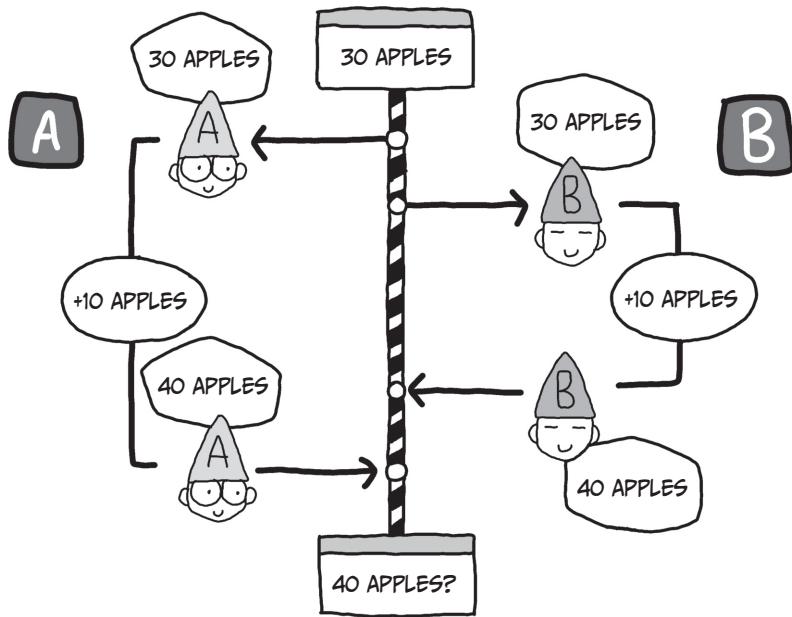
Q2

Write an SQL statement that can be used to cancel a transaction.

CONSISTENCY

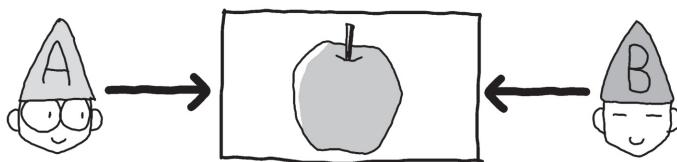
A transaction must not create errors. If the database was consistent before a transaction is processed, then the database must also be consistent after that transaction occurs.

Cain gave the example of Andy and Becky each trying to add 10 apples to an original total of 30 apples. Rather than yielding the correct amount of 50 apples, the database shows a total of 40 apples. This type of error is called a *lost update*.



When transactions are processed concurrently, more than one transaction may access the same table or row at the same time, and conflicting data may occur.

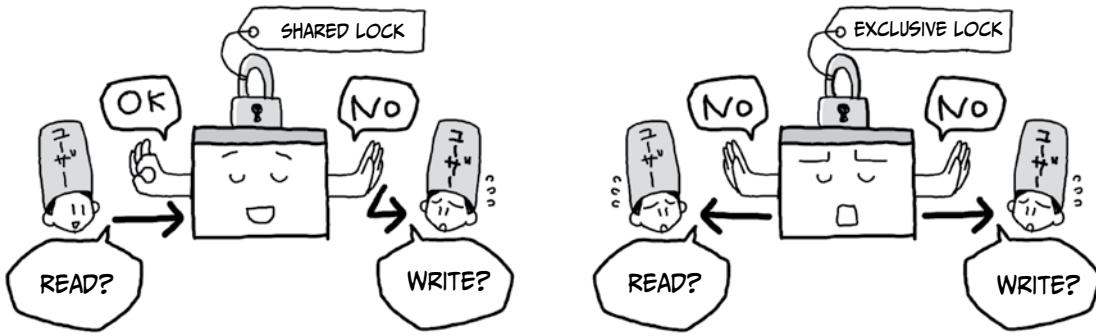
Tables and rows subject to operations in a transaction are referred to as *resources*. In a database, transactions should be able to access the same resource concurrently without creating inconsistencies.



ISOLATION

When two or more concurrent transactions yield the same result as if they were performed at separate times, that order of processing is referred to as *serializable*. The *isolation* property requires the schedule to be serializable and protects against errors.

In order to make the order of processing serializable, you need to have control over transactions that are attempted at the same time. The most commonly used method for this purpose is the lock-based control. A *shared lock* is used when reading data, while an *exclusive lock* is used when writing data.



When a shared lock is in use, another user can apply a shared lock to other transactions, but not an exclusive lock. When an exclusive lock is applied, another user cannot apply a shared lock or an exclusive lock to other transactions. The following summarizes the relationship between a shared lock and an exclusive lock.

CO-EXISTENCE RELATIONSHIP BETWEEN LOCK TYPES

	Shared lock	Exclusive lock
Shared lock	YES	NO
Exclusive lock	NO	NO



QUESTIONS

Do you understand locks? Answer these questions and check your answers on page 167.

Q3

When Andy has applied a shared lock, can Becky apply a shared lock?

Q4

When Andy has applied an exclusive lock, can Becky apply a shared lock?

Q5

When Andy has applied a shared lock, can Becky apply an exclusive lock?

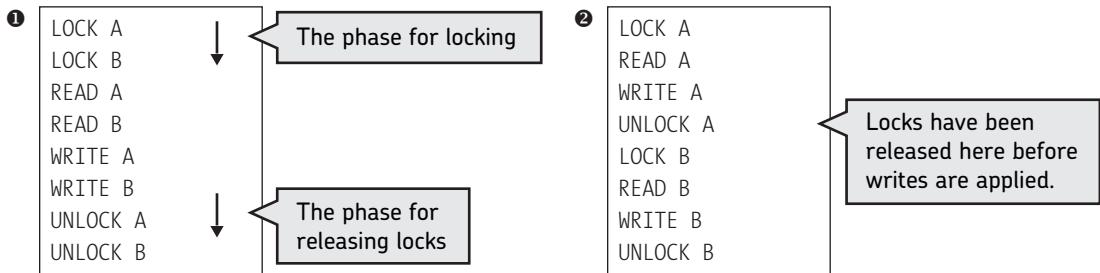
Q6

When Andy has applied an exclusive lock, can Becky apply an exclusive lock?

TWO-PHASE LOCKING

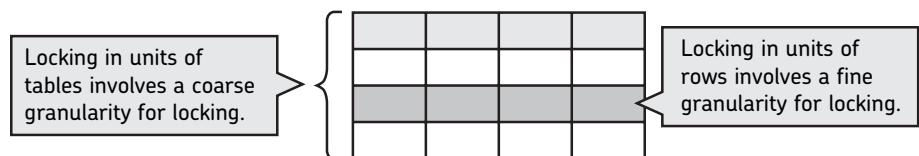
In order to make sure a schedule is serializable, we need to obey specific rules for setting and releasing locks. One of these rules is *two-phase locking*—for each transaction, two phases should be used: one for setting locks and the other for releasing them.

For example, suppose there are resources A and B, both subject to locking. Transaction ① observes the rule of two-phase locking, while transaction ② does not. Serialization can only be achieved when each transaction complies with the rule of two-phase locking.



LOCKING GRANULARITY

There are a number of resources that can be locked. For example, you can lock data in units of tables or units of rows. The extent to which resources are locked is referred to as *granularity*. *Coarse granularity* occurs when many resources are locked at once, and *fine granularity* occurs when few resources are locked.



When granularity is coarse (or high), the number of locks needed per transaction is reduced, making it easier to manage granularity. In turn, this reduces the amount of processing required by the CPU on which the database is operating. On the other hand, as more resources are locked, it tends to take longer to wait for locks used by other transactions to be released. Thus, the number of transactions you can carry out tends to drop when granularity is high.

In contrast, when granularity is fine (or low), a greater number of locks are used in one transaction, resulting in more operations for managing locks. This results in greater processing required by the CPU. However, since fewer resources are locked, you will spend less time waiting for other transactions to release locks. Thus, the number of transactions you can carry out tends to increase.

QUESTIONS

Answer these questions, and check the correct answers on page 168.

Q7

The target resource for locking has been changed from a table to a row. What will happen to the number of transactions you can carry out concurrently?



Q8

The target resource for locking has been changed from a row to a table. What will happen to the number of transactions you can carry out concurrently?

OTHER CONCURRENCY CONTROLS

You can use locking to effectively carry out two or more transactions at the same time. However, using locking comes with the burden of lock management, since *deadlocks*—places where user actions conflict—can occur. Simpler methods for concurrency control can be used when you have a small number of transactions or a high number of read operations. In such cases, the following methods may be used:

Timestamp Control

A label containing the time of access, referred to as a *timestamp*, is assigned to data accessed during a transaction. If another transaction with a later timestamp has already updated the data, the operation will be not permitted. When a read or write operation is not permitted, the transaction is rolled back.

Optimistic Control

This method allows a read operation. When a write operation is attempted, the data is checked to see if any other transactions have occurred. If another transaction has already updated the data, the transaction is rolled back.

LEVELS OF ISOLATION

In a real-world database, you can set the level of transactions that can be processed concurrently. This is referred to as the *isolation level*.

In SQL, the SET TRANSACTION statement can be used to specify the isolation levels of the following transactions:

- READ UNCOMMITTED
- READ COMMITTED
- REPEATABLE READ
- SERIALIZABLE

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

Depending on the isolation level setting, any of the following actions may occur.

	Dirty read	Non-repeatable read	Phantom read
READ UNCOMMITTED	Possible	Possible	Possible
READ COMMITTED	Will not occur	Possible	Possible
REPEATABLE READ	Will not occur	Will not occur	Possible
SERIALIZABLE	Will not occur	Will not occur	Will not occur

- A *dirty read* occurs when transaction 2 reads a row before transaction 1 is committed.
- A *non-repeatable read* occurs when a transaction reads the same data twice and gets a different value.
- A *phantom read* occurs when a transaction searches for rows matching a certain condition but finds the wrong rows due to another transaction's changes.

DURABILITY

A database manages important data, so ensuring security and durability in the case of failure is critical. Security is also important for preventing unauthorized users from writing data and causing inconsistencies.

In a database, you can set permissions for who can access the database or tables in it. Cain avoids dangers to the Kingdom's database by enhancing the database's security.

In a relational database, the GRANT statement is used to grant read and write permissions to users. You can use GRANT statements to grant permission for other users to process tables you have created. Setting permissions is an important task for database operation.

```
GRANT SELECT, UPDATE ON product TO Overseas_Business_Department;
```

This statement grants permission to process data.

You can assign the following privileges (permissions) with SQL statements.

DATABASE PRIVILEGES

Statement	Result
SELECT	Allows user to search for rows in a table.
INSERT	Allows user to insert rows in a table.
UPDATE	Allows user to update rows in a table.
DELETE	Allows user to delete rows in a table.
ALL	Gives user all privileges.

Granting a privilege with WITH GRANT OPTION enables the user to grant privileges to other users. With the statement shown below, the Overseas Business Department can allow other users to search and update the database.

```
GRANT SELECT, UPDATE ON product TO Overseas_Business_Department  
WITH GRANT OPTION;
```

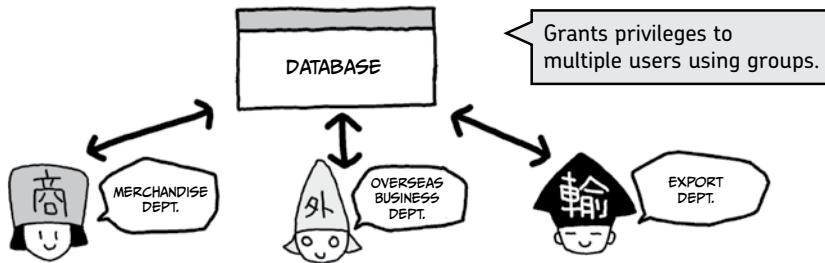
The granted user can grant privileges to other users.

You can also take away a user's privileges. To do this, use the REVOKE statement.

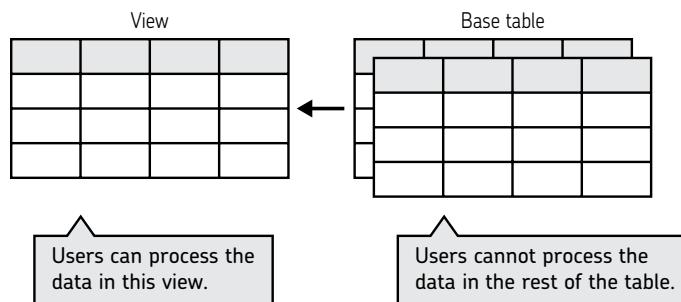
```
REVOKE SELECT, UPDATE ON product FROM  
Overseas_Business_Department;
```

This statement revokes the user's privileges.

Some database products can group a number of privileges and grant them to multiple users at once. Grouping makes privilege management easier.



Using views, as described on page 117, enables even more controlled management for enhanced security. First, extract part of a base table to create a view. Setting a privilege for this view means the privilege is also set on the selected portion of data in the view.



QUESTIONS



Try these questions on durability. The answers are on page 168.

Q9

Write an SQL statement that allows the Export Department to search for data in the Product Table.

Q10

Create an SQL statement to revoke the Overseas Business Department's privilege to delete data from the Product Table.

Q11

Privileges were set as follows on a Product Table created by the administrator. Enter a YES or NO in each cell of the table below to indicate the presence or absence of the privilege for each department, respectively.

```
GRANT ALL product TO Overseas_Business_Department;
GRANT INSERT, DELETE ON product TO Merchandise_Department;
GRANT UPDATE, DELETE ON product TO Export_Department;
```

	Search	Insert	Update	Delete
Overseas Business Dept.				
Merchandise Dept.				
Export Dept.				

WHEN DISASTER STRIKES



A database needs to have a mechanism that can protect data in the system in the event of a failure. To ensure durability of transactions, it is mandatory that no failure can create incorrect or faulty data. To protect itself from failure, a database performs various operations, which include creating backup copies and transaction logs.

TYPES OF FAILURES

Database failure can occur under various circumstances. Possible types of failure include the following:

- Transaction failure
- System failure
- Media failure

Transaction failure occurs when a transaction cannot be completed due to an error in the transaction itself. The transaction is rolled back when this failure occurs.

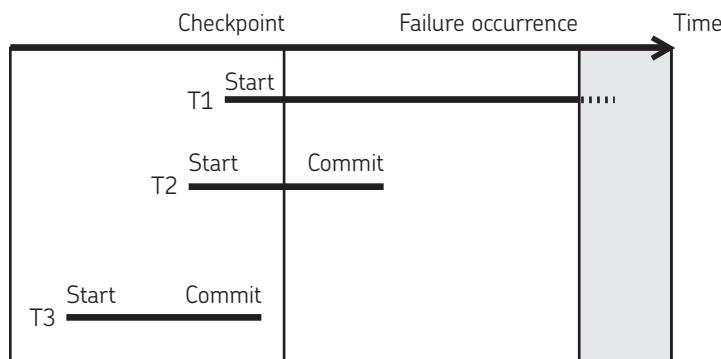
System failure occurs when the system goes down due to a power failure or other such disruption. In the case of a system failure, disaster recovery takes place after you reboot the system. Generally, transactions that have not yet been committed at the time of failure are rolled back, and those that have already been committed when a failure occurs are rolled forward.

Media failure occurs when the hard disk that contains the database is damaged. In the case of a media failure, disaster recovery is carried out using backup files. Transactions committed after the backup files were created are rolled forward.

CHECKPOINTS

In order to improve the efficiency of a write operation in a database, a *buffer* (a segment of memory used to temporarily hold data) is often used to write data in the short term. The contents of the buffer and the database are synchronized, and then a *checkpoint* is written. When the database writes a checkpoint, it doesn't have to perform any failure recovery for transactions that were committed before the checkpoint. Transactions that weren't committed before the checkpoint must be recovered.

Now, suppose the transactions shown below are being performed at the time a system failure occurs. Which transactions should be rolled back? Which ones should be rolled forward?





QUESTIONS

Try these questions based on the table on the previous page. The answers are on page 168.

Q12

How should T1 be processed?

Q13

How should T2 be processed?

Q14

How should T3 be processed?

In case of database failure, the recovery mechanisms described above will protect the database against inconsistency. That is why you can be reassured of database integrity when you use it.

INDEXES

A database manages massive amounts of data, so searching for specific data can be very time consuming. But you can use indexes to speed up searches!

Product code	Product name	Unit price	District
101	Melon	800G	South Sea
102	Strawberry	150G	Middle
103	Apple	120G	North Sea
104	Lemon	200G	South Sea
201	Chestnut	100G	North Sea
202	Persimmon	160G	Middle
301	Peach	130G	South Sea
302	Kiwi	200G	South Sea

It is very time consuming to search for each item row by row.

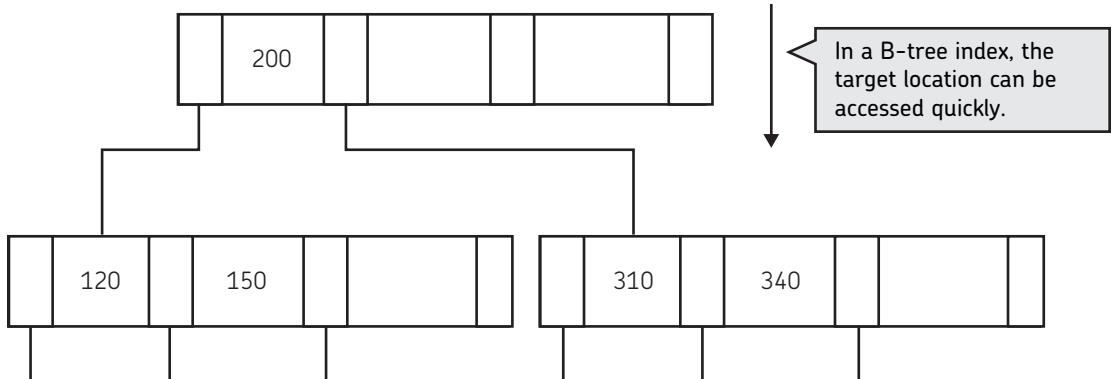
An *index* is a tool that allows you to speedily access the location of the target data. When looking for some data in a large database, searching with indexes promises fast results.

Product code	Product name	Unit price	District
101	Melon	800G	South Sea
102	Strawberry	150G	Middle
103	Apple	120G	North Sea
104	Lemon	200G	South Sea
201	Chestnut	100G	North Sea
202	Persimmon	160G	Middle
301	Peach	130G	South Sea
302	Kiwi	200G	South Sea

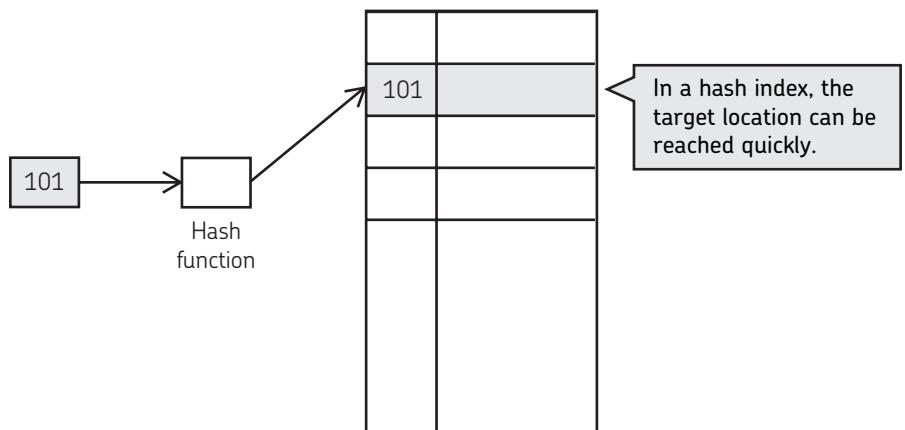
The target data location can be accessed quickly by using its index.

