Appendix C

The University Lab: Conceptual Design Verification, Logical Design, and Implementation

Preview

This appendix will *verify* the ER model developed in Appendix B, "The University Lab: Conceptual Design." **Verification** represents the link between the database modeling and design activities and the database applications design. Therefore, the verification process requires that you identify and define all database transactions (insert, update, delete, and outputs) and be flexible enough to support expected enhancements and modifications.

The verification process will also enable the designer to find and eliminate unnecessary data redundancies, to help ensure database integrity, to discover appropriate enhancements, and to verify that all stated end-user requirements are met. The verification process includes the integration of all of the different end-user views of the database, each with its own set of requirements and transactions. In this appendix, as in the real world, the verification process leads to modifications in the initial ER model. This verification process makes use of normalization procedures that are usually considered to be part of the *logical* design phase. However, in a real-world environment, the verification process generally uses modeling and normalization procedures *concurrently*. The modifications may include the creation and/or deletion of new entities, additional attributes in existing entities, and relationships.

Before the verification process can begin, you must identify and define all attributes and domains for each entity in the initial ER model and normalize the entities. You must also select a proper primary key and place foreign keys to link the entities. After completing the verification process, you finish the design process by formulating the logical and physical models.

verification
The process of refining a conceptual data model into a detailed design that is capable of supporting all required database transactions, and input and output requirements.

Data Files and Available Formats

MS Access | Oracle | MS SQL | My SQL

MS Access | Oracle | MS SQL | My SQL

There are no data files for this appendix.

Data Files Available on cengagebrain.com

C-1 Completing the Conceptual and Logical Database Designs

The conceptual database blueprint developed in Appendix B is still in a rough-draft format. Although it helps you define the basic characteristics of the database environment, the design lacks the details that allow you to implement it effectively. Using an analogy, if an architect's blueprint shows a wall, it is important to know whether that wall will be made of board, brick, block, or poured concrete and whether that wall will bear a load or merely act as a partition. In short, *detail matters*.

Before continuing, you might find it helpful to review the database life cycle (DBLC) in Chapter 9 Database Design. Doing this will help you evaluate what is accomplished in Appendix B and determine what remains to be done. In Appendix B, you completed:

- Phase 1 (the database initial study) of the database life cycle (DBLC).
- The initial pass through the DBLC's Phase 2 (the database design phase). That is, in Appendix B, you identified, analyzed, and refined the business rules; identified the main entities; and identified the relationships among those entities.

In this appendix, you will complete the conceptual and logical designs for the University Computer Lab's database. The physical design elements will be presented, and you will examine the issues to be confronted in the implementation phase. Table C.1 shows the specific tasks addressed.

TABLE C.1

TASKS ADDRESSED IN THIS CHAPTER

TASK	SECTION
Entity relationship modeling and normalization	Section C-2
Data model verification	Section C-3
Logical design	Section C-4
Physical design	Section C-5
Implementation	Section C-6
Testing and evaluation	Section C-7
Operation	Section C-8

The initial ER diagram in Appendix B (Figure B.19) will serve as the starting point. In other words, you will use the initial design as the basis for attribute definition, table normalization, and model verification to see if the design meets processing and information requirements. Keep in mind that the activities described are often concurrent and iterative. That is, they often take place simultaneously and are often repeated. For example, the definition of entities and their attributes is subject to normalization, which can generate additional entities and attributes, which are subject to normalization. If done properly, that process will yield an ER model whose entities, attributes, and relationships are capable of supporting the end-user data, information, and processing requirements.

To facilitate the completion of the conceptual model, you will use two modules, each supporting a functional area of the University Computer Lab. Those two modules, first introduced in Appendix B, Table B.4, are the

- Lab Management System, which reflects the Lab's daily operations. This module targets the Lab's users, the people who work in the Lab, and the scheduling of Lab resources. This module allows the computer lab director (CLD) to track the Lab's resources by user type, department, and so on. Such tracking will be an important resource when the Lab's budget is written.
- Inventory Management System, in which the equipment, supplies, orders, and repairs are tracked. (For example, equipment sent out for repair is temporarily *removed from* inventory, while repaired equipment is *returned to* inventory.) This module also allows the CLD to track equipment that is temporarily checked out for use by faculty members and staff.

A list of the entities identified during this process, as well as the attribute prefixes used, is shown in Table C.2.

TABLE C.2

THE UCL DESIGN ORGANIZATION

MODULE	ENTITIES	ATTRIBUTE PREFIX
Lab Management System	USER LOG LAB_ASSISTANT WORK_SCHEDULE HOURS_WORKED RESERVATION RES_SLOT	USER_ LOG_ LA_ SCHED_ HW_ RES_ RSLOT_
Inventory Management System	INV_TYPE ITEM STORAGE LOCATION REPAIR VENDOR ORDER ORDER_ITEM WITHDRAW WD_ITEM CHECK_OUT CHECK_OUT_ITEM INV_TRANS TR_ITEM	TY_ ITEM_ STOR_ LOC_ REP_ VEND_ ORD_ OI_ WD_ WI_ CO_ CI_ TRANS_ TI_

As you compare the entities listed in Table C.2 with the initial database design shown in Appendix B, you will note that several new entities have been introduced. For example, RES_SLOT has emerged because a single RESERVATION might trigger more than one reservation date and time. For example, on 11-Jan-2018, a professor might make three Lab reservations: from 8:00 a.m.–8:50 a.m. and from 1:00 p.m.–1:50 p.m. on 23-Jan-2018 and from 6:00 p.m.–8:40 p.m. on 25-Jan-2018. Therefore, there is a 1:M relationship between RESERVATION and RES_SLOT. The new entities will be discussed as you develop each module within the system. You will also discover that some of the entities shown in Table C.2 will be replaced by other entities during the revision process.

C-2 Completing the Conceptual Design: Entities, Attributes, and Normalization

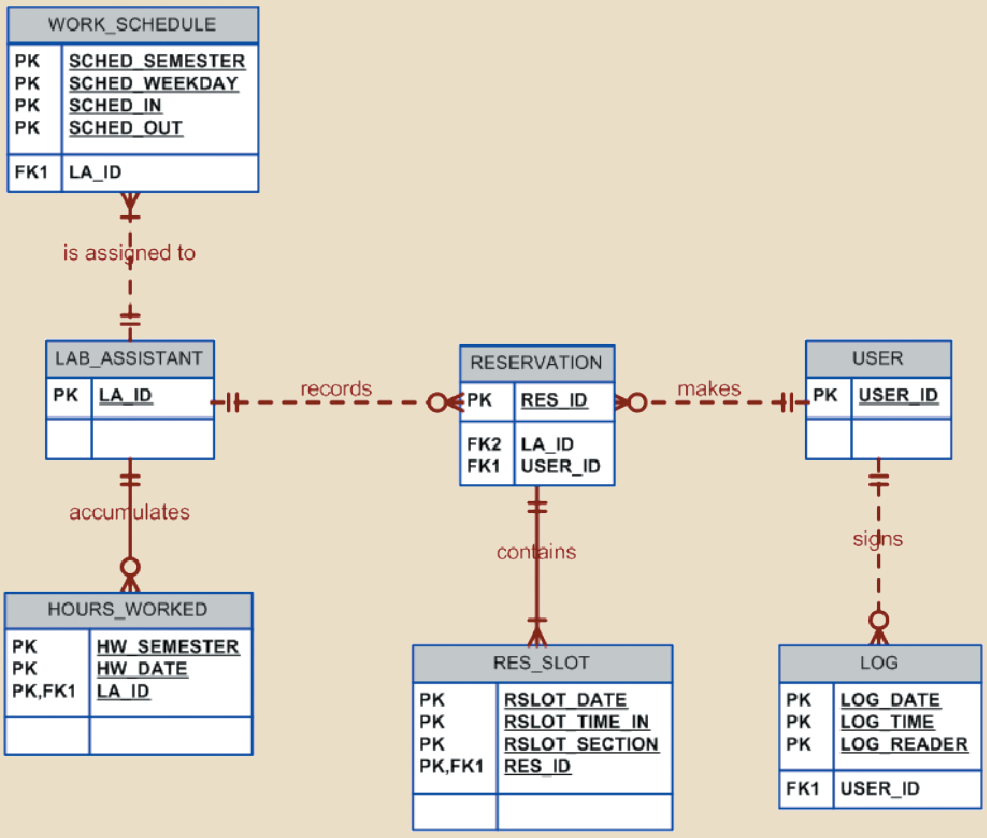
Section C-1 described two system modules. Each module's entities and their attributes will be defined next. Even as they are defined, entities and attributes are subject to the revisions that are often triggered by normalization. In other words, normalization is treated as an integral part of the ER modeling process. Therefore, functional dependencies are monitored carefully. Normalization techniques are used to discover new entities and some practical ways to evaluate their functions. The entities and their attributes are also subject to revision as they are evaluated in terms of end-user requirements.