Index

Indexing methods include B-tree and hash methods. A *B-tree index* is composed of parent nodes and child nodes, which can have further child nodes. The nodes are arranged in sorted order. Each parent contains information about the minimum and maximum values contained by all of its children. This allows the database to navigate quickly to the desired location, skipping entire sections of the tree that cannot possibly contain the desired value.



The *hash index* method finds the location of target data by applying a hash function to the key value of the data. The hash acts as a unique fingerprint for a value. The hash index method can perform specific full-match searches, such as a search for *product code 101*. However, it is not designed to search effectively for comparative conditions like *product codes no less than 101* or for fuzzy references like *products with names ending in n*.



In some cases, using an index may not speed up the search—using an index doesn't save time unless you are looking for only a small portion of the data. Additionally, there are cases where indexes are recreated every time data is updated, resulting in slower processing of an update operation.



QUESTIONS

Try these questions on indexing. The answers are on page 168.

Q15

Which index would be more powerful in a search with an equal sign, a B-tree or hash index?

Q16

Which index would be more powerful in a search with an inequality sign, a B-tree or hash index?

OPTIMIZING A QUERY

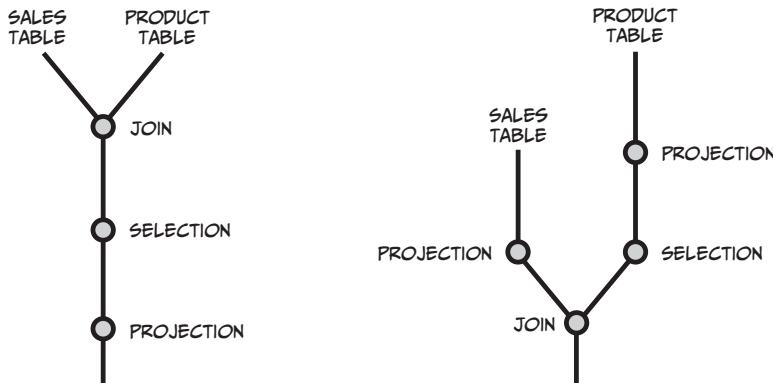
When you query a database, the database analyzes the SQL query and considers whether to use an index so it can process the query more quickly. Let's examine the procedure for processing a query.

The database can decide on an optimal order to process a query. Most queries can be processed in several orders with the same results, but with possibly different speeds. For example, suppose there is a query to extract dates of sale and product names for products with a unit price greater than 200G. This query can be seen as consisting of the following steps.

```
SELECT date, product_name
FROM product, sales
WHERE unit_price>=200
AND product.product_code = sales.product_code;
```

1. Join the Product Table and the Sales Table.
2. Select products whose unit price is greater than 200G.
3. Extract columns of dates and product names.

For example, the figure on the left below shows the query processed in order from 1 to 3. The figure on the right shows the query processed in order from 3 to 1. Either way, the queries are equivalent.



However, when processed from 1 to 3, the same query would generally require a longer processing time, because when the first join is performed, an intermediate table with many rows may be created. On the other hand, the procedure from 3 to 1 requires a shorter processing time, since selection and projection happen first, trimming unwanted data as soon as possible. Thus, the same query may require a different processing time, depending on the order in which projection, selection, and join are performed.

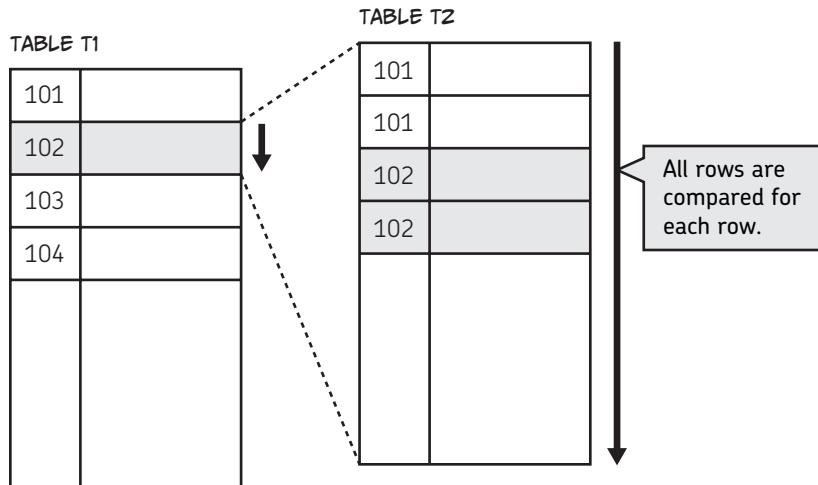
Generally, the database should use the following rules to find the best querying order:

- Execute selection first to reduce the number of rows.
- Execute projection first to reduce the number of columns irrelevant to the result.
- Execute join later.

There are different techniques for executing projection, selection, and join, respectively. For selection, you can use either a full-match search or an index-based search. For join, the following methods are available.

NESTED LOOP

The *nested loop* method compares one row in a table to several rows in another table (see the figure below). For example, one of the values in a row in Table T1 is used to find matching rows in Table T2. If the values are the same, then a joined row is created.



SORT MERGE

The *sort merge* method sorts and then merges rows in multiple tables (see the figure below). First, all or part of Tables T1 and T2 are sorted. Then they are compared starting with the top row, and a joined row is created whenever the same value is found. Since they have already been sorted, processing only needs to be done in one direction, so it will take less time. You should be aware, however, of the time needed for the initial sorting.

TABLE T1

101	
102	
103	
104	

TABLE T2

101	
101	
102	
102	

First sort,
then compare.

HASH

A *hash* divides one of the tables using a hash function and then merges it with a row in another table that has the same hash value. This method effectively selects the row to join.

TABLE T1

TABLE T2

Hash
function

Hash
function

OPTIMIZER

When a query is processed, these different techniques are examined for optimal performance. In a database, the function in charge of optimization of queries is referred to as the *optimizer*. There are two common types.

RULE-BASED PROCESSING

Certain rules are established before any operations are performed. For example, some operations can be combined or reordered in much the same way an algebraic equation can be manipulated and still mean the same thing. The optimizer tries to find the most efficient way to process the query that gives the same results.

COST-BASED PROCESSING

This method tries to estimate the cost of processing the query, based on statistics that the database maintains. Cost-based processing is sometimes more flexible than rule-based processing, but it requires periodical updates of the database's statistics. Managing and analyzing these statistics requires a lot of time.

SUMMARY



- You can set user privileges for a database.
- Locking ensures consistency when a database has multiple users.
- Indexing enables fast searches.
- A database has disaster recovery functions.

ANSWERS

Q1

COMMIT;

Q2

ROLLBACK;

Q3 Yes

Q4 No

Q5 No

Q6 No

Q7 Increases

Q8 Decreases

Q9

```
GRANT SELECT ON product TO Export_Department;
```

Q10

```
REVOKE DELETE ON product FROM Overseas_Business_Department;
```

Q11

	Search	Insert	Update	Delete
Overseas Business Dept.	YES	YES	YES	YES
Merchandise Dept.	NO	YES	NO	YES
Export Dept.	NO	NO	YES	YES

Q12

A rollback is performed since it is not committed at the time of the failure occurrence.

Q13

A roll forward is performed since it has been committed at the time of the failure occurrence.

Q14

No recovery operation is needed since it has been committed at the time of checkpoint.

Q15

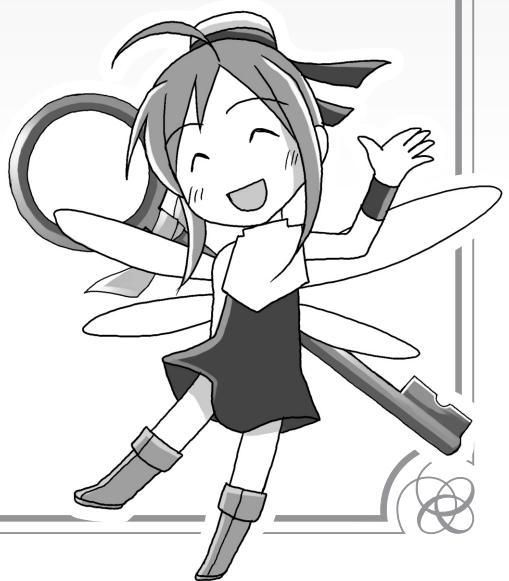
Hash

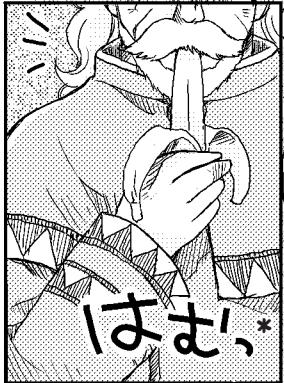
Q16

B-tree

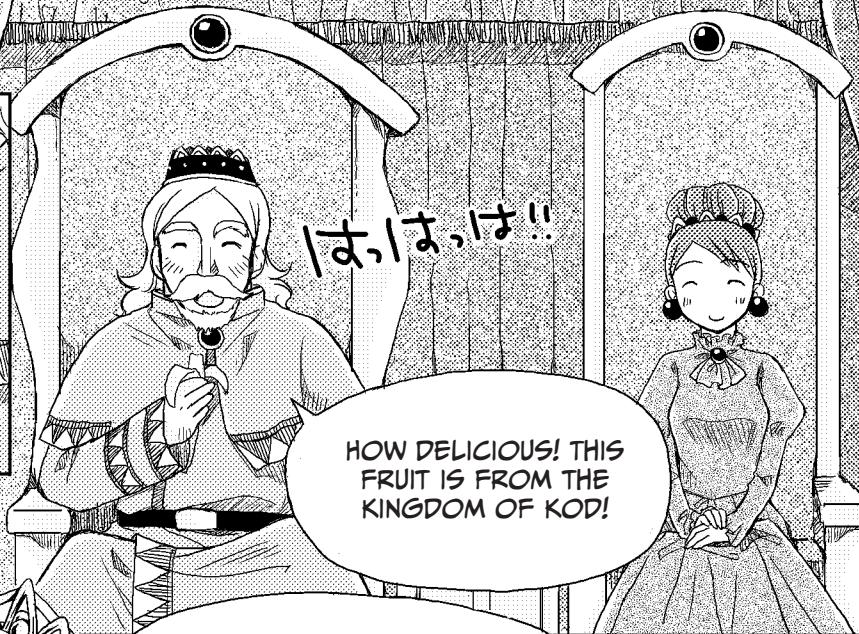
6

DATABASES ARE EVERYWHERE!





* OM NOM NOM!



FATHER!

NO, NO!

YES? WHAT'S THE MATTER? DO YOU WANT A BANANA, TOO, RURUNA?

FATHER,
MUNCHING
ON FRUIT IS
ALL YOU HAVE
DONE SINCE
YOU HAVE
RETURNED.

FORGIVE
ME! NO
OTHER FRUIT
COMPARES!

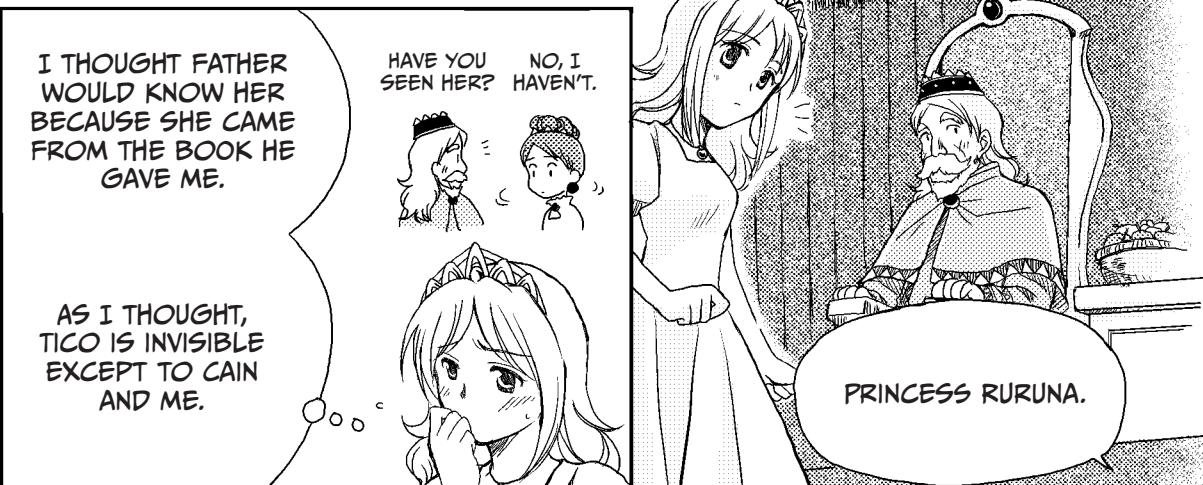
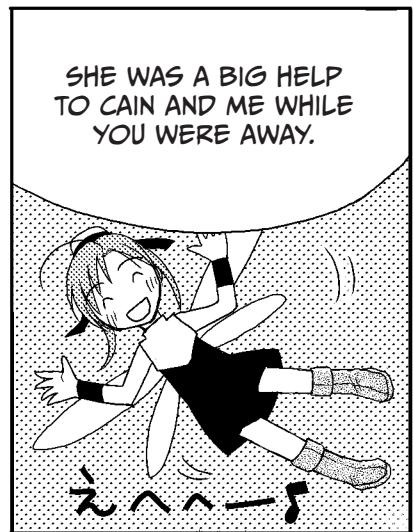
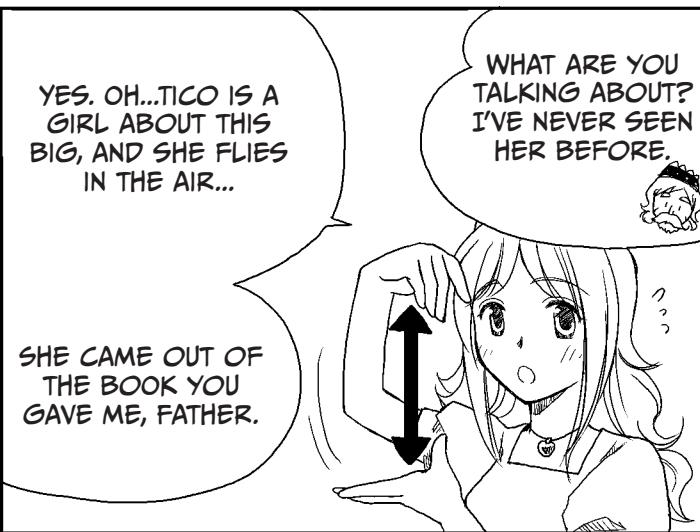
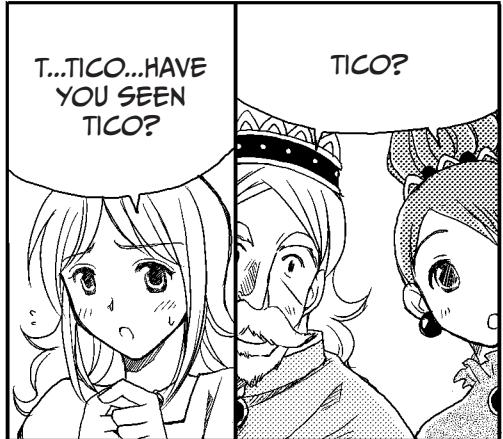
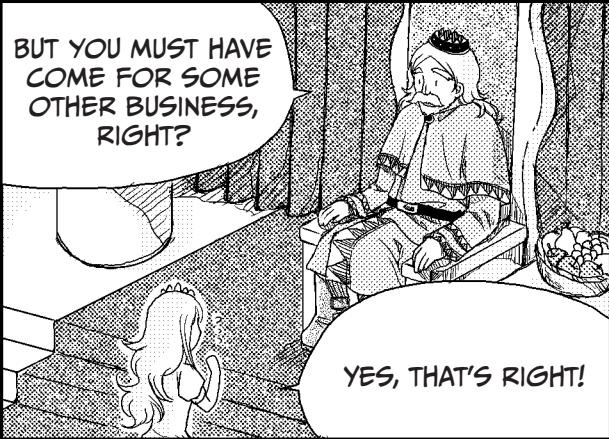
BUT I ADMIT, RURUNA
KEPT A TIGHT REIN
WHILE I WAS AWAY.

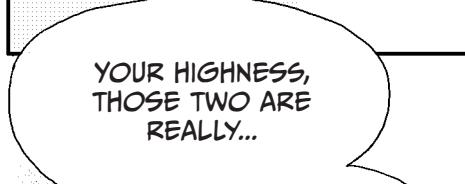
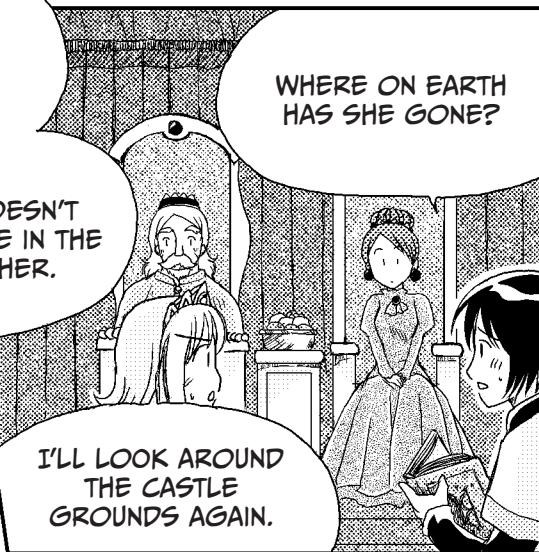
LOOK AT HOW
PROSPEROUS
THE KINGDOM
OF KOD IS!

REALLY, A
DATABASE IS
A CONVENIENT
THING!

はっはっは

木木木木

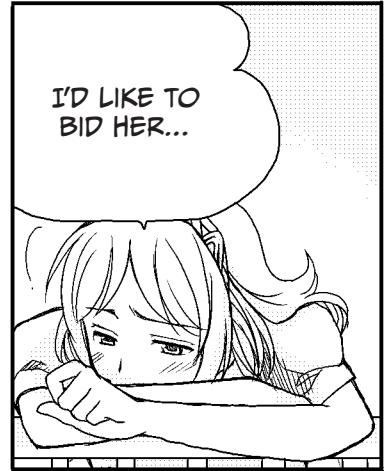




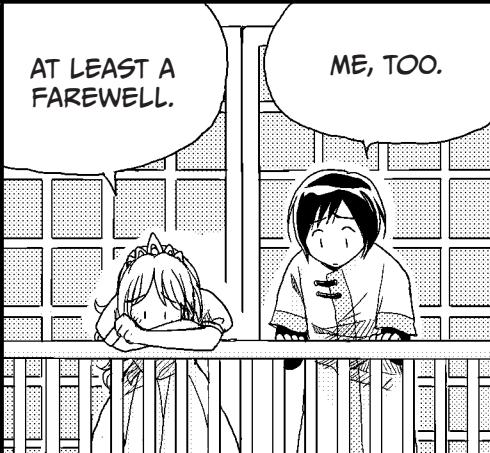


SHE'S NOWHERE!

TICO, WHY HAVE YOU
DISAPPEARED ALL OF
A SUDDEN?



I'D LIKE TO
BID HER...



AT LEAST A
FAREWELL.

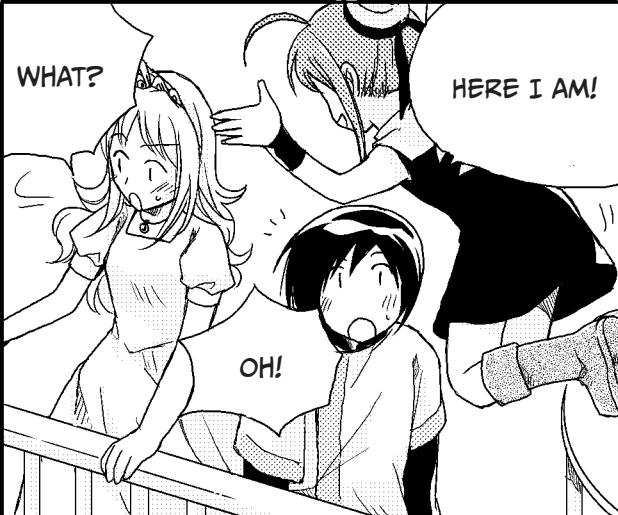
ME, TOO.



I WONDER
WHO ARE YOU
LOOKING FOR?



!!!



WHAT?

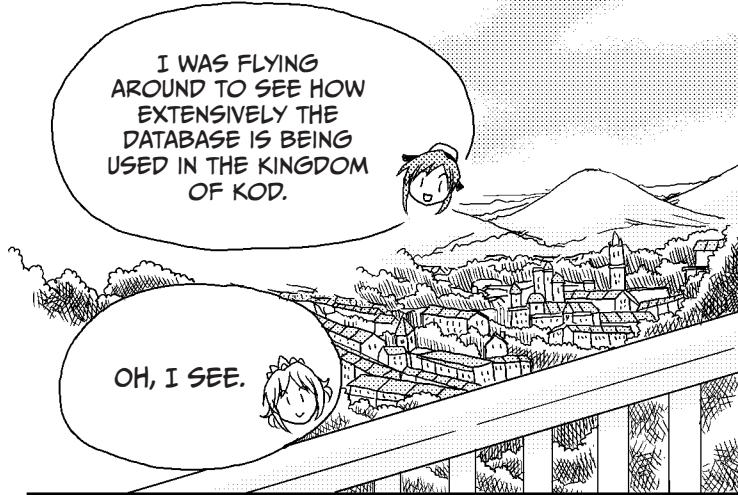
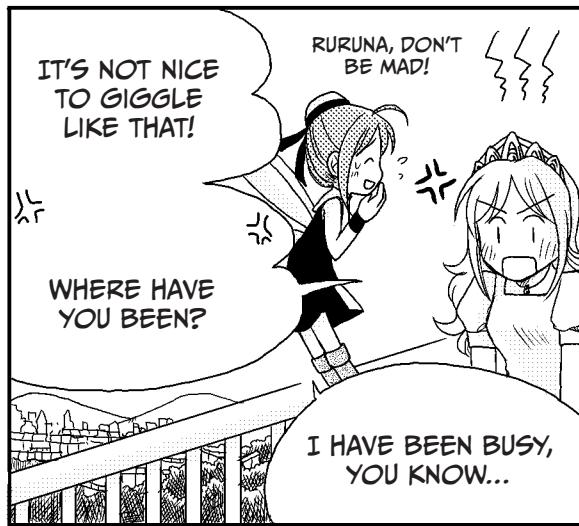
HERE I AM!

OH!

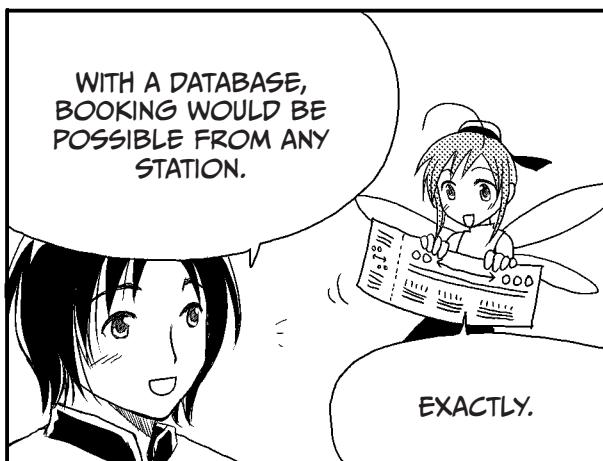
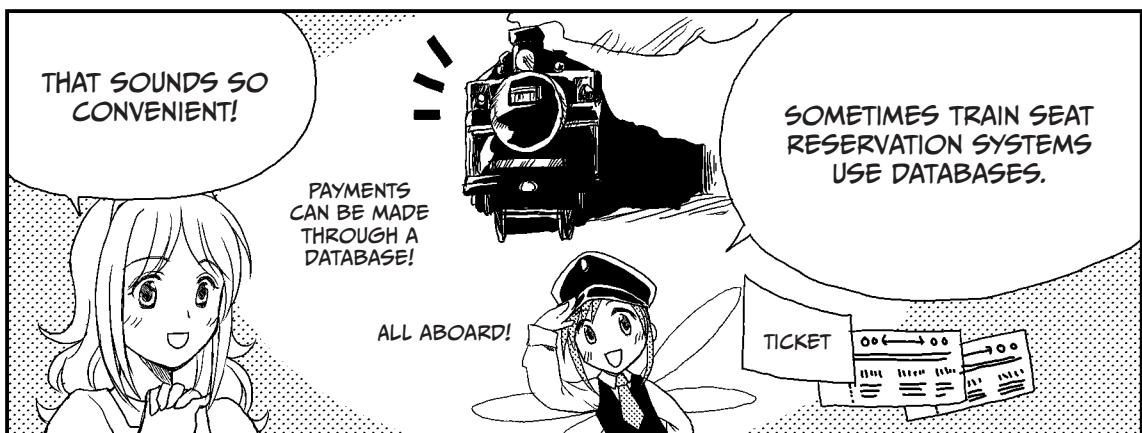
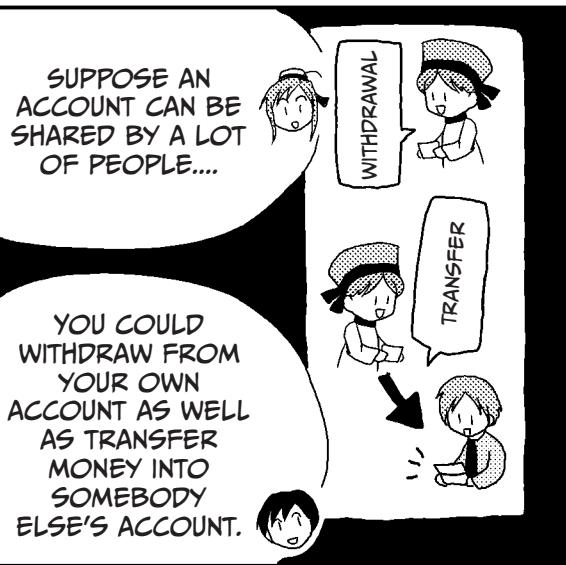
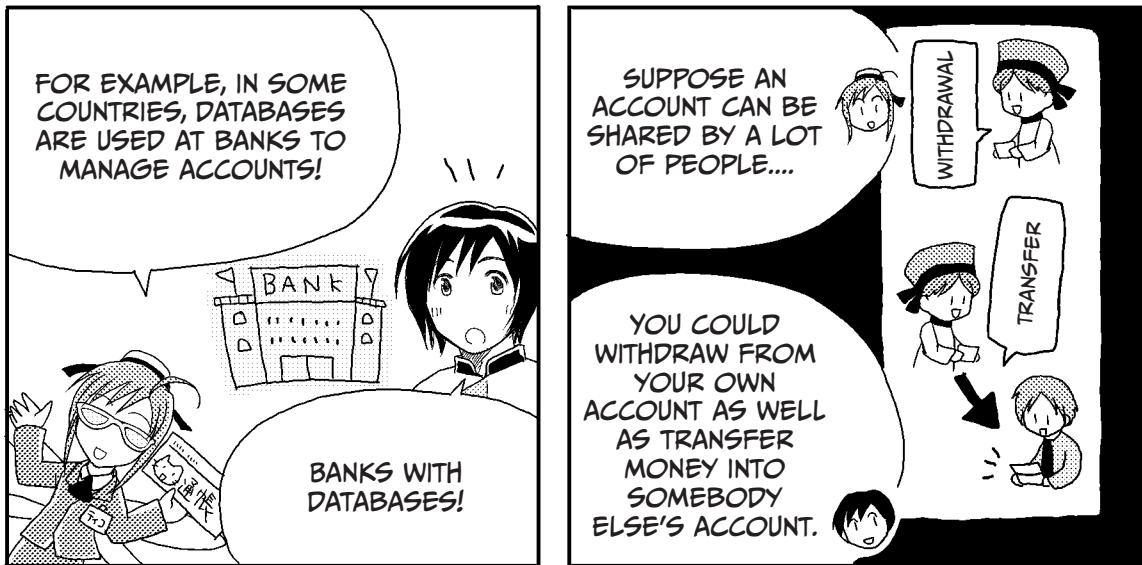


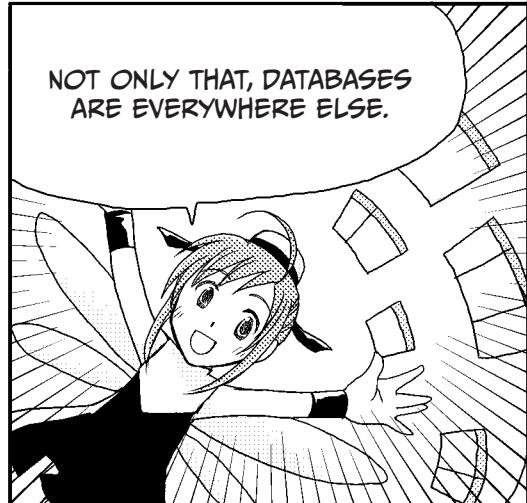
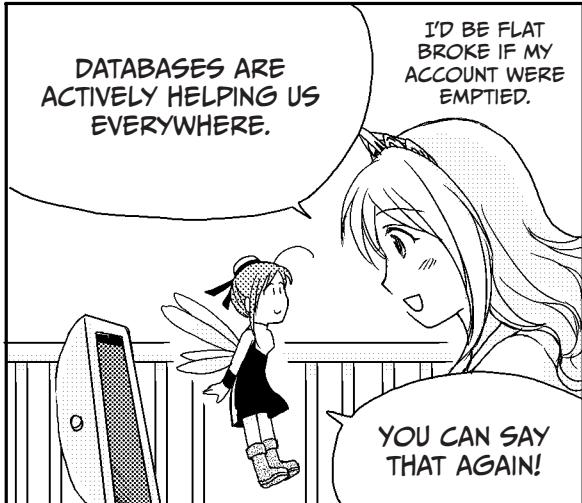
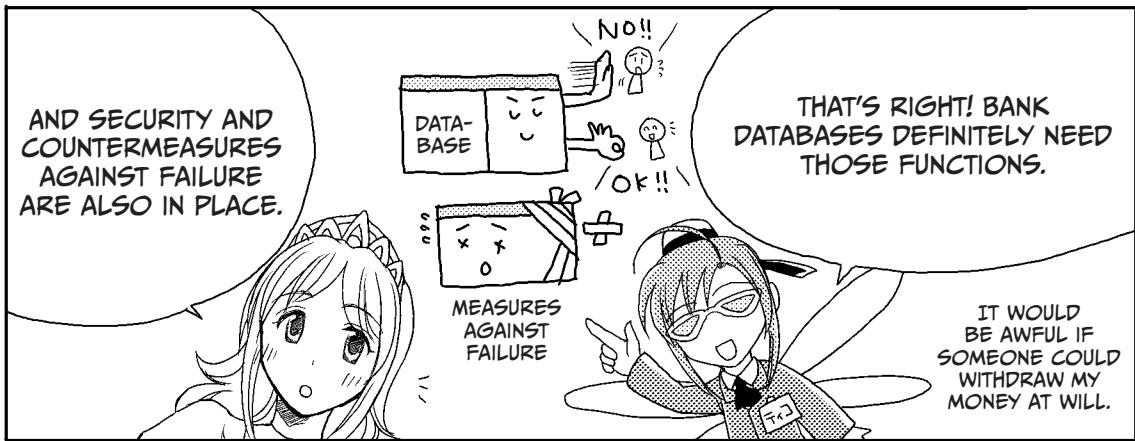
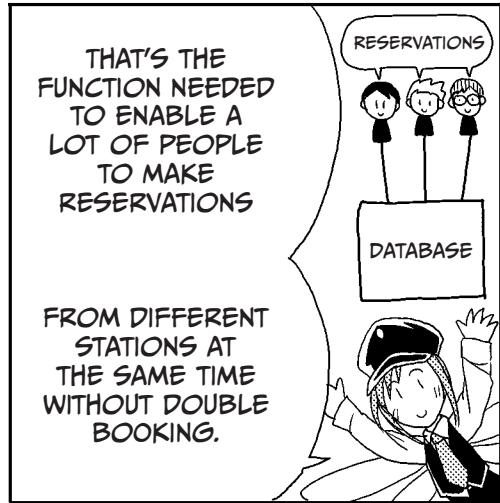
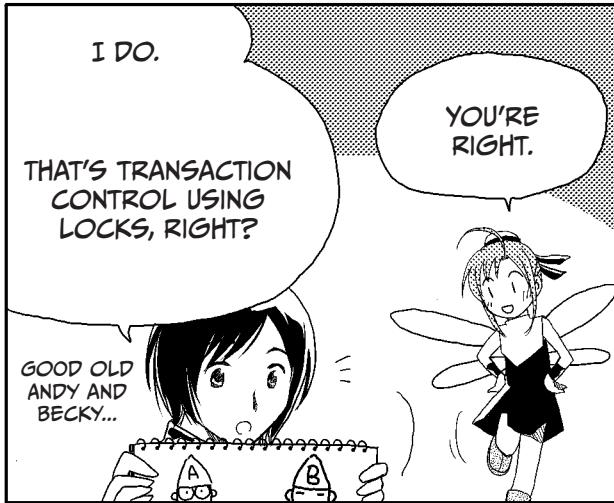
TICO!

GIGGLING

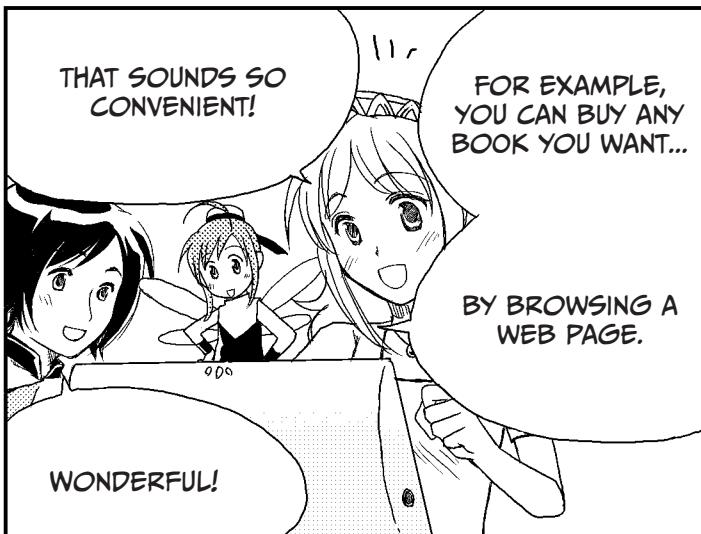
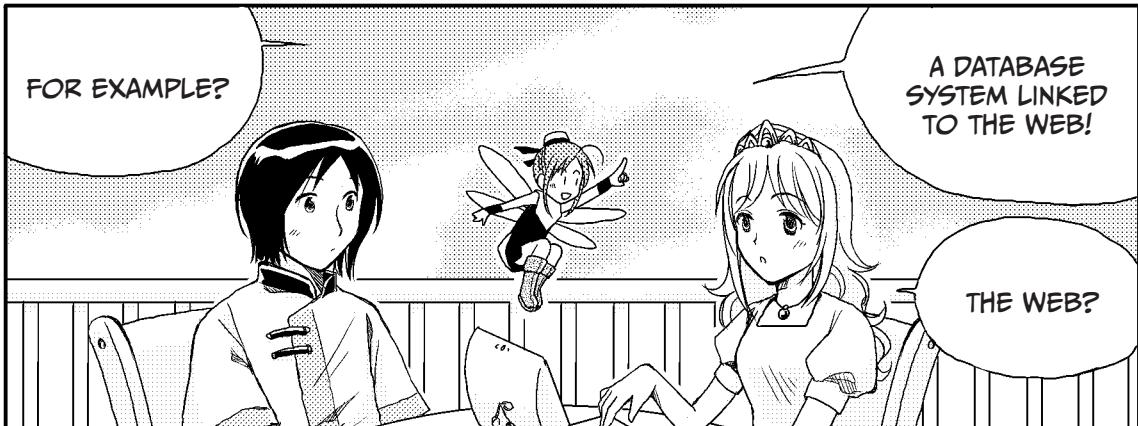


DATABASES IN USE





DATABASES AND THE WEB



WHEN YOU
LOOK FOR A
CERTAIN BOOK
YOU HAVE IN
MIND,

ほん、

Book
search>>>

TICO SEARCH

CATEGORIES

KEYWORD

+

ENTER A KEYWORD IN
A WEB BROWSER.



TICO
SEARCH?

WHAT CATEGORY
OF BOOKS ARE YOU
LOOKING FOR?

WELL, LET ME SEE...THAT
WOULD BE "FRUIT," I GUESS.

THEN TYPE
FRUIT IN THIS
KEYWORD
FIELD.

KEYWORD

Fruit

THIS KEYWORD IS SENT OUT
AS AN HTTP REQUEST.