The normalization techniques will not be covered again in this appendix. The structures presented here, however, have all been subjected to proper evaluation of their normalization levels. Your knowledge of the normalization principles and techniques will be necessary as you create and revise the entities and their attributes. As the revised model is developed, keep in mind the often conflicting requirements of design elegance, information requirements, and processing speed.

C-2a The Lab Management System Module

Before examining the structure of each of the Lab Management System module components, let's look at the ER segment presented in Figure C.1.

FIGURE C.1 THE LAB MANAGEMENT SYSTEM MODULE'S ER SEGMENT



The ER segment in Figure C.1 will be used as a map to track where you are in the process and where you are going. Using Figure C.1 and Table C.2 as a guide, let's begin by examining the USER entity's characteristics, shown in Table C.3.

TABLE C.3

THE USER ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
USER_ID	User identification code		PK	
DEPT_CODE	Department code			
USER_TYPE	User type: Fac = Faculty Staff = Staff Stu = Student			
USER_CLASS	User class: UG = Undergraduate GR = Graduate Fac = Faculty Staff = Staff			
USER_GENDER	M = Male F = Female			

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

Note

To let you focus on the relationships among the entities, all of the ERDs used in this appendix will show only the PK and FK attributes for each of the entities. You may, of course, add the additional attributes that will be defined for you in each of the summary tables.

If you use Visio Professional or any similar CASE tool to design the database, remember that you create only an entity's PK attribute at the *entity* level if its *relationship* to its parent entity is non-identifying. You never define the FK at the *entity* level for any entity, no matter what its relationship(s) to other entities. Instead, first create the *entities* and their PK attributes—as long as none of those PK attributes is inherited from related entities. If you use Visio Professional, attaching the relationship lines will ensure that the following actions are taken:

- All of the FKs will be inserted into the entities to properly reflect the relationships that you have defined for those entities.
- All of the PK components that are inherited from related entities will be properly inserted into each entity that requires the use of such inherited PK attribute(s).
- The FKs will always inherit the attribute characteristics from the PKs to which they point. That means you will never see an "incompatible data type" error message when you try to implement the design.
- The software will automatically check for inconsistent relationships, circular relationships, and incorrectly defined relationships. Therefore, you will, in effect, have a built-in design quality control feature.

Most CASE tools provide such PK/FK design services, and you should make use of those services when they are available. After all, the objective is to produce a clean design that can be implemented successfully. Once you know that the design contains no logical or implementation-level flaws, you can add the remaining attributes.

Naturally, if you are doing the preliminary design using pencil and paper—you are using a design tool that does not provide the just-described services—you will have to write all of the PK attributes and FK attributes at the entity level so you can see what the implementation implications are. In effect, you will be serving the same role as the "update foreign keys" function you'll find in most advanced database design software.

As you examine Table C.3, note that the DEPT_CODE attribute has been added, which lets the CLD track which departments use the Lab facilities. That information is important because departments share the Lab's budgeting equation; departments using the Lab more frequently contribute more to its operation than departments using the Lab less frequently. Therefore, the ability to count Lab use by department code is important. Although the DEPT_CODE is clearly a foreign key to a DEPARTMENT entity, there is no information requirement at this time for more detailed departmental data. Therefore, DEPARTMENT will not be included in this design, and the DEPT_CODE in the USER entity has not been designated as an FK at this point.

Note

The USER entity was initially created to prototype the system and to make sure that a working database could be supplied within this appendix. In the USER table, USER_ID, DEPT_CODE, and USER_TYPE can be used to summarize Lab usage for budgeting purposes.

As in most real-world database designs, the University Computer Lab Management System's (UCLMS) actual user data are part of an existing external database. That external database is controlled and maintained by the university. Its data entry procedures and structures are different from those addressed in the UCLMS design. For example, the Lab's user data are entered through a magnetic card reader that targets many more variables than are included here. Adding those variables to the design shown in this appendix would not add insight into the crucial design verification process on which you want to focus. For the same reason, the entities DEPARTMENT and COLLEGE were not included. (The inclusion of the DEPT_CODE in USER is sufficient to track Lab usage without having to access departmental and college details.)

Also, the data used in this appendix constitute a very small subset of the actual data. For example, the real system currently records over 30,000 Lab-use entries per semester, and that tally is growing rapidly. Finally, to conform to privacy requirements, all data values have been simulated.

Note that the USER table structure produces some redundancy. For example, USER_CLASS clearly determines the USER_TYPE If you know that USER_CLASS = UG, you also know that USER_TYPE is Stu. On the other hand, USER_TYPE is not a determinant of USER_CLASS because Stu can mean either UG or GR. In any case, you now know that the table is in 2NF. To eliminate the 2NF condition, you could combine USER_TYPE and USER_CLASS into a single attribute represented by a string to portray Stu/UG, Stu/GR, Fac, and Staff. However, because the university requires a report that shows Lab usage summaries by faculty, staff, and students, the current table structure is desirable. Additionally, the report requires a breakdown by various student subcategories (graduate/undergraduate, male/female). Real-world database design often requires a trade-off between information efficiency and design purity. Some sample USER data are shown in Figure C.2.

As you examine the data shown in Figure C.2, you should note that when the USER_TYPE is Fac or Staff, the USER_CLASS is also Fac or Staff. That duplication serves reporting requirements well because it enables you to generate USER_CLASS

summaries easily. Finally, note that hyphens have been used in the USER_ID data. Social Security numbers are read more easily when hyphens are used, and the cost of including hyphens in the string is only 2 bytes per entry. Data storage is cheap and getting cheaper, so the extra 2 bytes per USER_ID entry do not create much of a burden. On the other hand, if the data search is keyed to the User ID, the search speed is enhanced when the dashes are not included in an alphanumeric attribute. Modern database systems do provide the designer with an option to use input masks for presentation purposes (9-99-9999, rather than 9999999) *without* storing the dashes in the database table. As always, database professionals are expected to use sound judgment to balance competing requirements.

FIGURE C.2 SAMPLE USER DATA

USER_ID	DEPT_CODE	USER_TYPE	USER_CLASS	USER_GENDE
1-11-1111	CIS	Fac	Fac	F
1-11-1112	CIS	Fac	Fac	M
1-11-1113	ACCT	Staff	Staff	F
1-11-1114	ACCT	Fac	Fac	F
1-11-1115	BIOL	Staff	Staff	M
1-11-1116	MKT/MGT	Fac	Fac	M
1-11-1117	CIS	Fac	Fac	M
1-11-1118	SOC	Stu	GR	F
1-11-1119	ACCT	Fac	Fac	F
1-11-1120	ACCT	Fac	Fac	F
1-11-1121	CIS	Fac	Fac	F
1-11-1122	CIS	Stu	UG	F
1-11-1123	CIS	Stu	UG	F

Following the module layout in Table C.2, the LOG entity is examined next, represented by the LOG table. Each time a USER accesses the Lab facilities, that user's identification is read into the log by one of two magnetic card readers.

Note

To prototype the system and keep the database self-contained, the USER entry procedure has been modified. For example, if a USER_ID does not match a record in the USER table, the USER table is updated by the addition of the new user. Keep in mind that the real system must provide security, so it must refuse entry to a user whose identification does not match an existing record in the externally managed student database.

The LOG entity details are shown in Table C.4.

TABLE C.4

THE LOG ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
LOG_DATE	Log-in (system) date		PK	
LOG_TIME	Log-in (system) time		PK	
LOG_READER	Magnetic card reader number		PK	
USER_ID	User identification (ID)		FK	USER

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

Table C.4 shows a composite primary key based on LOG_DATE, LOG_TIME, and LOG_READER. The assumption is that it is impossible for any magnetic card reader to record the same time (to the second) for more than one entry because it takes a few seconds to complete the magnetic card swipe. If LOG_READER is not part of the primary key, it is possible that two different card readers swiped at the same time would record the same time and, thus, violate the entity integrity requirement.

The LOG's sample data are shown in Figure C.3.

FIGURE C.3 SAMPLE LOG DATA

LOG_DATE	LOG_TIME	LOG_READER	USER_ID
17-Jan-2014	2:24:32 PM	1	1-11-1138
17-Jan-2014	2:24:39 PM	2	1-11-1125
17-Jan-2014	2:25:41 PM	2	1-11-1123
17-Jan-2014	2:25:44 PM	1	1-11-1129
17-Jan-2014	2:25:58 PM	2	1-11-1143
17-Jan-2014	2:26:03 PM	1	1-11-1115
17-Jan-2014	2:26:28 PM	2	1-11-1112
17-Jan-2014	2:29:19 PM	1	1-11-1136
17-Jan-2014	2:30:35 PM	1	1-11-1114
17-Jan-2014	2:32:09 PM	1	1-11-1130

As you examine the LOG data in Figure C.3, it might occur to you that adding an attribute such as LOG_ID would eliminate the need for a composite primary key. You might also argue that such a LOG_ID attribute would be redundant because the combination of LOG_DATE, LOG_TIME, and LOG_READER already performs the primary key function. That's true enough. But the existence of a candidate key is not structurally damaging, and a single-attribute primary key decreases system overhead by diminishing the primary key index requirements. Here is yet another example of the many decisions that the database designer must make. (As you can tell, the decision was made to stick with the composite primary key.) Try to answer questions such as these: Is the attribute necessary or useful? If it is useful, what is the

cost of creating and using it? What function does it have that cannot be well served by other attributes? As you can see, database design requires the use of professional judgment.

The CLD manages a group of lab assistants (LAs). The University's policy is to limit the Lab staffing for daily operations to graduate assistants (GAs), student workers (SW), and work study students (WS). The GAs are limited to a 20-hour work week, the SWs are limited to a 10-hour work week, and the WSs are limited to a 4-hour work week. The LA attributes are summarized in Table C.5.

TABLE C.5

THE LAB_ASSISTANT ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
LA_ID	Lab assistant identification		PK	
LA_NAME	Lab assistant name	C		
LA_PHONE	Lab assistant campus phone	C		
LA_SEMESTER	Most recent working semester	C		
LA_TYPE	Lab assistant classification: GA = Graduate assistant SW = Student worker WS = Work study student			
LA_HIRE_DATE	Date hired			

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

As you examine Table C.5, remember that information requirements often determine the degree to which composite entities are decomposed. For example, it is likely that the CLD will want to generate a phone list to simplify contacting LAs. Therefore, the decomposition of the LA_NAME into its component first name, last name, and initial is appropriate. On the other hand, it is unlikely that much will be gained by decomposing a phone number such as 4142345 into the 414 exchange number and its 2345 extension. Although information needs are generally better served by greater atomism, the needless proliferation of attributes increases complexity without generating appropriate return benefits. Similarly, the LA_SEMESTER is expressed by entries such as SPRING12 to indicate the most recent semester during which the LA was working. End-user reporting requirements indicate that little would be gained by decomposing that entry into the SPRING semester designation and the 12 year designation.

A few of the LAB_ASSISTANT records are shown in Figure C.4 to illustrate the data entries.

FIGURE C.4 SAMPLE LAB_ASSISTANT DATA

LA_ID	LA_LNAME	LA_FNAME	LA_INITIAL	LA_PHONE	LA_SEMESTER	LA_TYPE	LA_HIRE_DATE
1-11-2001	Jones	James	C	4123234	SPRING18	GA	07-Jan-2018
1-11-2002	Smith	Anne	G	4123245	SPRING18	SW	10-Jun-2018
1-11-2003	Hernandez	Maria	M	4123245	SPRING18	GA	07-Jan-2018
1-11-2004	Inum	Idong	J	4123234	SPRING18	WS	07-Jan-2018
1-11-2005	Jamerson	George	D	4126789	SPRING18	SW	13-Jul-2017
1-11-2006	Patterson	Herman	W	4127890	SPRING18	GA	07-Jan-2018
1-11-2007	Troyana	Emily	H	4121121	FALL17	SW	04-Jan-2018
1-11-2008	Evans	Peter	G	4123234	SPRING18	GA	07-Jan-2018
1-11-2009	Vann	Evangeline	D	4121121	SPRING18	SW	07-Jan-2018

Note

The LA_SEMESTER attribute enables the CLD to check whether an LA is available for assignment during the current semester. Some LAs work in the Lab one semester, perform cooperative duties the next semester, and return to their Lab assignment the following semester. Thus, a FALL17 entry would indicate that the LA's last work assignment was during the Fall of 2017. Because LAB_ASSISTANT records are purged (and archived) after the LA's graduation, termination, or resignation, an LA_SEMESTER designation other than the current semester's indicates the last semester during which the LA worked.

To keep track of the LA work schedules, the CLD keeps a scheduling sheet like the one shown in Table C.6. Table C.6 defines the attributes of the WORK_SCHEDULE entity, which is shown in Table C.7.

TABLE C.6

THE LAB ASSISTANT WORK-SCHEDULING SHEET

TIME SLOT		MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY	SUNDAY
06–08	1.	Jones (GA)	Thomas (GA)	Gabril (GA)	Evans (GA)	Hernando (GA)		
	2.	Jamerson (WS)	Chung (SW)	Chung (SW)	Tabrin (GA)	Mustava (GA)		
	3.	Hernando (GA)	Womack (SW)	Thomas (GA)	Jones (GA)	Tabrin (GA)		
	4.	Vann (SW)	Dalton (SW)	Smith, C (SW)	Smith, C (SW)	Rommel (SW)		
08–10	1.	Jones (GA)	Thomas (GA)	Gabril (GA)	Evans (GA)	Hernando (GA)		
	2.	Hernandez (GA)	Porter (WS)	Chung (SW)	Tabrin (GA)	Mustava (GA)		
	3.	Jamerson (WS)	Womack (SW)	Thomas (GA)	Dalton (SW)	Tabrin (GA)		
	4.	Vann (SW)	Chung (SW)	Smith, C (SW)	Smith, C (SW)	Rommel (SW)		

TABLE C.6

THE LAB ASSISTANT WORK-SCHEDULING SHEET (CONTINUED)