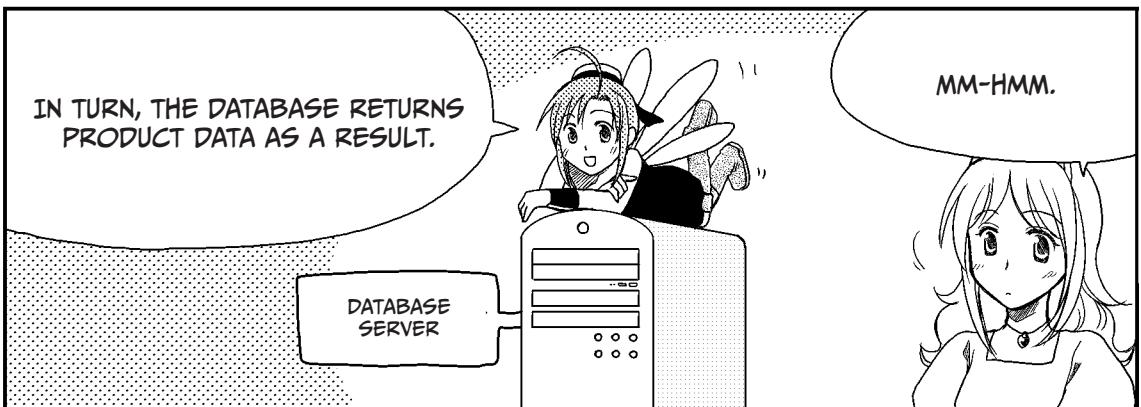
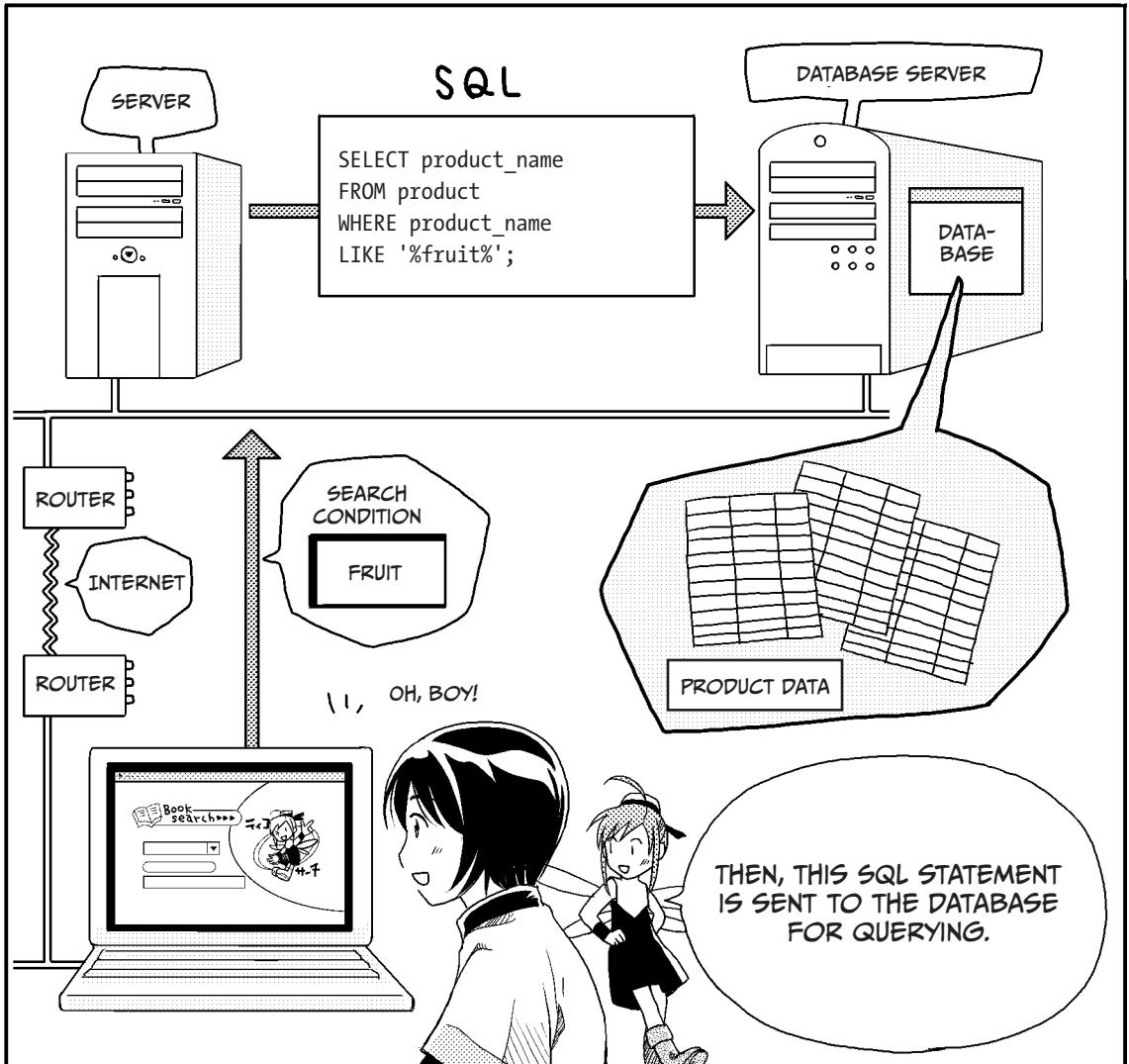
AN SQL STATEMENT IS
MADE ON THE SERVER,
I SUPPOSE.

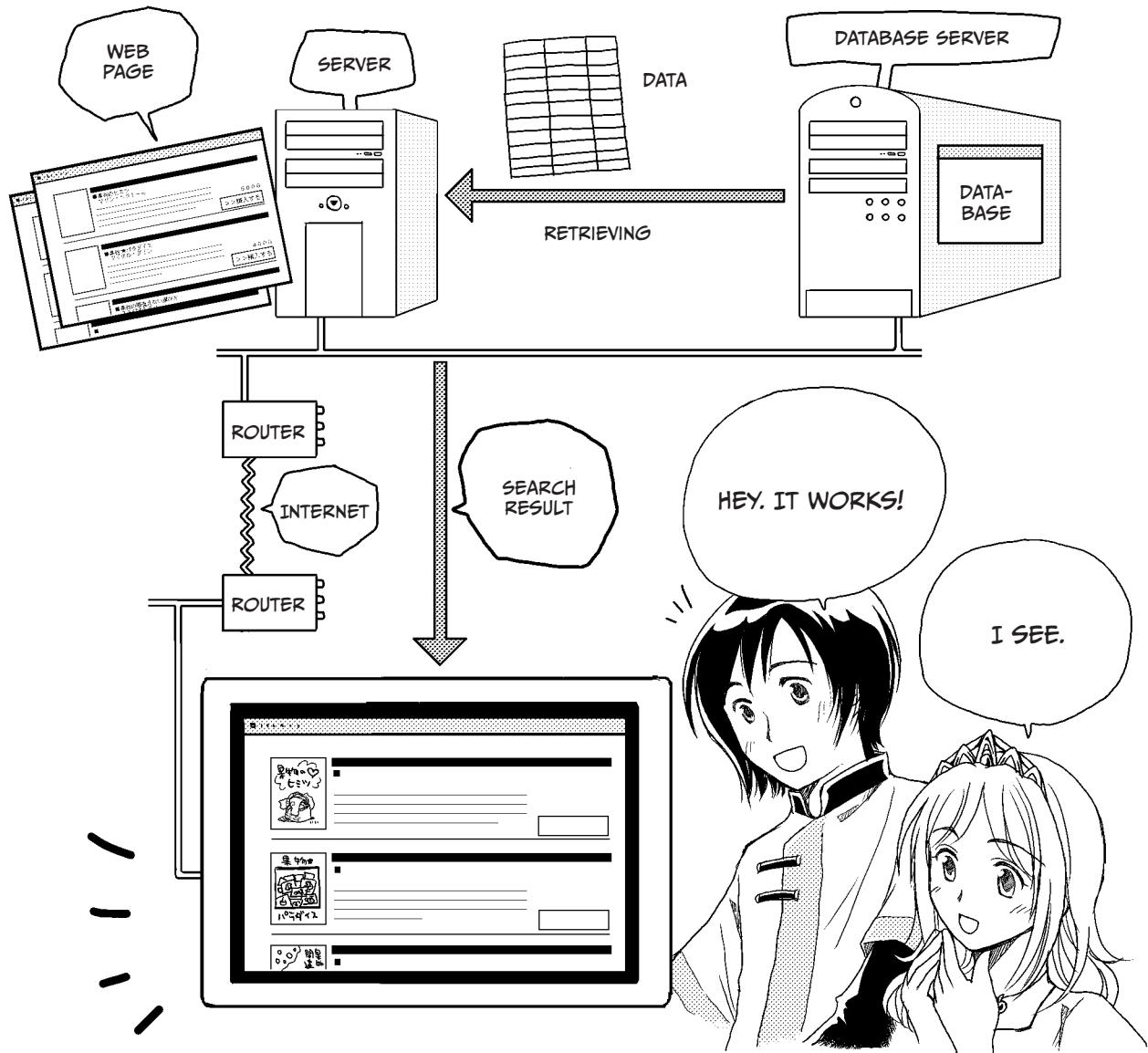
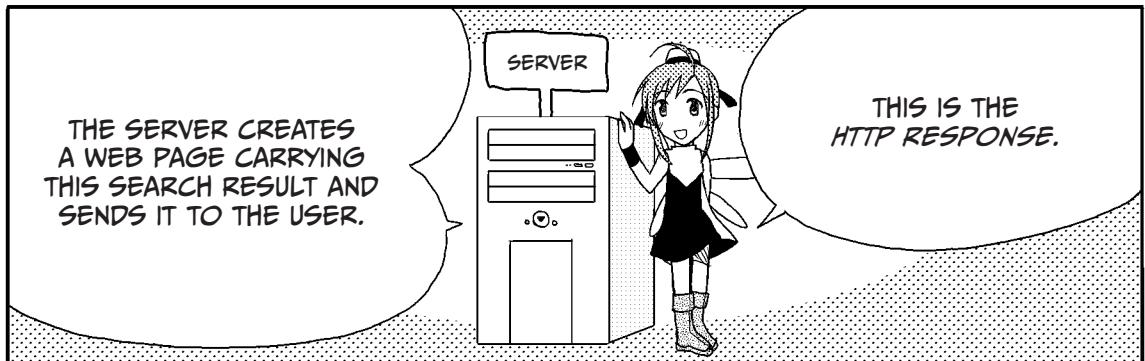
A COMPUTER THAT
RECEIVES A REQUEST
AND PROCESSES IT IS A
SERVER.

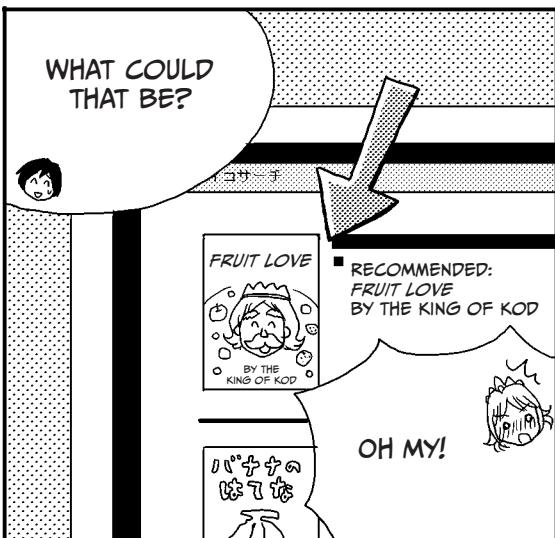
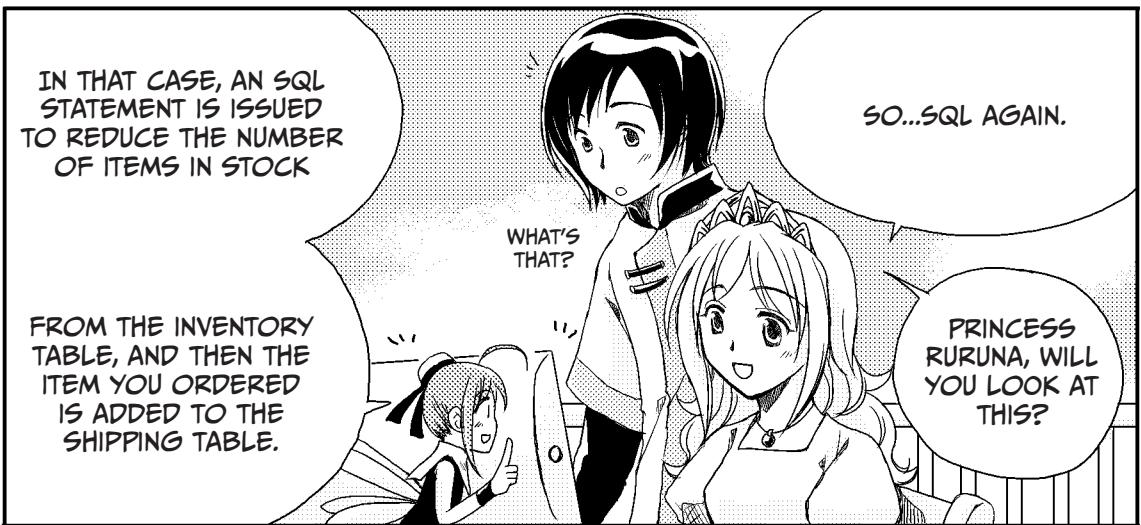
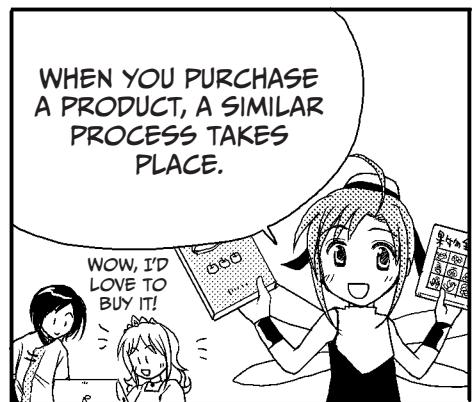
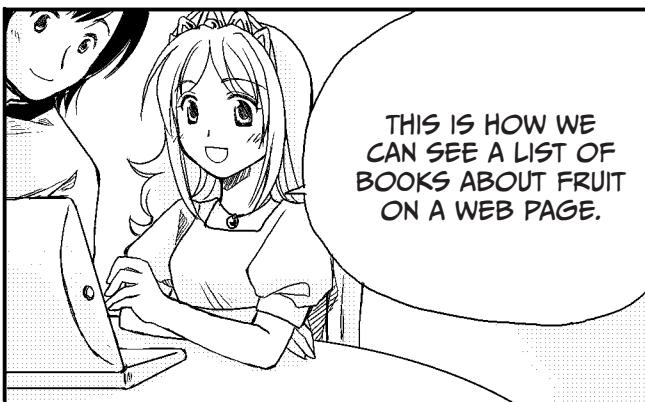
CATEGORIES

KEYWORD

Fruit







I IMAGINE A LOT OF CUSTOMERS ACCESS A WEB BOOKSHOP AT THE SAME TIME.

EVEN IF LOCKS AND SECURITY FUNCTIONS GIVE THE DATABASE FULL PROTECTION,

IT MUST BE A LOT OF PROCESSING.

IN THAT CASE, THE BURDEN OF PROCESSING IS SHARED AMONG A NUMBER OF SERVERS.

YOU MEAN MORE THAN ONE SERVER IS INVOLVED?

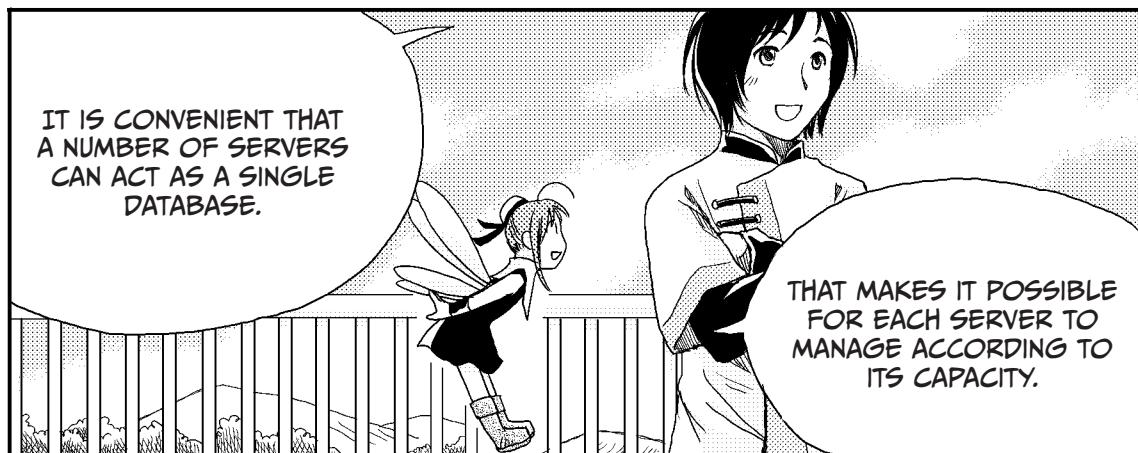
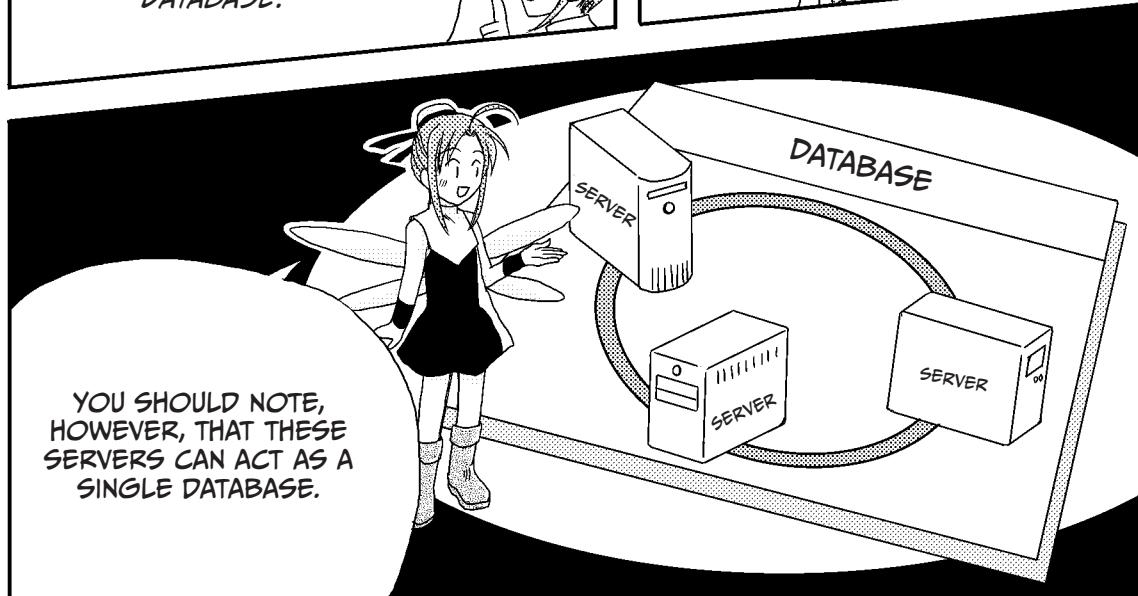
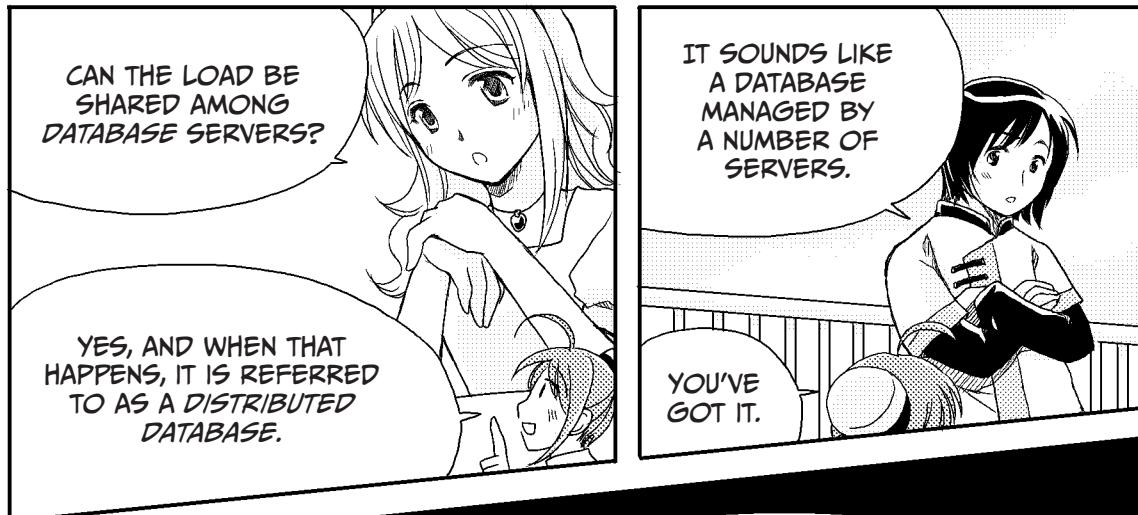
YES, THE LOAD IS DISTRIBUTED AMONG DIFFERENT SERVERS, LIKE A WEB SERVER AND AN APPLICATION SERVER.