TIME SLOT		MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY	SUNDAY
10–12	1.	Hernandez (GA)	Jones (GA)	Gabril (GA)	Jones (GA)	Hernando (GA)		
	2.	Troyana (SW)	Porter (WS)	Troyana (SW)	Tabrin (GA)	Antony (SW)		
	3.	Jamerson (WS)	Willis (GA)	Willis (GA)	Antony (SW)	Tabrin (GA)		
	4.	Morris (SW)	Womack (SW)	Antony (SW)	Dalton (SW)	Rommel (SW)		
12–02	1.	Evans (GA)	Jones (GA)	Gabril (GA)	Jones (GA)	Kallen (GA)	Jones (GA)	Tabrin (GA)
	2.	Trayana (SW)	Vann (SW)	Troyana (SW)	Tabrin (GA)	Evans (GA)	Dalton (SW)	Mustava (GA)
	3.	Willis (GA)	Willis (GA)	Willis (GA)	Antony (SW)	Mustava (GA)		
	4.	Highlon (SW)	Womack (SW)	Antony (SW)	Smith, C (SW)	Rostav (SW)		
02–04	1.	Evans (GA)	Hernando (GA)	Kallen (GA)	Kallen (GA)	Kallen (GA)	Gabril (GA)	Tabrin (GA)
	2.	Inum (WS)	Vann (SW)	Troyana (SW)	Tabrin (GA)	Willis (GA)	Dalton (SW)	Mustava (GA)
	3.	Jones (GA)	Morris (SW)	Morris (SW)	Mustava (GA)	Mustava (GA)	Batey (SW)	Kadin (SW)
	4.	Highlon (SW)	Womack (SW)	Chung (SW)	Jones (WS)	Batey (WS)	Avery (SW)	
04–06	1.	Evans (GA)	Hernando (GA)	Kallen (GA)	Kallen (GA)	Kallen (GA)	Gabril (GA)	Sorals (GA)
	2.	Kadin (SW)	Vann (SW)	Sorals (GA)	Thomas (GA)	Willis (GA)	Dalton (SW)	Mustava (GA)
	3.	Winston (SW)	Morris (SW)	Morris (SW)	Jones, A (WS)	Jones, A (SW)	Batey (SW)	
	4.	Rostav (SW)	Avery (SW)	Avery (SW)	Highlon (SW)	Jones (GA)	Rommel (SW)	
06–08	1.	Evans (GA)	Hernando (GA)	Kallen (GA)	Kallen (GA)	Kallen (GA)	Gabril (GA)	Sorals (GA)
	2.	Kadin (SW)	Thomas (GA)	Sorals (GA)	Sorals (GA)	Willis (GA)	Aaron (SW)	Mustava (GA)
	3.	Winston (SW)	Avery (SW)	Thomas (GA)	Jones, A (WS)	Aaron (SW)	Rommel (SW)	
	4.	Rostav (SW)	Winston (SW)	Avery (SW)	Kadin (SW)	Batey (SW)		
08–10	1.	Evans (GA)	Hernando (GA)	Kallen (GA)	Sorals (GA)	Kallen (GA)	Gabril (GA)	Sorals (GA)
	2.	Kadin (SW)	Thomas (GA)	Sorals (GA)	Thomas (GA)	Willis (GA)	Aaron (SW)	Witte (SW)
	3.	Highlon (SW)	Avery (SW)	Thomas (GA)	Jones, A (WS)	Aaron (SW)	Winston (SW)	
	4.	Rostav (SW)	Winston (SW)	Winston (SW)	Rostav (SW)	Rommel (SW)		
10–12	1.	Casey (GA)	Casey (GA)	Casey (GA)	Casey (GA)	Casey (GA)	Gabril (GA)	Sorals (GA)
	2.	Thompson (SW)	Thompson (SW)	Thompson (SW)	Karpov (SW)	Karpov (SW)	Witte (SW)	Witte (SW)
	3.							
	4.							

TABLE C.7

THE WORK_SCHEDULE ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
SCHED_SEMESTER	Semester ID	C	PK	
SCHED_WEEKDAY	Schedule weekday		PK	
SCHED_IN	Time slot start		PK	
SCHED_OUT	Time slot end			
SCHED_SLOT	Weekday slot number (Value range 1–4)		PK	
LA_ID	Lab assistant ID		FK	LAB_ASSISTANT
* The attribute type may be Composite (C), Derived (D), or Multivalued (M).				

The WORK_SCHEDULE's primary key is a composite key, created by the combination of SCHED_SEMESTER, SCHED_WEEKDAY, SCHED_IN, and SCHED_SLOT. The requirement is that any given LA cannot have two of the same scheduled starting times for the same weekday. For example, an LA cannot have two starting times of 10 a.m. on Monday. The SCHED_IN and SCHED_OUT entries are based on a 24-hour time clock and range from 0600 to 2400.

A second requirement is that there can be no more than four lab assistants assigned to work during the same time slot. Unfortunately, the initial primary key selection doesn't fit those requirements well. Fortunately, the current design shortcomings can be fixed by adopting a three-pronged approach, as follows:

1. Create a primary key composed of SCHED_SEMESTER, SCHED_WEEKDAY, SCHED_IN, and SCHED_SLOT.
2. Create a unique index based on SCHED_SEMESTER, SCHED_WEEKDAY, SCHED_IN, and LA_ID.
3. Create a data validation rule to specify that the SCHED_SLOT values must be 1, 2, 3, or 4.

Option 1 can be implemented by declaring the PK components when the table is created. Option 2 can be implemented through the CREATE INDEX command. Option 3 requires the use of application code to enforce the validity of SCHED_SLOT values. If you are using Oracle, you can implement option 3 by using a trigger. (If you are using MS Access, you can use a data validation rule to implement option 3.) The implementation of options 1 and 3 will ensure that a maximum of four lab assistants will be working at any given scheduled time. Option 2 ensures that no lab assistant appears more than once in any given weekday/time combination.

Sample WORK_SCHEDULE data entries are shown in Figure C.5.

FIGURE C.5 SAMPLE WORK_SCHEDULE DATA

SCHED_SEMESTER	SCHED_WEEKDAY	SCHED_IN	SCHED_OUT	SCHED_SLOT	LA_ID
SPRING18	Friday	06:00	08:00	1	1-11-2003
SPRING18	Friday	08:00	10:00	1	1-11-2003
SPRING18	Friday	10:00	12:00	1	1-11-2003
SPRING18	Friday	14:00	16:00	2	1-11-2018
SPRING18	Friday	16:00	18:00	2	1-11-2018
SPRING18	Friday	16:00	18:00	4	1-11-2001
SPRING18	Friday	18:00	20:00	2	1-11-2018
SPRING18	Friday	20:00	22:00	2	1-11-2018
SPRING18	Monday	06:00	08:00	1	1-11-2001
SPRING18	Monday	06:00	08:00	2	1-11-2005
SPRING18	Monday	06:00	08:00	3	1-11-2003

Keep in mind that the order in which the WORK_SCHEDULE attributes are stored in the table is immaterial to the relational model. However, your DBMS is very likely to index the table according to its primary key. In this case, the indexing order begins with SCHED_SEMESTER and, within that order, moves to index SCHED_WEEKDAY, SCHED_IN, and SCHED_SLOT. The data displayed in Figure C.5 conform to such an indexing order. In that case, the primary key ensures an order that is independent of the LA_ID or the assistant's last name. Once again, you see a condition that is inconsequential to the relational model, but that eventually affects how the designer evaluates the design for implementation and applications development.

The HOURS_WORKED structure, shown in Table C.8, tracks the number of hours worked by each LA during a two-week pay period.

TABLE C.8

THE HOURS_WORKED ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
LA_ID	Lab assistant ID		PK, FK	LAB-ASSISTANT
HW_SEMESTER	Semester designation	C	PK	
HW_DATE	Work period ending data		PK	
HW_HOURS_WORKED	Total hours worked			

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

The HOURS_WORKED entity's primary key is a composite key and consists of LA_ID, HW_SEMESTER, and HW_DATC. The inclusion of HW_DATE as part of the primary key is required to maintain entity integrity because the combination of LA_ID and HW_SEMESTER can produce many occurrences. (Each LA works many weeks within the semester.) Also note that the HW_HOURS_WORKED attribute represents the total hours worked by the LA during the pay period *and is entered manually by the end user*. (In this case, the HW_HOURS_WORKED attribute is *not* a derived attribute. Note that there are no attributes in this table from which the HW_HOURS_WORKED attribute

could be computed. If the HOURS_WORKED entity had included HW_TIME_IN and HW_TIME_OUT attributes, the HW_HOURS_WORKED attribute values could have been calculated from the other two time attributes and would, *in that case*, have been a derived attribute.) The HOURS_WORKED data form the basis for payroll applications. A few sample data entries are shown in Figure C.6.

FIGURE C.6 SAMPLE HOURS_WORKED DATA

LA_ID	HW_SEMESTER	HW_DATE	HW_HOURS_WORKED
1-11-2001	SPRING18	15-Jan-2018	40
1-11-2003	SPRING18	15-Jan-2018	40
1-11-2005	SPRING18	15-Jan-2018	8
1-11-2007	SPRING18	15-Jan-2018	20
1-11-2009	SPRING18	15-Jan-2018	20
1-11-2017	SPRING18	15-Jan-2018	20
1-11-2018	SPRING18	15-Jan-2018	40

Although the Lab is used mostly by students doing their assignments, sections of the Lab may be reserved by faculty members for teaching purposes or by staff members for hardware and software maintenance and updates. To enable the system to handle those reservations, the initial RESERVATION Entity structure was developed, as shown in Table C.9.