A WEB SERVER IS A SERVER IN CHARGE OF CREATING A WEB PAGE, ISN'T IT?

YEP! AND AN APPLICATION SERVER IS IN CHARGE OF COMPOSING SQL STATEMENTS, AMONG OTHER THINGS.

SELECT _____
FROM _____
WHERE _____

DISTRIBUTED DATABASES



MANY SERVERS PROVIDE EXTRA PROTECTION AGAINST FAILURE, TOO!

Knock Out!!

TI-CO!
TI-CO!

THAT MEANS THE ENTIRE DATABASE SYSTEM WON'T GO DOWN, EVEN IF FAILURE OCCURS ON SOME SERVERS IN THE SYSTEM.

DOWN FOR THE COUNT

FAILURE

OH MY!

BUT KEEP THIS IN MIND:
IT TAKES SOME CARE TO
HANDLE YOUR DATABASE IN
THIS WAY.

FOR EXAMPLE?

WHEN A TRANSACTION IS COMMITTED, YOU MUST ENSURE CONSISTENCY ACROSS YOUR DISTRIBUTED DATABASE.

ALSO, FOR INSTANCE, ALL SERVERS MUST BE UPDATED PROPERLY IN CASE ANY PROBLEM OCCURS ON THE NETWORK.

STORED PROCEDURES AND TRIGGERS

A NETWORK IS A MUST IN ANY ENVIRONMENT WHERE A SET OF SERVERS IS USED.

RIGHT! THAT'S WHERE STORED PROCEDURES ARE USEFUL;

THEY ARE SOMETIMES CREATED TO HELP REDUCE THE BURDEN ON THE NETWORK.

STORED...?

AHA!

DOESN'T STORE MEAN PUT INTO MEMORY, IN OTHER WORDS?

RIGHT!

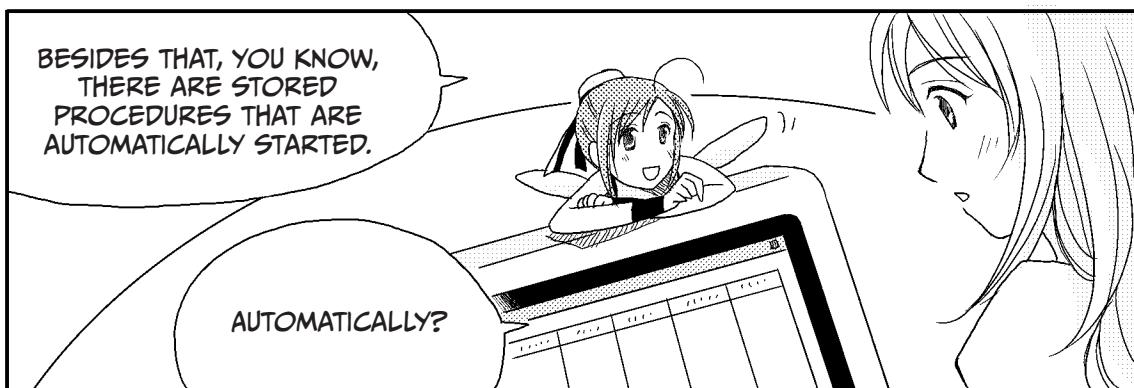
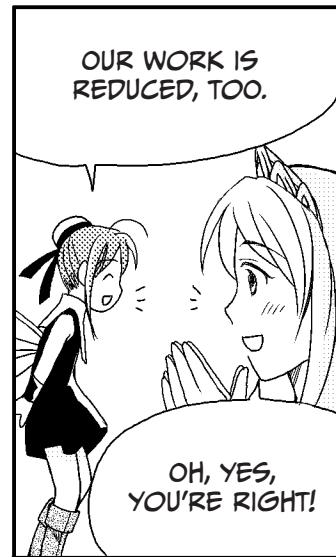
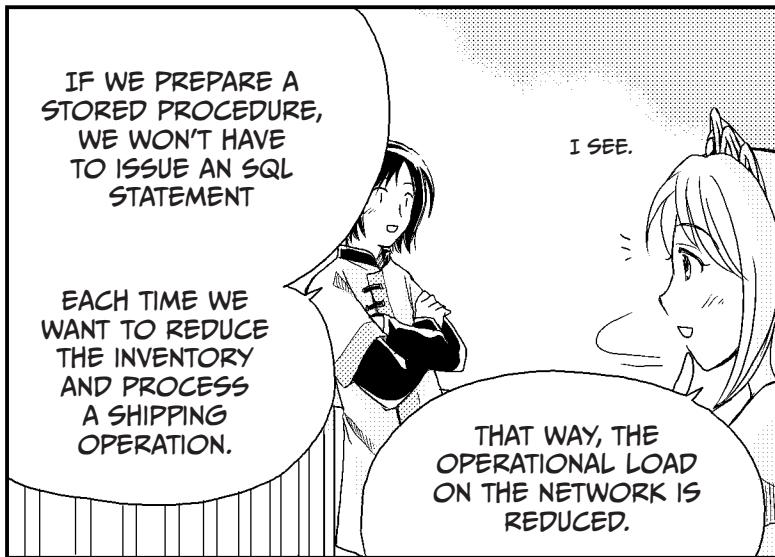
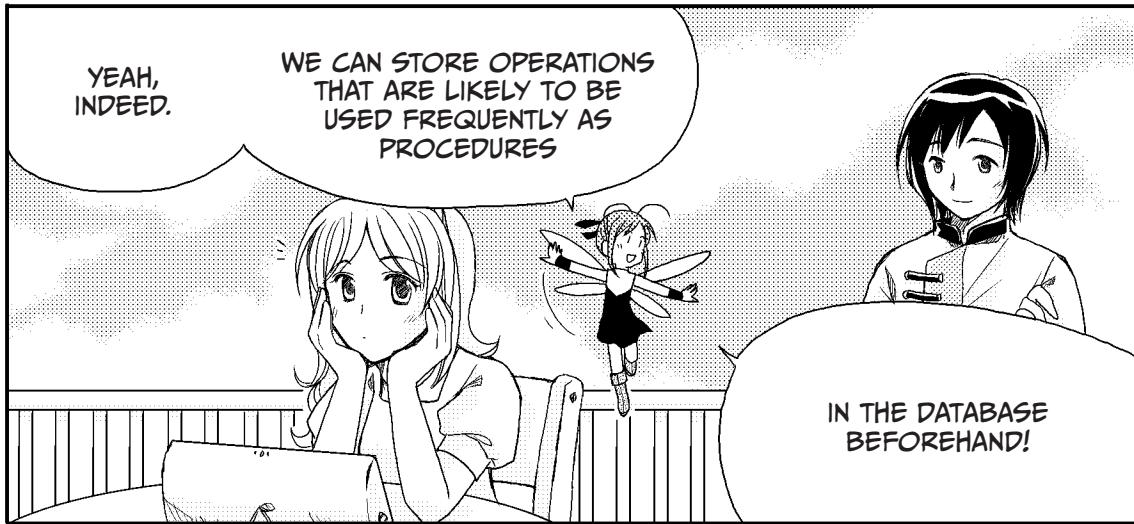
IN ORDER TO REDUCE THE BURDEN ON THE NETWORK, FREQUENTLY USED OPERATIONS CAN BE STORED IN DATABASES.

FREQUENTLY USED OPERATIONS, YOU SAY...WHAT KIND OF OPERATIONS ARE THEY, I WONDER?

WELL, SINCE WE WERE TALKING ABOUT OPERATIONS FOR BUYING A BOOK, SUBTRACTING FROM THE IN-STOCK COUNT IN THE INVENTORY TABLE AND ADDING DATA TO THE SHIPPING TABLE—

LET'S SEE...

AREN'T THOSE TYPICAL OPERATIONS?

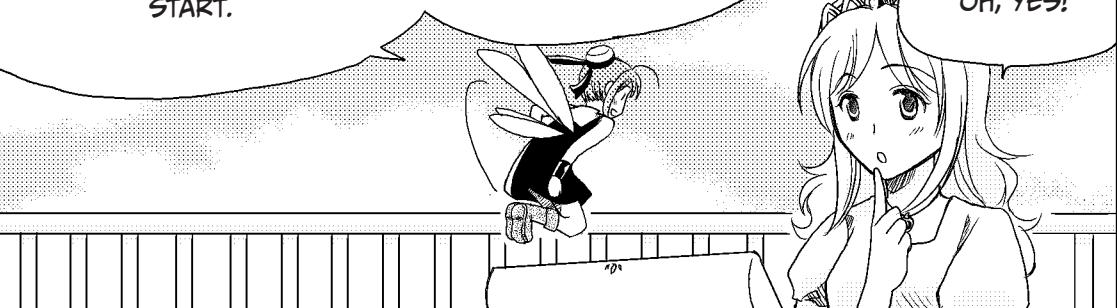


WHEN DATA IS UPDATED,
FOR EXAMPLE, A
STORED PROCEDURE
CAN AUTOMATICALLY
START.

IT'S CALLED A
TRIGGER.

TRIGGER...

OH, YES!



BECAUSE IT DOES
WHAT A TRIGGER ON
A GUN DOES!

IT WOULD BE
CONVENIENT,
INDEED, IF PLACING
AN ORDER AND
UPDATING THE
DATABASE

WHY AM I ALSO
IN THIS OUTFIT?

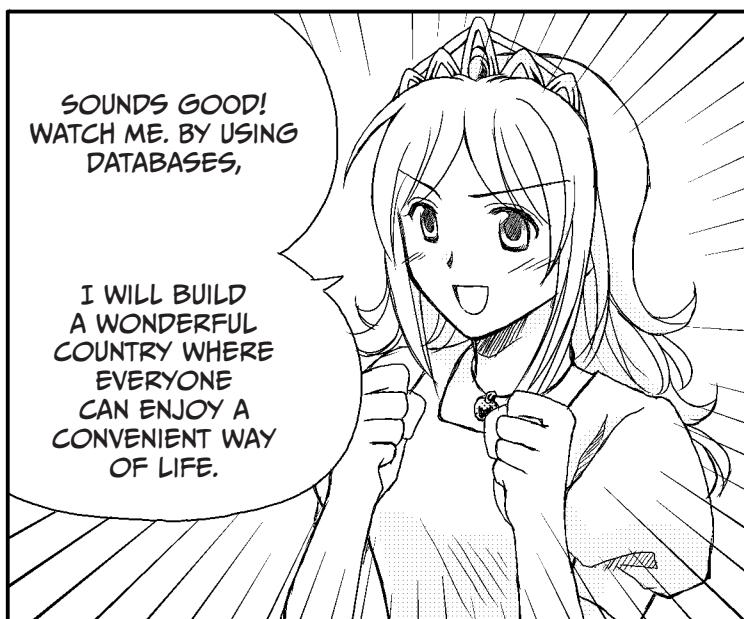
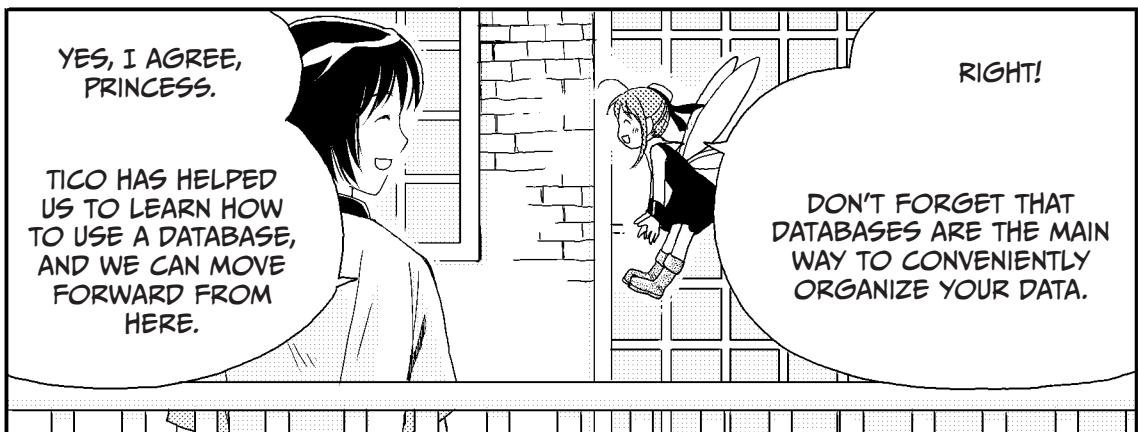
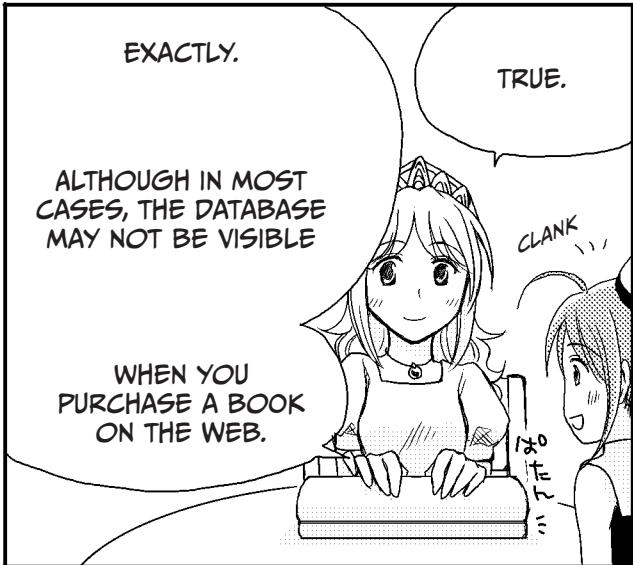
AUTOMATICALLY
LAUNCHED AN
OPERATION TO
REDUCE THE
INVENTORY AND
ARRANGE FOR
SHIPPING.

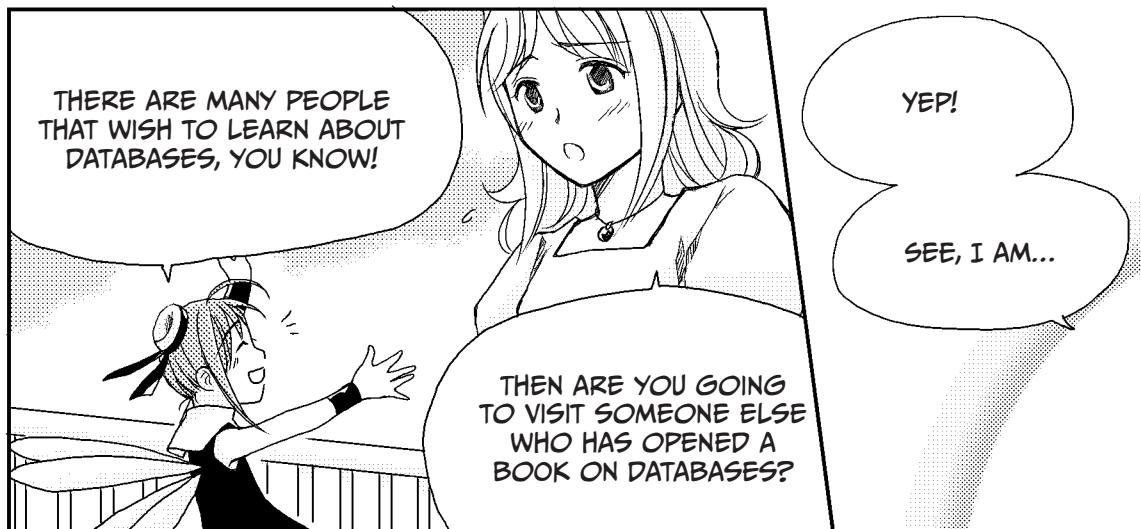
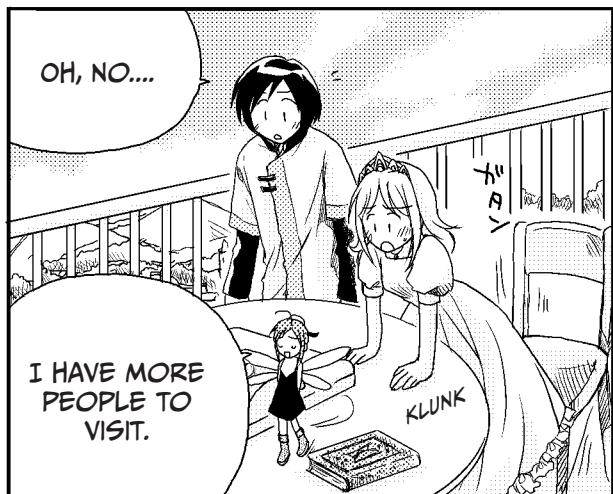
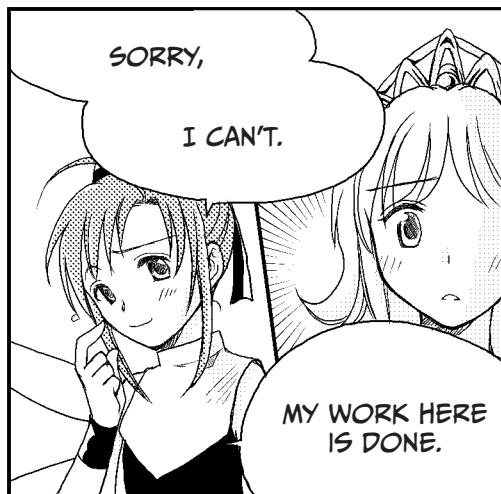
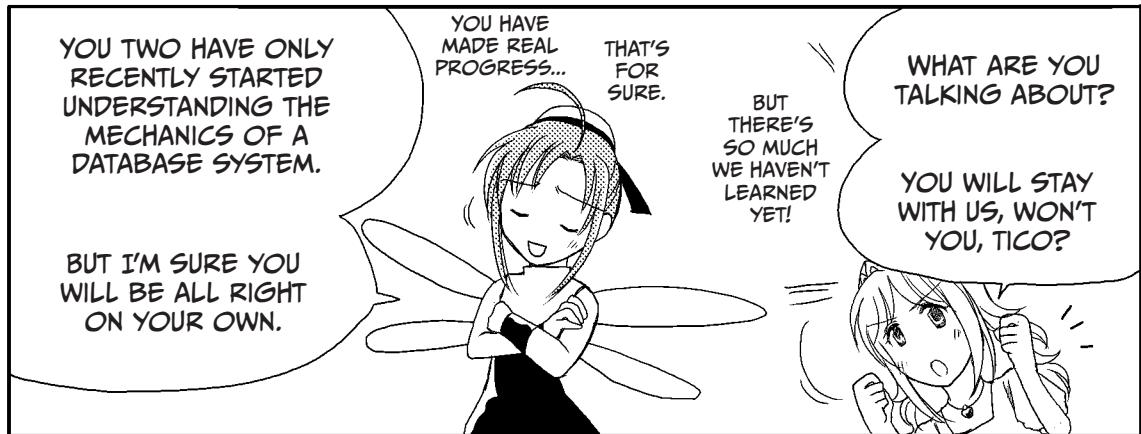


PULL THE TRIGGER AND A
BULLET IS SHOT. UPDATE DATA
AND A STORED PROCEDURE IS
ACTIVATED.

JUST BUYING ONE
BOOK CREATES A LOT
OF WORK BEHIND THE
SCENES, DOESN'T IT?







A
DATABASE
FAIRY!

びしょ

I WAS SIMPLY GOING
TO SAY GOOD-BYE
TODAY.

BUT SOMEHOW I'VE
MADE IT MORE THAN
THAT, IN SPITE OF
MYSELF.

IT'S BEEN A SHORT
BUT HAPPY TIME WITH
YOU TWO!

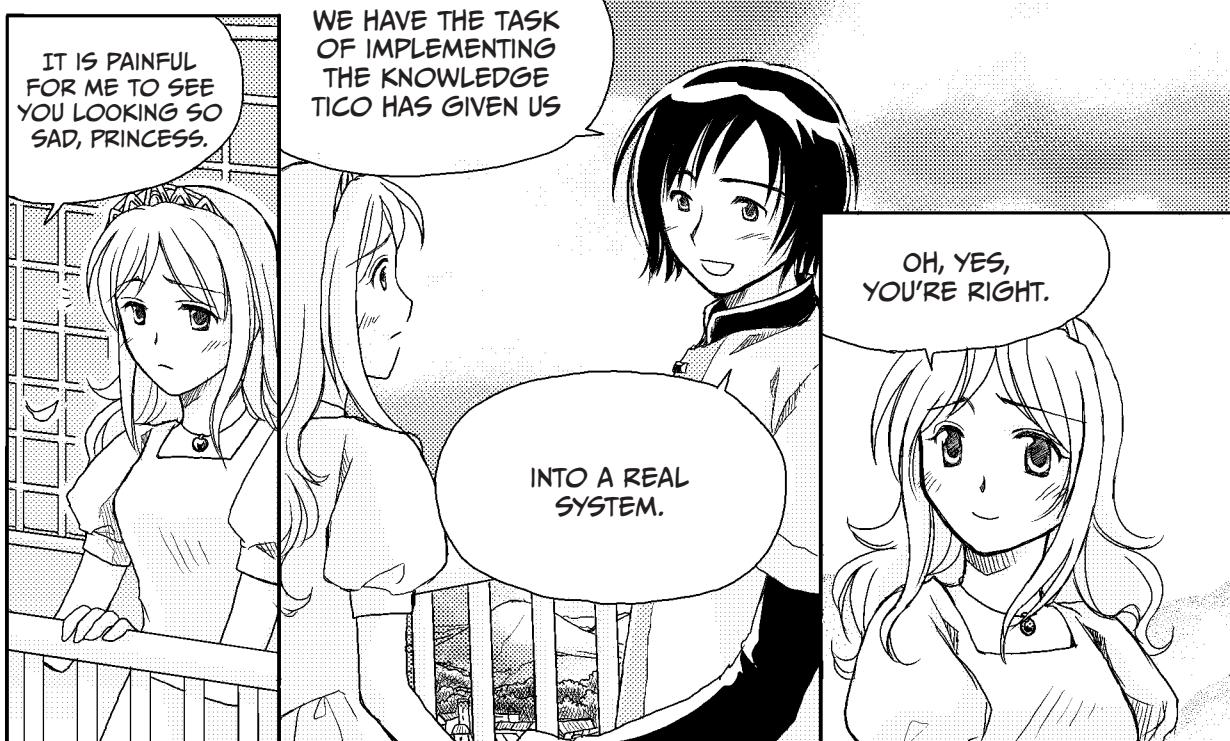
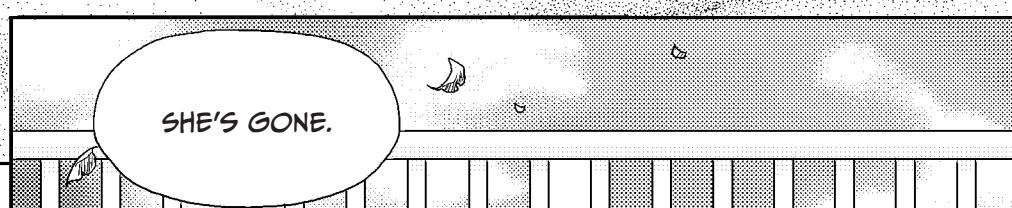
TICO, WAIT!

I HAVE TO...

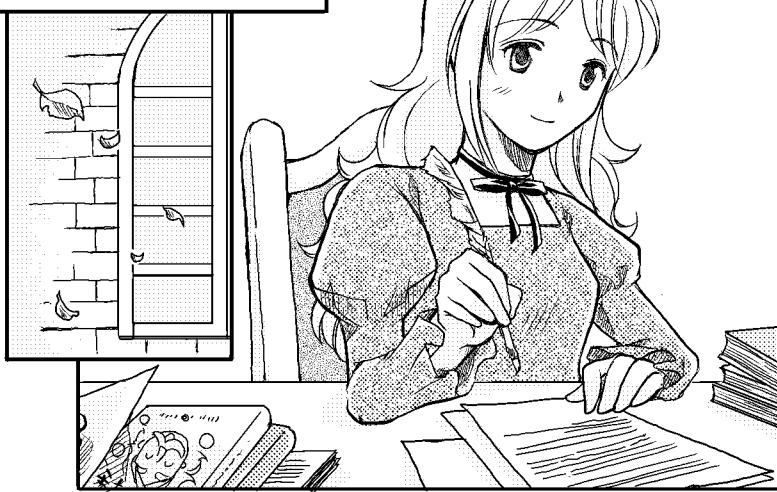
TICO!!

TICO...

TICO, DEAR!



DAYS HAVE GONE BY...



IS EVERYTHING ALL
RIGHT WITH YOUR
BOOK ON DATABASES,
PRINCESS?



YEAH!

DO YOU WANT
TO TAKE A
LOOK?

I AM MAKING
THINGS EASY FOR
EVERYBODY TO
UNDERSTAND.

SURE!

IT'S A GOOD
IDEA TO DO
IT IN A COMIC
BOOK STYLE.

SO CUTE...

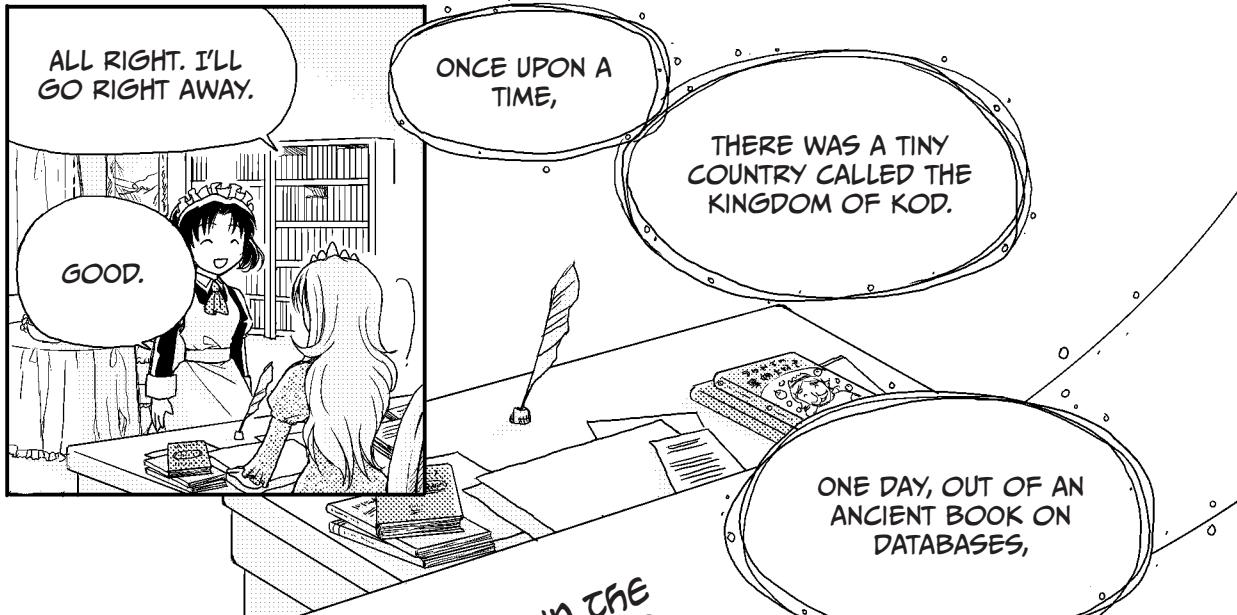
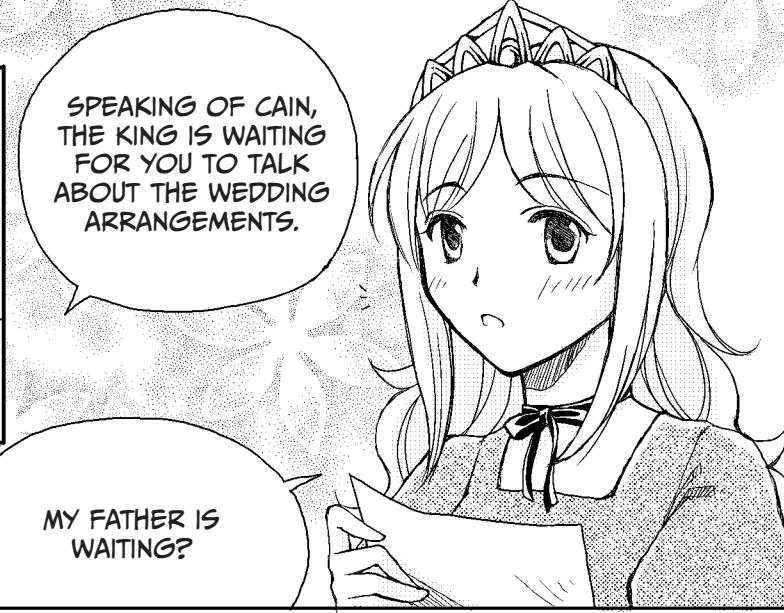
AND CAIN'S
DRAWINGS ARE
EXCELLENT.

FABULOUS!

AND
LOOK!

HERE! THIS IS THE
FRONT COVER.





DATABASES IN THE
KINGDOM OF KOD



FLEW A TINY
LITTLE GIRL....

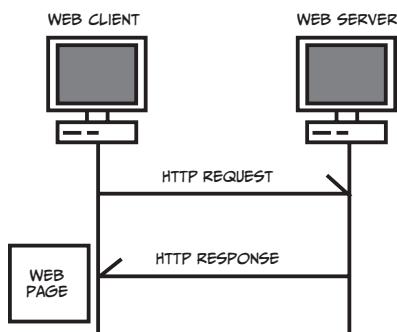
THE END

DATABASES ON THE WEB

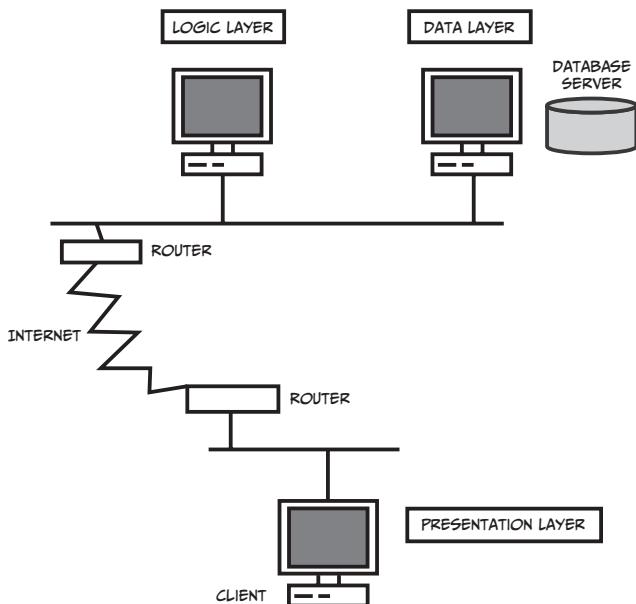


Databases are used for many different purposes, such as train seat reservation systems and bank deposit systems. They are indispensable in daily life and in business operations. As I showed Ruruna and Cain, web-based database systems are popular as well. In a web-based system, the communications protocol used is HyperText Transfer Protocol (HTTP). Server software running on a web server waits for a request from a user. When a user request (HTTP request) is sent, the software answers the request and returns a corresponding web page (HTTP response).

A *web page* consists of text files in HTML format. Other files specified by Uniform Resource Locators (URLs) are embedded within a web page to present information such as images.



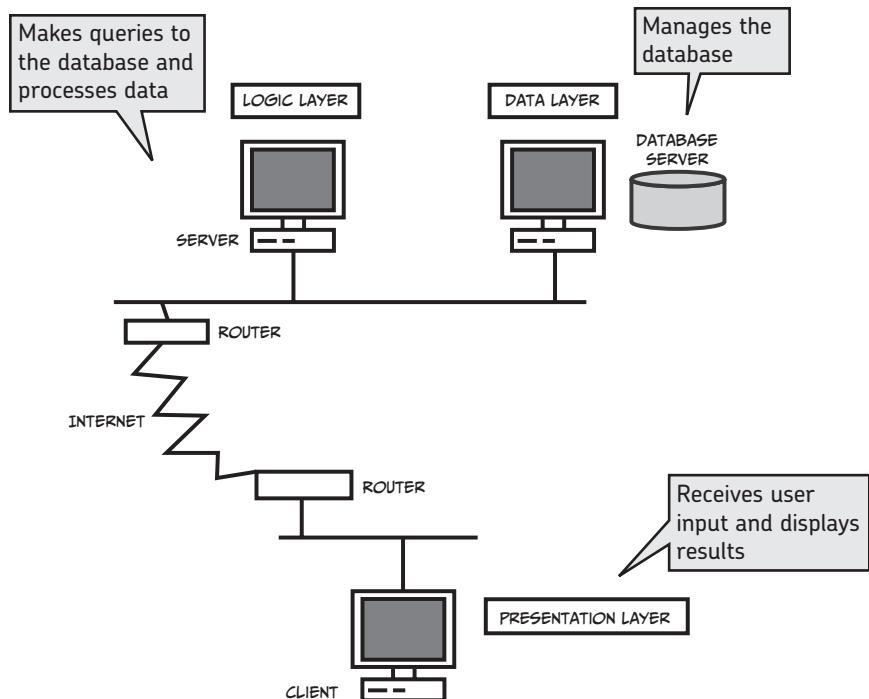
When a database is used with a web page, a database server is added to the system shown above. This system can be configured in three layers and is referred to as a *three-tier client/server system*. A three-tier client/server system consists of a presentation layer, a logic layer, and a data layer.



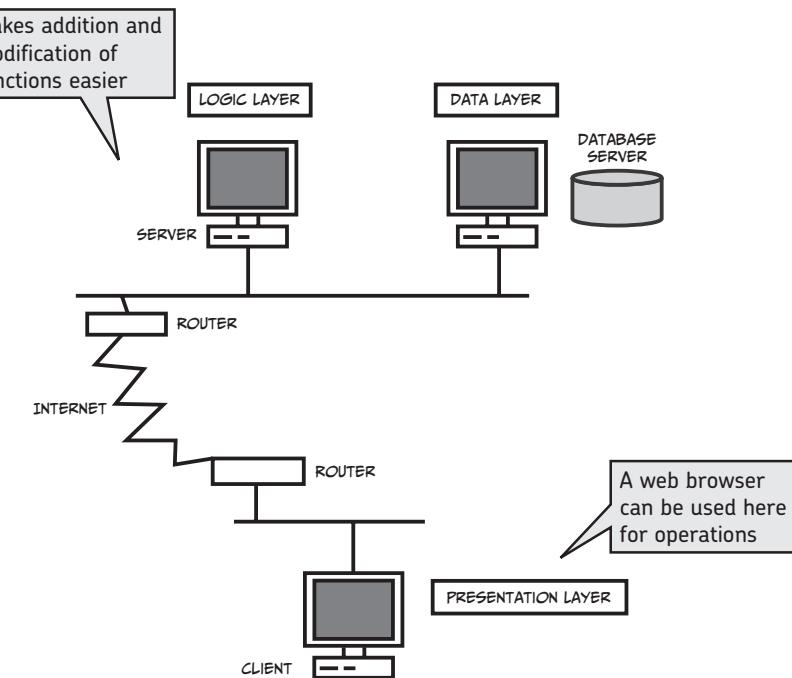
The *presentation layer* receives user input, such as search conditions, that needs to be passed on to the database. The presentation layer also processes query results received from the database so that they can be displayed. A web browser (such as Internet Explorer or Firefox) functions as a presentation tool for the user.

The *logic layer* performs data processing. This layer is where SQL statements are composed. Processes performed here are written in one or more programming languages. Depending on the contents and load of processes, several servers, such as an application server and a web server, may be used to handle processing.

The *data layer* processes data on a database server. Search results are returned from the database in response to SQL queries.



The three-tier client/server configuration is a flexible and simple system. For example, when making additions or modifications to an application, you can separate the portion you want to edit as a logic layer. In the presentation layer, you can use a web browser, eliminating the need for installing a separate software program.