TABLE C.9

THE RESERVATION ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
RES_DATE	Reservation date		PK	
USER_ID	User ID (faculty/staff only)		PK, FK	USER
RES_DATES_RESVD	Date(s) reserved	M	PK	
RES_TIME_IN	Time(s) in reserved	M		
RES_TIME_OUT	Time(s) out reserved	M		
RES_USERS	Number of users during the scheduled reserved time	M		
LA_ID	Lab assistant who entered the reservation		FK	LAB_ASSISTANT

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

As you examine the RESERVATION entity's structure, note its many multivalued attributes. For example, a faculty member may reserve the Lab for several dates and, within those dates, several times per day, each time for a different number of users. Such multivalued attributes are guaranteed to create problems at the implementation stage. For example, how many RES_DATES_RESVD derivative attributes (RES_DATES_RESVD1, RES_DATES_RESVD2, RES_DATES_RESVD3) should you reserve for storing the reservation dates? When you create too many, you have many nulls. When you create too few, reservations are limited by the available attributes. And if you want to allow

additional reservation dates later, you'll have to modify the table structure. The problem is magnified by the fact that, for each reserved date, there are many possible reserved times. The number of such derivative attributes can multiply dramatically. (The authors once did a database audit in which one of the tables contained 114 attributes—and the list was growing. Not surprisingly, the database did not function very well.)

The RESERVATION structure can be split into two tables in a 1:M relationship. The first of those two tables, still named RESERVATION, represents the "1" side. Its structure is shown in Table C.10.

TABLE C.10

THE REVISED RESERVATION ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
RES_ID	Reservation ID		PK	
RES_DATE	Date on which the reservation was made			
USER_ID	User ID (faculty/staff only)		FK	USER
LA_ID	Lab assistant who entered the reservation		FK	LAB_ASSISTANT

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

Some sample RESERVATION data are shown in Figure C.7.

FIGURE C.7 SAMPLE RESERVATION DATA

RES_ID	RES_DATE	USER_ID	LA_ID
523	25-Jan-2018	1-11-1111	1-11-2003
524	25-Jan-2018	1-11-1112	1-11-2003
525	26-Jan-2018	1-11-1120	1-11-2008

This new RESERVATION structure works much better. Each time an LA records a set of reservations, the date on which the reservations are made is recorded in RES_DATC. You can also track who (USER_ID) made the reservation and who (LA_ID) recorded it. The multiple occurrences of the reservations are then handled by the "Many" side in a table named RES_SLOT, whose structure is shown in Table C.11.

TABLE C.11

THE RES_SLOT (WEAK) ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
RES_ID	Reservation ID		PK, FK	RESERVATION
RSLOT_DATE	Date reserved		PK	
RSLOT_TIME_IN	Reservation time in		PK	
RSLOT_TIME_OUT	Reservation time out			
RSLOT_USERS	Number of users during the scheduled reserved time			
RSLOT_LAB_SECTION	Reserved section of the lab		PK	

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

As you examine the structure in Table C.11, note that the participation of the RSLOT_LAB_SECTION makes it possible to have two reservations on the same date and time when the reservations involve different sections of the Lab. Also note that RES_SLOT is a weak entity because it is existence-dependent on RESERVATION and because one of its primary key components, RES_ID, is inherited from the RESERVATION entity.

Figure C.8 shows some sample data to illustrate the reservation process.

FIGURE C.8 SAMPLE RES_SLOT DATA

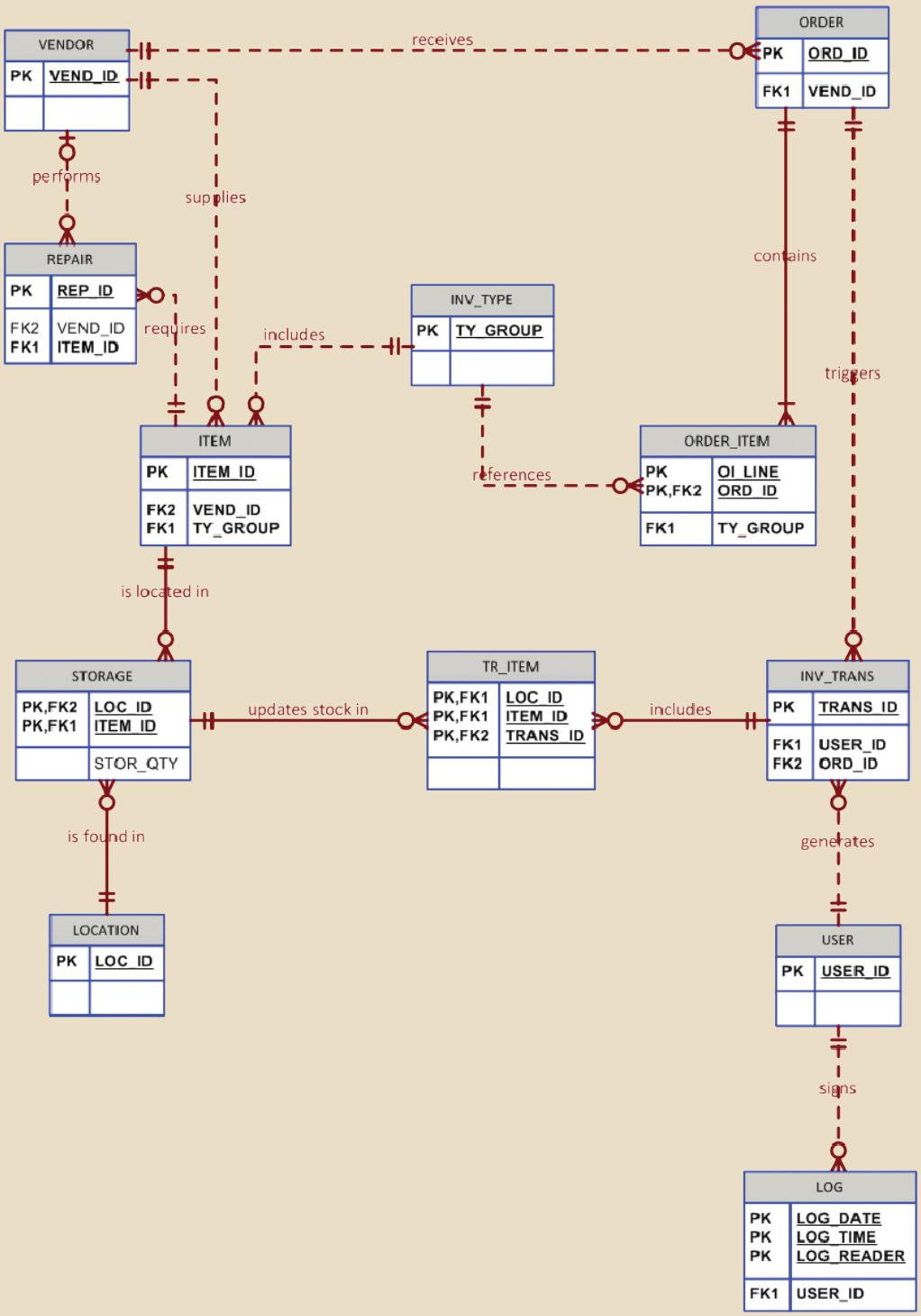
RES_ID	RSLOT_DATE	RSLOT_TIME_IN	RSLOT_SECTION	RSLOT_TIME_OUT	RSLOT_USERS
523	03-Feb-2018	8:00 AM	A	9:50 AM	23
523	10-Feb-2018	8:00 AM	A	9:50 AM	23
524	04-Feb-2018	2:00 PM	C	3:15 PM	35
525	03-Feb-2018	6:00 PM	A	8:40 PM	18
525	07-Feb-2018	10:00 AM	A	10:50 AM	24
525	10-Feb-2018	6:00 PM	B	8:40 PM	18

By examining the sample data in Figure C.8, you can easily trace the reservation process when you keep in mind the 1:M relationship between RESERVATION and RES_SLOT. Note, for example, that on January 25, 2018 (see the RESERVATION data in Figure C.7), user 1-11-1111 made reservations (see the RES_SLOT data in Figure C.8) for 23 users for February 3, 2018 from 8:00 a.m.–9:50 a.m. in Section A of the Lab and for 23 users for February 10, 2018 from 8:00 a.m.–9:50 a.m. in Section A of the Lab.

C-2b The Inventory Management Module

To help track the Inventory Management System's detailed development process, it is useful to look at its ER components, shown in Figure C.9. Using that illustration as your guide, you will find it much easier to understand the revision process. Refer to Figure C.9 often as the Inventory Management System's entities and their attributes are developed.

FIGURE C.9 THE INVENTORY MANAGEMENT MODULE'S ER SEGMENT



As you examine the ER segment in Figure C.9, you might wonder why the WITHDRAW and CHECK_OUT entities used in the initial ER diagram in Appendix B and in Table C.2 in this appendix do not appear. You will also see that a few new entities, such as INV_TRANS, have been added. Those changes are part of the ER data model verification process, and they will be discussed in detail later in this section. Also, the USER entity, which is not an explicit part of the Inventory Management System and was already discussed in Section C-2a, is the "connector" between the two entity segments. Therefore, although it will not be discussed further, the inclusion of USER makes sense in this segment, too.

The INV_TYPE entity performs an important role in the Inventory Management module. Its presence makes it easy for the CLD to generate detailed inventory summaries. (For example, how many boxes of 8.5" × 11" single-sheet paper are in stock? How many laser printers are available? How many boxes of writable CDs are on hand?) To understand the INV_TYPE's function, you must first understand the role of a classification hierarchy, as shown in Table C.12.

TABLE C.12

AN INVENTORY CLASSIFICATION HIERARCHY

GROUP	CATEGORY	CLASS	TYPE	SUBTYPE
HWPCDTP3	Hardware (HW)	Personal computer (PC)	Desktop (DT)	Intel Core i7
HWPCLTP4	Hardware (HW)	Personal computer (PC)	Laptop (LT)	Intel Core i7
HWPCLTCE	Hardware (HW)	Personal computer (PC)	Laptop (LT)	Intel Core i5
HWPRLSBL	Hardware (HW)	Printer (PR)	Laser (LS)	Black (BL)
HWPRIJCO	Hardware (HW)	Printer (PR)	Ink-jet (IJ)	Color (CO)
SUPPSS11	Supply (SU)	Paper (PP)	Single-sheet (SS)	11-inch
HWEXVIXX	Hardware (HW)	Expansion board (EX)	Video (VI)	XX
SWDBXXXX	Software (SW)	Database (DB)	XX	XX

As you examine the classification hierarchy in Table C.12, note that three categories have been created: hardware, software, and supply. Also note that each group code precisely describes the inventory type discussed here. For example, the first group code, HWPCDTP3, describes the category "hardware" (HW), the class "personal computer" (PC), the type "desktop" (DT), and the subtype Intel Core i7. Some group codes, such as the last one in Table C.12, do not specifically identify type and subtype; rather, they use the code XX to indicate that no specification was made. You will see later that the classification hierarchy group codes can be used as primary keys to define the INV_TYPE rows and that the category, class, type, and subtype components will be stored as separate attributes to enhance the inventory reporting capabilities.

The classification hierarchy can also be presented as a tree diagram, as shown in Figure C.10.

The classification hierarchy illustrated in Table C.12 and Figure C.10 is reflected in the INV_TYPE structure shown in Table C.13 and in the INV_TYPE's sample data illustrated in Figure C.11.

FIGURE C.10 THE INV_TYPE CLASSIFICATION HIERARCHY AS A TREE DIAGRAM

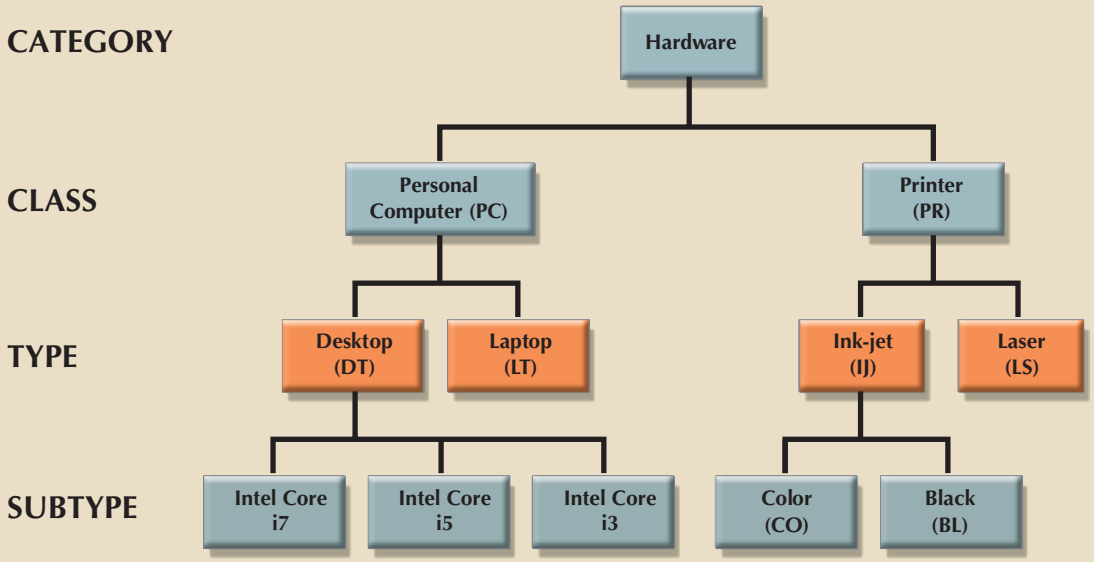


TABLE C.13

THE INV_TYPE ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
TY_GROUP	Inventory group code	C	PK	
TY_CATEGORY	Inventory category			
TY_CLASS	Inventory class			
TY_TYPE	Inventory type			
TY_SUBTYPE	Inventory subtype			
TY_DESCRIPTION	Group description			
TY_UNIT	Unit of measurement (box, ream, and so on)			

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

As you examine the INV_TYPE structure in Table C.13, note that the INV_TYPE uses a single-attribute primary key (TY_GROUP), which renders the system faster and more efficient in the query mode. Although a composite primary key could have been created by combining TY_CATEGORY, TY_CLASS, TY_TYPE, and TY_SUBTYPE, such a multiple-attribute primary key would produce a more complex pointer system for the DBMS, thus slowing down the system. Yet the decomposition of TY_GROUP into TY_CATEGORY, TY_CLASS, TY_TYPE, and TY_SUBTYPE allows a greater variety of reporting summaries to be performed easily while having the benefit of a single-attribute primary key. (Remember that information requirements help drive the design process. Table C.13 and its sample data in Figure C.11 provide an appropriate illustration of that point.)

Although the INV_TYPE provides much flexibility in terms of inventory summary statements, you still must be able to reference specific units. For example, it is useful to know that there are 217 computers in inventory, but you must also be able to track each computer that was installed in a professor's office or in the Lab. The relationship between INV_TYPE and ITEM (review the ER segment in Figure C.9) provides that capability. The ITEM's entity structure is shown in Table C.14.