USING STORED PROCEDURES

In a web-based system, too much traffic on the network can be a problem. Fortunately, you can store program logic inside the database server itself as stored procedures.

Storing procedures on the database server helps reduce the load on the network, because it eliminates the need for frequent transfers of SQL queries. In addition, storing procedures also makes it easier to develop applications, since standard processes can be encapsulated into easy-to-use procedures. Actually, stored procedures are just a special kind of a more broad category called *stored programs*. The other two types of stored programs are *stored functions* and *triggers*.

TYPES OF STORED PROGRAMS

Program	Definition
Stored procedure	Program that does not return values from the processing procedure
Stored function	Program that returns values from the processing procedure
Trigger	Program that is launched automatically before and after the database operations



QUESTIONS

Can you answer these questions? The correct answers are on page 205.

Q1

In a three-tier client/server system, on which layer does the database operate?

Q2

In a three-tier client/server system, on which layer are user interactions received and results displayed?

WHAT IS A DISTRIBUTED DATABASE?



In a Web-based system, processing is distributed among a database server, a web server, and a web browser, with different tasks assigned to each. This type of distributed system allows for flexible processing and decreases the processing capacity required by each server.

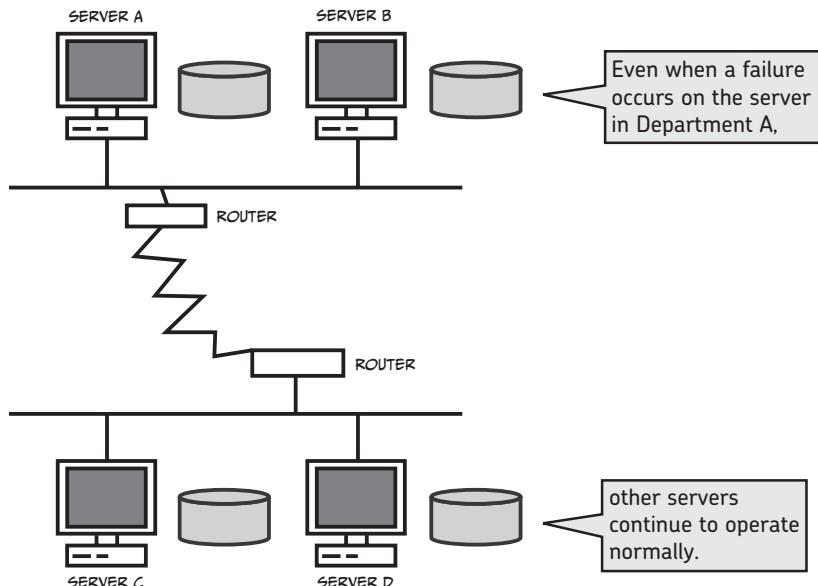
But a database server itself can be distributed among several servers. Distributed database servers can be in different locations or on the same network. Note, however, that a distributed database may be handled as a single database. If the distributed database appears to be a single server, the user doesn't have to worry about data locations or transfers.

A database can be distributed horizontally or vertically, as you'll see.

HORIZONTAL DISTRIBUTION

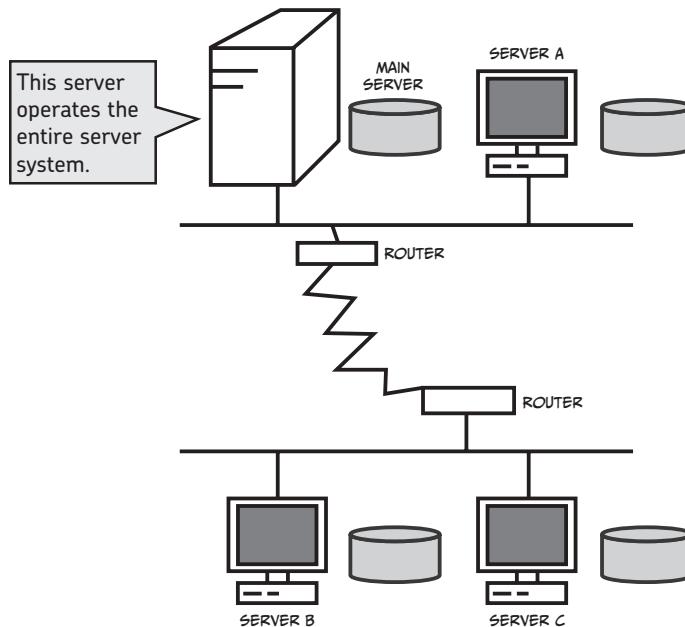
Horizontal distribution uses several peer database servers. Each database server can use data from other database servers, and in turn, each one makes itself available to the other database servers. This structure is used for a system of extended databases that operate separately in each department.

A horizontally distributed database is a failure-resistant system by design, since failure on one server will not affect database operation.



VERTICAL DISTRIBUTION

Vertical distribution assigns different functions to different database servers. One of the servers functions as the main server and performs a key role, while the others are in charge of processing tasks as requested. A vertically distributed database makes it easier to manage the main database server, though this main server will have a heavy load. An example of vertical distribution would include a company-wide main server and individual servers operating in each department.

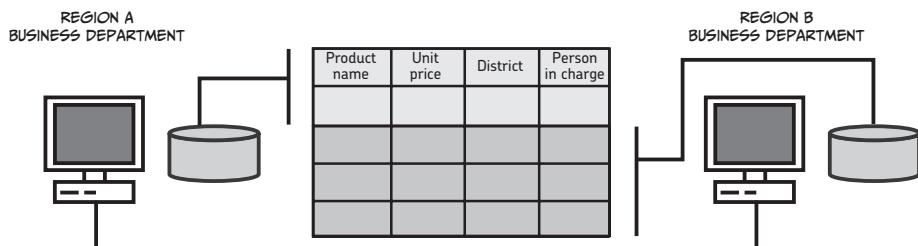


PARTITIONING DATA

In a distributed database, data is spread across servers for storage. You should carefully consider how to divide up the data. Data can be split in the following ways.

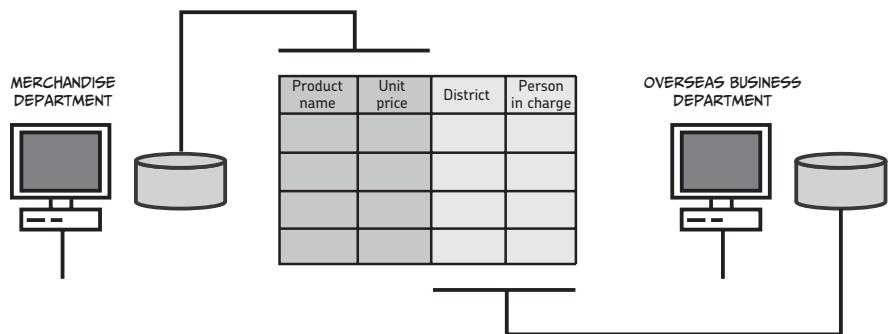
HORIZONTAL PARTITIONING

A *horizontal partition* divides data into units of rows. Rows resulting from the split are distributed across servers. This form of partitioning is often used when data can be ordered into groups in such a way that related data, which is often accessed at the same time, is stored on the same server.



VERTICAL PARTITIONING

A *vertical partition* divides data into units of columns. Columns resulting from the split are distributed across servers. For example, a vertical partition can be used to manage and join independent databases belonging to departments like the Merchandise Department, the Overseas Business Department, and the Export Department.



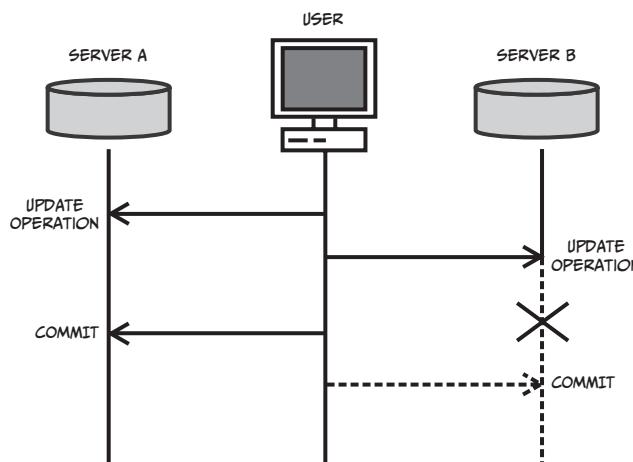
PREVENTING INCONSISTENCIES WITH a TWO-PHASE COMMIT



Databases on different servers in a distributed database system can be configured to act as a single database in the eyes of users. To achieve this, various steps must be taken to deal with the fact that data is actually distributed across different servers.

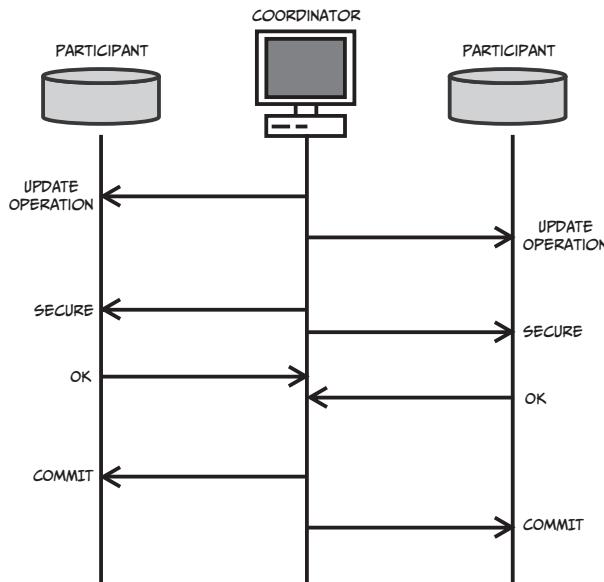
First, whenever data is committed, all data on all servers must be updated consistently.

In a distributed database system, the standard commit method may lead to one of the servers being updated while another is not, as shown below. This is a violation of the atomicity property of transactions, as this transaction will not end with either a commit or rollback. This would also cause the database system as a whole to become inconsistent.

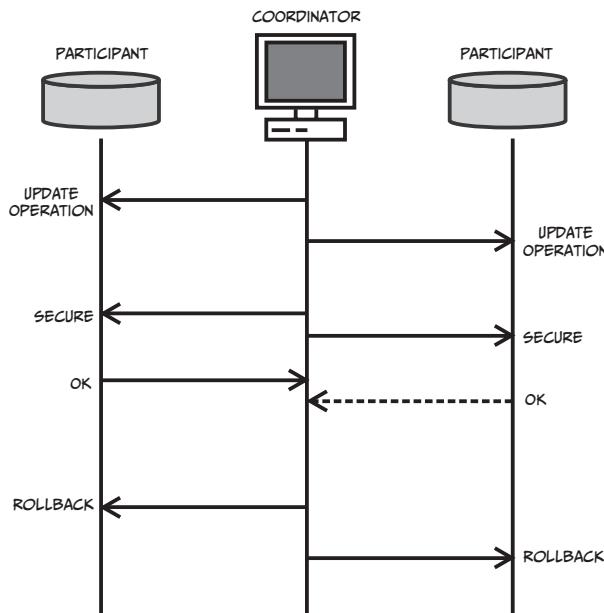


Therefore, a two-phase commit is adopted in a distributed database system. The *two-phase commit* creates one commit operation from both the first and the second commit operations.

A two-phase commit operation involves a coordinator and participants. In the first phase of a two-phase commit operation, the coordinator asks the participants if a commit operation is possible. The participants send an OK reply if it is. This preparatory step is referred to as a *prepare*. In the second phase, the coordinator gives the instructions for a commit, and all participants perform a commit accordingly.



If any one node fails to secure the operation in the two-phase commit, all participants receive a rollback directive. This is how databases on all servers remain consistent with each other.





QUESTIONS

Try these questions about two-phase commits. The answers are on page 205.

Q3

In a two-phase commit scheme, what instructions does the coordinator give during the first phase?

Q4

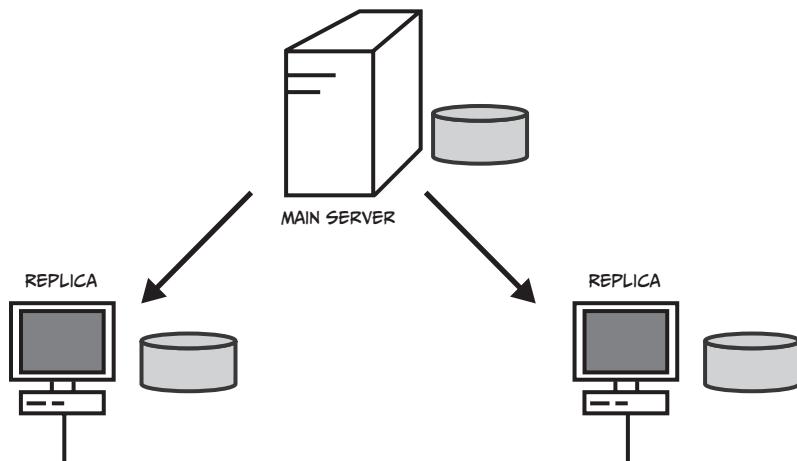
In a two-phase commit scheme, what instructions does the coordinator give during the second phase?

DATABASE REPLICATION

Some distributed databases have a duplicated, or replica, database that reduces the load on the network. This practice is referred to as *replication*. The primary database is referred to as the *master database*, and the copy is called the *replica*. There are several types of replication.

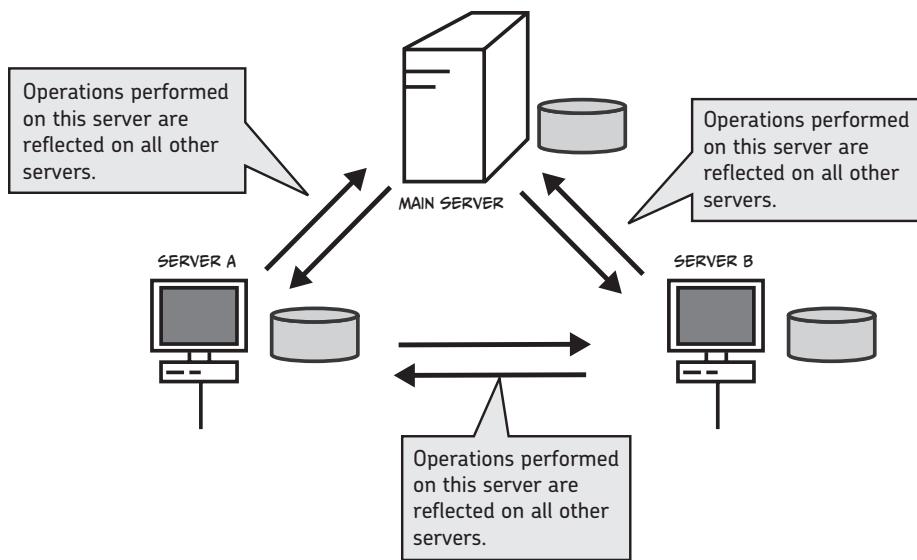
READ-ONLY

A *read-only replica* is created and downloaded from the master database on the main server. To change data, users must connect to the main server.



REPLICATION ENABLED FOR ALL SERVERS

In this method, the same master database is shared by all servers. Updates to any of the servers are reflected in all other servers.



FURTHER APPLICATION OF DATABASES



This final section introduces applied technologies related to databases.

XML

Extensible Markup Language (XML) is becoming increasingly popular as a data storage method. XML represents data by enclosing it in tags. Since these tags can convey information about the data they contain, this language is useful for data storage and retrieval.

XML is useful because its strictly structured grammar makes programmed processes easy. Moreover, XML comes in text files (which are easy to edit) and can communicate with other systems. For these reasons, XML is sometimes used as a data representation method in place of a database.

```
<?xml version="1.0"?>
<products>
  <fruit>
    <product code>101</product code>
    <product name>Melon</product name>
    <unit price>800</unit price>
  </fruit>
  <fruit>
    <product code>102</product code>
    <product name>Strawberry</product name>
    <unit price>150</unit price>
  </fruit>
  <fruit>
    <product code>103</product code>
    <product name>Apple</product name>
    <unit price>120</unit price>
  </fruit>
</products>
```

OBJECT-ORIENTED DATABASES

A relational database stores text data in a table. However, a relational database may be inadequate when handling certain types of data. That's where an object-oriented database (OODB) comes in.

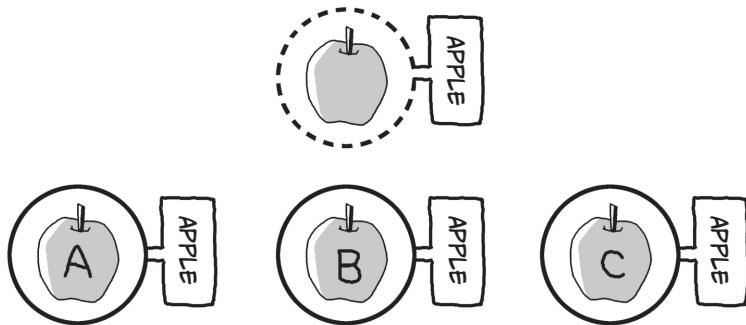
The object-oriented method uses *objects*—sets of data and instructions on how that data should be used. You can hide the data and only expose the operations upon the data in order to handle the object as an independent component. This technique is referred to as *encapsulation*.

In an object-oriented database, each object is represented with an identifier. Sometimes, an object is also called an *instance*.

In an object-oriented database, you can also manage *compound objects*—one object nested within another. This means, for example, that you can store data consisting of an image combined with text as a single object. The object-oriented database allows for flexible management of complex data.

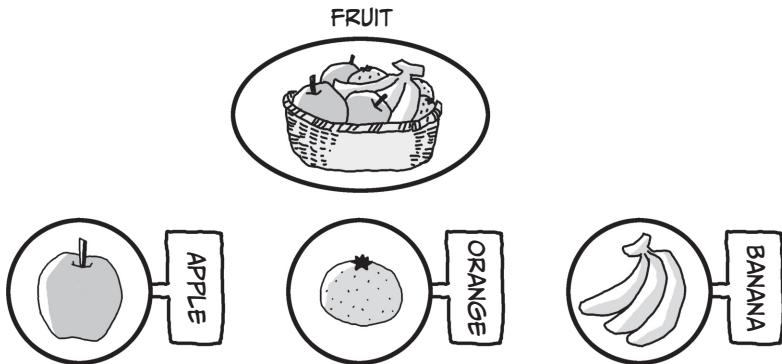


In an object-oriented database, various concepts can ease object-oriented development. The template for objects is referred to as *class*. For example, suppose you have designed an Apple class. Objects (instances) in that class may be Apple A, Apple B, and so on. The Apple class enables the creation of these objects.



In an object-oriented scheme, a class can also have hierarchical relationships. You can create a child class that has the same data and functions of a parent class. This relationship is referred to as *inheritance*. You can also give unique functions to the child class.