FIGURE C.11 SAMPLE INV_TYPE DATA

TY_GROUP	TY_CATEGORY	TY_CLASS	TY_TYPE	TY_SUBTYPE	TY_DESCRIPTION	TY_UNIT
HWBAPNXX	HW	BA	PN	XX	Bar Code Reader, Pen Type	UN
HWCAMO12	HW	CA	MO	12	External Modem Wire, 12'	UN
HWCASP08	HW	CA	SP	08	Serial Printer Cable, 8-pin	UN
HWEXHDID	HW	EX	HD	ID	Expansion Board-IDE HD ctrl.	UN
HWEXHDMF	HW	EX	HD	MF	Expansion Board-MFM HD ctrl.	UN
HWEXMEXX	HW	EX	ME	XX	Expansion Board, Memory	UN
HWEXVIXX	HW	EX	VI	XX	Expansion Board, Video	UN
HWMSXXXX	HW	MS	XX	XX	Hardware, Miscellaneous	UN
HWNCETCX	HW	NC	ET	CX	Ethernet NIC, Coax	UN
HWNCETTP	HW	NC	ET	TP	Ethernet NIC, Twisted Pair	UN
HWNCTR4M	HW	NC	TR	4M	Token Ring, NIC 4M	UN
HWPCDT48	HW	PC	DT	48	Tier 2 desktop	UN
HWPCDTCE	HW	PC	DT	CE	Tier 3 desktop computer	UN
HWPCDTM2	HW	PC	DT	M2	Tier 1 Apple computer	UN
HWPCDTP2	HW	PC	DT	P2	Desktop PC, tier 3	UN
HWPCDTP3	HW	PC	DT	P3	Desktop PC, tier 2	UN
HWPCDTP4	HW	PC	DT	P4	Desktop PC, tier 1	UN

TABLE C.14

THE ITEM ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
ITEM_ID	Item identification code		PK	
TY_GROUP	Inventory group code		FK	INV_TYPE
ITEM_UNIV_ID	University inventory ID			
ITEM_SERIAL_NUM	Item (manufacturer's) serial number			
ITEM_DESCRIPTION	Item description			
ITEM_QTY	Total quantity on hand at all locations	D		
VEND_ID	Original vendor code		FK	VENDOR
ITEM_STATUS	Item status: 1 = available 2 = under repair 3 = out of order 4 = checked out			
ITEM_BUY_DATE	Purchase date			

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

Let's examine the behavior of the ITEM entity's attributes according to the three main inventory types:

- *Hardware.* If 20 computers are bought, each one will be assigned an ITEM_ID, an ITEM_UNIV_ID, and a manufacturer's ITEM_SERIAL_NUM, thus generating 20 records.
- *Supply.* If 20 boxes of laser printer paper are bought, only one record is generated because there is no need to identify each box individually. Therefore, in the case of the boxes, no university ID number is required, and the ITEM_UNIV_ID will be null. In addition, a box of paper would not have a serial number, so the ITEM_SERIAL_NUM will be null. However, because the CLD must be able to track the boxes, it is necessary to have ITEM_ID as the primary key. Naturally, you may use special codes, such as 00000, for the ITEM_UNIV_ID and the ITEM_SERIAL_NUM to avoid the use of nulls.
- *Software.* If a license is bought for 180 software copies, only one license record will exist in the ITEM entity. Individual installations can be tracked through the use of the STORAGE entity, shown in Table C.15.

TABLE C.15

THE STORAGE ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
LOC_ID	Location ID		PK, FK	LOCATION
ITEM_ID	Item ID		PK, FK	ITEM
STOR_QTY	Quantity stored at this location			

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

Note also that the ITEM_QTY in Table C.14 is a derived attribute because it sums the quantity on hand *at all locations*. Keep in mind that design purity would dictate the elimination of a derived attribute. Yet its presence here reflects the end user's desire for simple and quick answers to such ad hoc questions as "How many boxes of 8.5" × 11" paper do we have at all locations?" Because the derived attribute ITEM_QTY is computed and written into the ITEM table by the application software any time there is a transaction involving the inventory, there is no chance of creating data anomalies through its inclusion.

Some sample ITEM data are shown in Figure C.12.

FIGURE C.12 SAMPLE ITEM DATA

ITEM_ID	TY_GROUP	ITEM_UNIV_ID	ITEM_SERIAL_NUM	ITEM_QTY	VEND_ID	ITEM_STATUS	ITEM_BUY_DATE
1212004	HWEXMEXX	00000	00000	2	PCJUN	2	10-May-2017
2088723	HWPCLTP3	3009765	CX-5437688	1	PCJUN	4	12-May-2017
2879954	SUPPCO11	00000	00000	84	PCPAL	1	12-May-2017
2998950	SWDBXXXX	00000	00000	20	COMPU	1	15-May-2017
3045887	HWEXVIXX	3422012	BBF-985643	1	COMPU	1	15-May-2017
3154567	SUPPSS11	00000	00000	121	PCPAL	1	15-May-2017
3210946	HWEXHDID	3215457	12Q31223D9	1	PCJUN	1	16-May-2017
3212345	HWNCETCX	3245367	TR/3255675	1	PCJUN	1	16-May-2017

It is necessary to be able to locate an item in inventory at any given time. Therefore, the item's storage location must be known. The STORAGE entity, shown in Table C.15, plays an important role.

The STORAGE sample data are shown in Figure C.13.

FIGURE C.13 SAMPLE STORAGE DATA

LOC_ID	ITEM_ID	STOR_QTY
KOM106-1	3154567	19
KOM106-1	4238130	15
KOM106-1	4238132	10
KOM203E-1	2088723	1
KOM203E-1	3212345	1
KOM203E-1	4456789	1
KOM203E-1	4456791	1
KOM203E-2	4562397	1
KOM205B-1	2879954	10
KOM205B-1	3154567	25
KOM245A-1	2879954	35
KOM245A-1	4238131	18
KOM245A-1	4238132	10
KOM245A-1	4451236	1
KOM245A-2	3154567	52
KOM245A-2	4009212	1
KOM245A-2	4112151	1
KOM245A-2	4238132	12
KOM245B-1	3154567	5
KOM245B-1	4228753	1
KOM245B-1	4358255	1
KOM245B-1	4358258	1
KOM245B-1	4451235	1
KOM245B-2	3154567	8

By tracing the ITEM_ID in STORAGE (see Figure C.13) to the ITEM_ID in ITEM (see Figure C.12) and then to TY_GROUP = SUPPSS11 in INV_TYPE (see Figure C.11), you can determine that the first record in STORAGE shows 19 boxes of single-sheet paper (ITEM_ID = 3154567) located in KOM106-1. An additional 25 boxes of single-sheet paper are located in KOM205B-1, 52 boxes are located in KOM245A-2, and 5 boxes are located in KOM245B-1.

The storage location details are stored in the LOCATION entity, shown in Table C.16. By reviewing the ER segment in Figure C.9, you will note that there is a 1:M relationship between LOCATION and STORAGE.

TABLE C.16

THE LOCATION ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
LOC_ID	Location ID		PK	
LOC_DESCRIPTION	Location description (examples: faculty office, classroom, cabinet, and so on)			

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

The LOCATION entity's sample data are shown in Figure C.14.

FIGURE C.14 SAMPLE LOCATION DATA

LOC_ID	LOC_DESCRIPTION
KOM106-1	CIS Department office
KOM106-2	CIS Department office
KOM200-1	Classroom
KOM200-2	Classroom
KOM200-3	Classroom
KOM200-4	Classroom
KOM203E-1	Faculty office
KOM203E-2	Faculty office
KOM205A-1	Hall storage closet
KOM205A-2	Hall storage closet
KOM205A-3	Hall storage closet
KOM205A-4	Hall storage closet
KOM205B-1	Hall storage closet
KOM205B-2	Hall storage closet
KOM245A-1	Computer lab, section A
KOM245A-2	Computer lab, section A
KOM245A-3	Computer lab, section A
KOM245A-4	Computer lab, section A
KOM245A-5	Computer lab, section A
KOM245B-1	Computer lab, section B
KOM245B-2	Computer lab, section B
KOM245B-3	Computer lab, section B

You can create as much detail as necessary in the LOCATION entity's LOC_DESCRIPTION. For example, you can specify bins, shelves, and other details within the storage location. In fact, if you were to use that design technique in a store or plant location, you might create a series of new attributes to specify section, aisle, shelf, and bin.

At this point, you are able to:

- Provide detailed descriptions of the items in the UCL's inventory.
- Provide inventory category summaries easily and efficiently.
- Determine the item status.
- Trace the items to their storage locations.

Because inventory tracking is very important, especially at auditing time, the emerging system is already showing considerable end-user potential. The remaining Inventory Management System will be built on that solid foundation.