For example, class Apple and class Orange may inherit the data and functions from class Fruit, but they also each have their own unique data and functions. In an object-oriented scheme, you can use hierarchical relationships to allow for efficient development.



SUMMARY



- The three-tier client/server system is a method of Web-based system configuration.
- A database acts as a data layer.
- A distributed database system handles databases that are dispersed.
- A two-phase commit method is used in a distributed database.

ANSWERS

- Q1** Data layer
Q2 Presentation layer
Q3 Prepare
Q4 Commit or rollback

CLOSING REMARKS

Have you enjoyed studying databases? You will need to learn even more before you can manage all the aspects of operating a database, but the fundamentals of databases always stay the same. By firmly understanding the basics, you can identify significant data in the real world and design and operate databases. You can acquire advanced database skills by building on your fundamental knowledge. Good luck!

FREQUENTLY USED SQL STATEMENTS

BASIC QUERY

```
SELECT column_name, ...
FROM table_name;
```

CONDITIONAL QUERY

```
SELECT column_name, ...
FROM table_name
WHERE condition;
```

PATTERN MATCHING

```
SELECT column_name, ...
FROM table_name
WHERE column_name LIKE 'pattern';
```

SORTED SEARCH

```
SELECT column_name, ...
FROM table_name
WHERE condition
ORDER BY column_name;
```

AGGREGATING AND GROUPING

```
SELECT column_name, ...
FROM table_name
WHERE condition
GROUP BY column_names_for_grouping
HAVING condition_for_grouped_rows
```

JOINING TABLES

```
SELECT table_name1.column_name, ...
FROM table_name1,table_name2, ...
WHERE table_name1.column_name = table_name2.column_name
```

CREATING A TABLE

```
CREATE TABLE table_name(  
    column_name1 datatype,  
    column_name2 datatype,  
    ...  
)
```

CREATING A VIEW

```
CREATE VIEW view_name  
AS SELECT statement
```

DELETING A REAL TABLE

```
DROP TABLE table_name;
```

DELETING A VIEW

```
DROP VIEW view_name;
```

INSERTING A ROW

```
INSERT INTO table_name(column_name1, ...)  
VALUES (value1, ...)
```

UPDATING A ROW

```
UPDATE table_name  
SET column_name = value1, ...  
WHERE condition;
```

DELETING A ROW

```
DELETE FROM table_name  
WHERE condition;
```

REFERENCES

- Chen, P. P. 1976. "The Entity-Relationship Model: Toward a Unified View of Data," *ACM Transactions on Database Systems* 1 (1): 9-36.
- Codd, E. F. 1970. "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM*, 13 (6): 377-387.
- Date, C.J. and Hugh Darwen. 1997. *A Guide to the SQL Standard*, 4th ed. Reading, MA: Addison-Wesley.
- Masunaga, Yoshifumi. 1990. *Basics of Relational Database*. Tokyo: Ohmsha.
- Database Language: SQL, JIS X3005-1-4, 2002.
- ISO/IEC 9075, Information Technology—Database Languages—SQL, 1992.
- ISO/IEC 9075, Information Technology—Database Languages—SQL, 1995.
- ISO/IEC 9075, 1, 2, 3, 4, Information Technology—Database Languages—SQL, 1999.
- IT Engineers' Skill Standards—Technical Engineers (Database), Information-Technology Promotion Agency, Japan.

INDEX

A

access rights, 19, 106, 126–129, 141–142, 159–160, 167
actual condition, 52, 55–56, 59, 74
aggregate functions, 98–100, 110–111
ALL statements, 159
application servers, 182, 195
atomicity, 153–154
authorized users, 141, 159
AVG (average) function, 98, 99, 110

B

backup copies, 161
base tables, 160
B-tree indexing, 163
buffers, 161

C

cardinality, 74
Cartesian product operations, 37, 39, 42
character strings, 84, 108
checkpoints, 161–162
coarse granularity, 157
columns, 34, 84
COMMIT statements, 133, 137, 150, 154, 205
comparison operators, 107
compound objects, 203
conceptual schema, 81
concurrency controls
 isolation levels, 158
 lock-based controls, 131–137, 155–157, 167
 optimistic controls, 158
 timestamp controls, 158
conflicting data, 13, 17–18, 21, 60, 71, 116, 153, 158
consistency, 153, 154–155, 184
controlling, user access, 19, 106, 126–129, 141–142, 159–160, 167
correlated subqueries, 113–114
corrupted data, 20, 154
cost-based processing, 167
COUNT functions, 99–100, 110
CREATE TABLE statements, 103, 115–119
CREATE VIEW statements, 117

D

Data Control Language (DCL), 106
Data Definition Language (DDL), 106
data extraction operations, 36–37, 39–47
data input, 21, 90–92, 103–104, 106, 116
data layers, 194–196, 205
Data Manipulation Language (DML), 106
data models, 32–39
data processing, 35–37, 47–48, 130, 159, 167, 182, 195–198
data recovery, 20, 147–152, 161–164, 167
data security, 19, 127, 138–142, 159–160, 161–164, 167, 176, 182, 184
data tags, 202
database design, 19, 26
 determining data conditions, 74
 E-R (entity-relationship) model, 50–55, 74–77, 81
 normalization, 60–72, 78–81
 steps for, 81, 84
database failures, 161
database management system (DBMS), 21
database replication, 201–202
database terms, 26–31
databases
 building from existing systems, 14
 defined, 6, 10, 15, 187
 efficiency of, 3–4, 15, 19, 146, 174
 types of, 32–39
 use of, 19–21, 175–182
DBMS (database management system), 21
DCL (Data Control Language), 106
DDL (Data Definition Language), 106
deadlocks, 136, 158
DELETE statements, 104, 116, 118, 119, 159
difference operations, 37, 39, 41
dirty read, 158
disaster recovery, 20, 147–150, 161–164, 167
disk access count, 145
distributed database systems
 horizontal distribution, 197
 overview of, 183–184, 197–199, 205
partitioning data, 198–199
replication in, 201–202

two-phase commit operations in, 199–201, 205

vertical distribution, 198

dividing tables. See normalization

division operations, 37, 43, 45

DML (Data Manipulation Language), 106

DROP TABLE/DROP VIEW

 statements, 118

duplicated data, 11, 16, 18, 19, 21, 29

durability, 153, 159–160

E

encapsulation, 203

entities, 52–54, 74

entity-relationship (E-R) model, 50–55, 74–77, 81

exclusive locks, 134–136, 155–156

Extensible Markup Language (XML), 202

external schema, 81

extraction operations, data, 36–37, 39–47

F

failure-resistant systems, 184, 197

failures, database, 161

fields, 27–28, 30, 34, 35, 48

file-based systems, 3, 10, 16, 21, 32

fine granularity, 157

first normal forms, 62–64, 66, 78–79

foreign keys, 44, 48, 72, 101

forms, 62–70, 81–82

full-match searching, 163

functionally dependent values, 79

G

GRANT statements, 159, 168

granularity, 157

GROUP BY phrase, 110

grouping, 110, 159

H

hash function, 167

hash indexing, 163

HAVING phrase, 111

hierarchical data model, 32, 33, 39, 204

horizontal distribution, 197

horizontal partition, 198

HyperText Markup Language (HTML), 194

HyperText Transfer Protocol (HTTP), 178, 180, 194

I

inconsistent data, 153, 154, 159, 199–201
independent data management, 19, 72
indexes/indexing, 143–147, 162–164, 167
inner join, 115
inputting data, 21, 90–92, 103–104, 106, 116
INSERT statements, 104, 116, 119, 159
instances, 203
internal schema, 81
International Organization for Standardization (ISO), 124
International Standard Book Number (ISBN), 45
Internet databases. *See* Web-based database systems
intersection operations, 37, 39, 41
ISBN (International Standard Book Number), 45
ISO (International Organization for Standardization), 124
isolation, 153, 155–158
isolation levels, 158

J

Japanese Industrial Standards (JIS), 124
join operations, 37, 43, 44, 48, 165
joining tables, 44, 101–102, 114–115

K

keys
foreign, 44, 48, 72, 101
primary, 35, 44, 48, 65, 67, 72, 78–79, 101, 103, 115

L

LIKE statements, 97, 108
locking granularity, 157
locks/lock-based controls, 131–137, 155–157, 167, 175–176, 182
logic layer, 194–196
logical operators, 107
logs, 148–149
lost data, 20, 154

M

many-to-many relationships, 55, 74, 75, 81
master databases, 201–202
mathematical operations. *See* operations

MAX (maximum value) function, 99–100, 110
media failures, 161
memory. *See* stored procedures
MIN (minimum value) function, 99, 110

N

nested loop method, 165
network data model, 33, 39
non-repeatable read, 158
normalization, 60–72, 78–81
normalized tables, 72, 91
null, 30, 108

O

object-oriented databases (OODB), 203–205
one-to-many relationships, 55, 75, 81
one-to-one relationships, 74, 81
OODB (object-oriented databases), 203–205
operations
Cartesian product, 37, 39, 42
data extraction, 39–47
difference, 37, 39, 41
division, 37, 43, 45
intersection, 37, 39, 41
join, 37, 43, 44, 48, 165
projection, 36, 37, 43, 165
relational, 43–47
selection, 37, 39, 43, 47, 48, 165
set, 39–42
union, 37, 39, 40, 48

operators, 107
optimistic controls, 158
optimization, query, 164–167
optimizers, 167
ORDER BY statements, 98
outer join, 115

P

partitioning data, 198–199
passwords, 141
pattern matching, 108
permissions, 19, 141–142, 159–160, 167
phantom read, 158
presentation layers, 205
primary keys, 35, 44, 48, 65, 67, 72, 78–79, 79, 101, 103, 115
problems, data management
conflicting data, 13, 17–18, 21, 60, 71, 116, 153, 158
corrupted/lost data, 20, 154

database failures, 161

difficulty changing data, 13, 14, 17, 18
duplicated data, 11, 16, 18, 19, 21, 29
inconsistent data, 153, 154, 159, 199–201
shared data, 12, 20, 21, 129, 175
processing data, 35–37, 47–48, 130, 159, 167, 182, 195–198
programming languages, 178, 180, 194, 202
projection operations, 36, 37, 43, 165
protecting data, 19, 127, 138–142, 159–160, 161–164, 167, 176, 182, 184

Q

queries. *See* SQL; SQL statements
query optimization, 164–167

R

READ COMMITTED transactions, 158
read operations, 130, 133, 134, 159
READ UNCOMMITTED transactions, 158
read-only replica, 201
records, 27–28, 34, 48, 148–149
recovery mechanisms, 20, 147–150, 161–164, 167
relational data model, 33–34, 35, 39, 47, 48
relational operations, 43–47
relationships
concept of, 54, 74
E-R (entity-relationship) model, 50–55, 74–77, 81
hierarchical relationships, 32, 33, 39, 204
many-to-many relationships, 55, 74, 75, 81

one-to-many relationships, 55, 75, 81
one-to-one relationships, 74, 81
remarks, 30–31
REPEATABLE READ transactions, 158
replicas, 201–202
resources in transactions, 155
retrieving data, 36–37, 39–47, 90–92, 95–99, 101–102, 106, 180, 202
REVOKE statements, 159, 160, 168
right outer join, 115
ROLLBACK statements, 136–137, 150, 153–154, 205
rolling forward, 149
rows, 34, 84, 116
rule-based processing, 167

S

schemas, 81
search methods, 93–97, 106, 108,
 112–115, 163. *See also* SQL
second normal forms, 62, 64, 66–69,
 78–79, 82
security, data, 19, 127, 138–142,
 159–160, 161–164, 167, 176,
 182, 184
SELECT statements, 93–97, 98, 105,
 106, 113, 119, 159
selection operations, 37, 39, 43, 47,
 48, 165
separate data management, 11, 19, 72
SERIALIZABLE transactions, 155,
 156, 158
servers, 178–185, 194–197, 205
set functions, 98, 110
set operations, 39–42
SET TRANSACTION statements, 158, 160
shared data, problems with, 12, 20, 21,
 129, 175
shared locks, 133, 155–156
sort merge method, 166
sorting. *See* aggregate functions; indexes/
 indexing
SQL (Structured Query Language)
 aggregate functions, 98–100, 110–111
 comparison operators, 107
 conditions, creating, 95–96, 101,
 107–109
 data manipulation, 90–92, 100,
 106, 116
 GROUP BY phrase, 110
 HAVING phrase, 111
 joining tables, 44, 101–102, 114–115
 logical operators, 107
 overview of, 90–92, 106, 116, 124
 pattern matching, 108
 phrases used in, 94–97, 106,
 110–111, 119
 query optimization, 164–167
 search methods, 93–97, 106, 108,
 112–115, 163
 standardization of, 124
 subqueries, 112–114
 tables, creating, 91–92, 103–105, 106,
 115–119
 views, creating, 117, 160
Web-based databases and, 178–179,
 195–196
WHERE phrase, 94–97, 106, 110, 119
wild cards, 97, 108

SQL statements

ALL, 159
COMMIT, 133, 137, 150, 154, 205
CREATE TABLE, 103, 115–119
CREATE VIEW, 117
DELETE, 104, 116, 118, 119, 159
DROP TABLE/DROP VIEW, 118
GRANT, 159, 168
INSERT, 104, 116, 119, 159
LIKE, 97, 108
ORDER BY, 98
REVOKE, 159, 160, 168
ROLLBACK, 136–137, 150, 153–154,
 154, 205
SELECT, 93–97, 98, 105, 106, 113,
 119, 159
SET TRANSACTION, 158, 160
UPDATE, 104, 116, 119, 159
SQL92/SQL99, 124
stored functions, 196
stored procedures, 185–188, 196
Structured Query Language. *See* SQL
 (Structured Query Language)
subqueries, 112–114
SUM function, 99, 110
system failures, 161

T

tables
 base, 160
 concept of, 34, 39, 48
 constraints of, 116
 creating, 57–59, 91–92, 103–105,
 106, 115–119
 forms for, 62–70, 81–82
 joining, 44, 101–102, 114–115
 multiple, 59
 normalization of, 60–72, 78–81
 normalized, 72, 91
 two-dimensional, 34, 79
 views, 117, 160
tabulation, 34, 47, 57. *See also*
 normalization
tags, 202
third normal forms, 62, 68, 69–70,
 78–79, 81, 82
three-tier client/server systems, 194–196,
 197, 205
timestamp controls, 158
transactions
 defined, 126–130
 disaster recovery and, 149–150
 failures in, 161

properties of, 153–160

read/write operations, 130, 133,
 134, 159
transitively dependent values, 79
triggers, 187, 196
two-phase commit operations,
 199–201, 205
two-phase locking, 156–157

U

unauthorized data overwrites, 140
uncoordinated data management, 10
uniform data management, 19
union operations, 37, 39, 40, 48
unique fields, 30
unnormalized forms, 62, 78–79
UPDATE statements, 104, 116, 119, 159
user permissions, 19, 106, 126–129,
 141–142, 159–160, 167
usernames, 141

V

vertical distribution, 198
vertical partition, 199
views, creating, 117, 160

W

Web-based database systems, 177–182,
 194–197
WHERE phrase, 94–97, 106, 110, 119
wild cards, 97, 108
write operations, 130, 133–134, 159

X

XML (Extensible Markup Language), 202

ABOUT THE AUTHOR

Mana Takahashi is a graduate of the University of Tokyo, Faculty of Economics, in Tokyo, Japan. She is an active technical writer and has published a number of books on topics such as Java, C, XML, Information Engineering, and System Administration.

COLOPHON

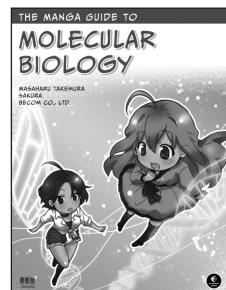
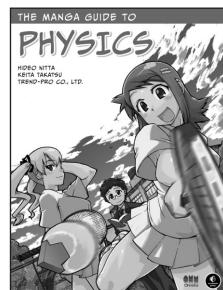
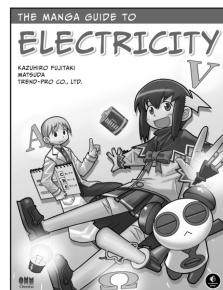
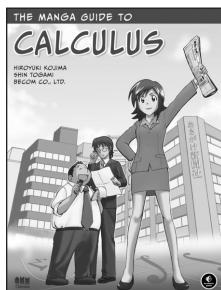
The Manga Guide to Databases was laid out in Adobe InDesign. The fonts are CCMeanwhile and Chevin.

UPDATES

Visit http://www.nostarch.com/mg_databases.htm for updates, errata, and other information.

MORE MANGA GUIDES

Find more Manga Guides at your favorite bookstore, and learn more about the series at <http://www.edumanga.me/>.



UNLEASH THE POWER OF THE DATABASE!



PRINCESS RURUNA AND CAIN HAVE A PROBLEM: THEIR FRUIT-SELLING EMPIRE IS A TANGLE OF CONFLICTING AND DUPLICATED DATA, AND SORTING THE MELONS FROM THE APPLES AND STRAWBERRIES IS CAUSING REAL DIFFICULTIES. BUT WHAT CAN THEY DO?

WHY, BUILD A RELATIONAL DATABASE OF COURSE, WITH THE HELP OF TICO, THE MAGICAL DATABASE FAIRY. FOLLOW ALONG IN **THE MANGA GUIDE TO DATABASES** AS TICO TEACHES RURUNA AND CAIN HOW TO BUILD A DATABASE TO MANAGE THEIR KINGDOM'S SALES, MERCHANDISE, AND EXPORTS. YOU'LL LEARN HOW DATABASES WORK AND THE MEANING OF TERMS LIKE SCHEMAS, KEYS, NORMALIZATION, AND TRANSACTIONS.

TOGETHER WITH RURUNA AND CAIN YOU'LL LEARN HOW TO:

- EXTRACT DATA FROM A RELATIONAL DATABASE USING SET AND RELATIONAL OPERATIONS

- APPLY THE ENTITY-RELATIONSHIP MODEL TO ACCURATELY REPRESENT YOUR OWN DATA
- CONTROL USER PERMISSIONS AND USE LOCKS TO PREVENT CONFLICTING OR DUPLICATED DATA
- USE SQL TO UPDATE OR RETRIEVE DATA AND CREATE REPORTS

YOU'LL EVEN EXPLORE THE BASICS OF INDEXING, SECURITY, DISASTER RECOVERY, REPLICATION, AND MORE.

IF YOUR HEAD SPINS WHEN PEOPLE SAY "DATABASE" OR YOU JUST FEEL LOST IN A MAZE OF NUMBERS AND DATA THAT YOU CAN'T SEEM TO CONTROL, TAG ALONG WITH RURUNA AND CAIN AS THEY LEARN EVERYTHING THEY NEED TO KNOW IN THE PAGES OF **THE MANGA GUIDE TO DATABASES**.



THE FINEST IN GEEK ENTERTAINMENT™
www.nostarch.com

\$19.95 (\$19.95 CDN)

SHELF IN: COMPUTERS/DATABASES

ISBN: 978-1-59327-190-9



9 781593 271909



5 1 9 9 5



6 89145 71905 5