Items in inventory are dynamic; that is, the items don't stay in inventory forever. In fact, some items, such as supplies, have a very limited inventory life. Paper, for example, doesn't last long in a computer lab. Software becomes obsolete, as does hardware. Hardware might break down and require repair, or it might require disposal. In fact, the need for repair produces some special database handling. An item sent out for repair still belongs to the Lab, but it is not available for use. Neither is an item that has broken down but has not yet been sent out for repair. Some items can be repaired in-house, and some items are returned to the vendor for replacement. In short, repair is an ever-present issue that requires tracking. Therefore, the REPAIR entity shown in Table C.17 plays an important role in the design.

TABLE C.17

THE REPAIR ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
REP_ID	Repair ID		PK	
ITEM_ID	Item identification code		FK	ITEM
REP_DATE	Date on which item needed repair			
REP_DESCRIPT	Problem description			
REP_STATUS	Repair status: 1 = in repair 2 = repaired 3 = returned to vendor 4 = out of order			
VEND_ID	Vendor code		FK	VENDOR
REP_REF	Reference number supplied by vendor			
REP_DATE_OUT	Date on which item was sent out			
REP_DATE_IN	Date on which item was returned			
REP_COST	Repair cost			

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

Several of the REPAIR attributes in Table C.17 require additional explanation.

- VEND_ID is an optional foreign key because the VENDOR does not enter the repair picture until the REP_STATUS = 3. If the repair status is 3, the VEND_ID must contain a valid VEND_ID entry—that is, one that matches a vendor in the VENDOR table. If you want to avoid nulls by using a "no vendor" code, the VENDOR table that the code references must contain a "no vendor" entry to maintain referential integrity.
- Because the cost is not known until the repair has been completed, REP_COST will be $0.00 until REP_STATUS has been changed to 2. Some repairs are done at the vendor's expense, so it is quite possible that the REP_COST will remain $0.00 when the REP_STATUS = 2. Also, the many repairs done in-house do not carry a charge, except the cost of replacement parts.
- REP_DESCRIPT cannot be null; there must be some description of the problem that occurred.

A few REPAIR records are shown in Figure C.15.

FIGURE C.15 SAMPLE REPAIR DATA

REP_ID	ITEM_ID	REP_DATE	REP_DESCRIPT	REP_STATUS	VEND_ID	REP_REF	REP_DATE_OUT	REP_DATE_IN	REP_COST
121	3045887	15-Jan-2018	Memory board failure	2	COMPU	RT-54566-674	17-Jan-2016	20-Jan-2018	0.00
122	5453234	17-Jan-2018	Token ring board failure	2	PCJUN	231156	17-Jan-2016	22-Jan-2018	54.82
123	3045887	05-Jan-2018	Video board failure	1	COMPU	RT-57455-121	07-Jan-2016		0.00
124	3244536	01-Jan-2018	Power supply short/burn	3	PCPAL	FRG-324458	09-Jan-2016		0.00

To place orders, to return equipment for repair, and so on, the system must contain VENDOR data. Table C.18 shows a simplified structure for the VENDOR entity.

As you examine the VENDOR entity's structure, keep the following points in mind:

TABLE C.18

THE VENDOR ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
VEND_ID	Vendor identification code		PK	
VEND_NAME	Vendor name	C		
VEND_ADDRESS	Vendor street address			
VEND_CITY	Vendor city			
VEND_STATE	Vendor state			
VEND_ZIP	Vendor zip code			
VEND_PHONE	Vendor phone	C		
VEND_CONTACT	Vendor contact person	C		
VEND_CON_PHONE	Vendor contact phone	C		
VEND_TECH_PHONE	Vendor tech support phone	C		
* The attribute type may be Composite (C), Derived (D), or Multivalued (M).				

- Because the system must be able to generate shipping labels for items returned to the vendor, it is necessary to decompose the vendor address into street address, city, state, and zip code.
- There was no end-user requirement for a vendor telephone number to be broken down by area code, nor was there a requirement for an alphabetically arranged list of contacts and technical support people. Therefore, the last four VENDOR attributes were left as composites.

A few sample VENDOR records are shown in Figure C.16.

FIGURE C.16 SAMPLE VENDOR DATA

VEND_ID	VEND_NAME	VEND_ADDRESS	VEND_CITY	VEND_STATE	VEND_ZIP	VEND_PHONE	VEND_CONTACT	VEND_CON_PHONE	VEND_TECH_PHONE
COMPU	ComputerLand	1012 Hard Drive,	San Francisco	CA	12345	8004567789	John E. Miesler	8004567785	8004567762
PCJUN	PC Junction	1234 Video Circle	Nashville	TN	23456	6153454567	Anna D. Williamson	6153454574	6153454573
PCPAL	PC Palace	4567 Ram Lane	New York	NY	34567	8002341234	Juan H. Cordova	8002341232	8002341235

Given the Lab's frequent hardware, software, and supply updates, the system's ORDER entity plays a crucial role. Its necessary attributes, required to satisfy budgeting, auditing, and various system end-user requirements, are reflected in the ORDER entity in Table C.19.

TABLE C.19

THE ORDER ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
ORD_ID	Order ID code		PK	
ORD_DATE	Order date			
VEND_ID	Vendor ID code		FK	VENDOR
ORD_VEND_REF	Reference number supplied by the vendor (optional)			
ORD_PO_NUM	Purchase order number			
ORD_TOT_COST	Total order cost, including shipping and handling			
ORD_STATUS	Order status: OPEN = open order REC = received CANCEL = canceled order PAID = paid order			
ORD_FUND_TYPE	Order funding source: BUS = College of Business budget			
USER_ID	Person who requested the order		FK	USER

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

Appendix B stated that each order contains one or many ordered items. Using the normalization rules, the order was split into two entities: ORDER and ORDER_ITEM. The ORDER entity is related in a one-to-many relationship with the ORDER_ITEM entity. (Review Figure C.9 to see the precise relationship between ORDER and ORDER_ITEM.) The ORDER entity (the "1" side) contains general data about the order; the ORDER_ITEM entity (the "M" side) contains the items in the order.

The ORDER entity's ORD_STATUS reflects reporting requirements. Clearly, it is possible for an order to have been received and not yet paid for, so a distinction must be made between REC and PAID. Although canceled orders have no impact on inventory movements, it is important to keep track of them. For example, before you make the payment on a bill, it would be wise to find out if the order has been canceled. The ORD_FUND_TYPE lets you know to which budget you should charge the payments. And because accountability is an ever-present factor, you must be able to track (through USER_ID) the person who originated the order.

Information requirements might also determine on which side of the 1:M relationship the data is stored. For example, it is quite possible for only part of an order to arrive at a given time. Let's say that the order consisted of 12 computers and 25 boxes of paper. All 25 boxes of paper but only 8 computers might have arrived. Because you must trace those portions of the order that are complete, the receipt date must be stored on the "M" side of the 1:M relationship. Using a similar approach, you must decide where to store the USER_ID attribute. For example, if information requirements demand that you get a precise listing of who ordered what specific item in any one order, the USER_ID must be stored on the "M" side. On the other hand, if you merely need to know who placed the

entire order, the USER_ID is stored on the "1" side. Given the latter scenario, you may then list the person who requested the specific item within an order as part of each order line description.

The placement of the VEND_ID depends on a simple business rule. In this case, it is quite reasonable to assume that each order is placed with a single vendor. (Just think how many checks you would have to write if each order line were tied to a different vendor within the same order.) Therefore, the VEND_ID is written on the "1" side of the relationship between the order and its order lines. The revised ORDER structure, shown in Table C.19, shows what decisions were made in that design.

To illustrate the data placement, some ORDER sample data are shown in Figure C.17.

FIGURE C.17 SAMPLE ORDER DATA

ORD_ID	ORD_DATE	VEND_ID	ORD_VEND_REF	ORD_PO_NUM	ORD_TOT_COST	ORD_STATUS	ORD_FUND_TYPE	USER_ID
121	06-Feb-2018	PCJUN	320021	21234	17335.13	OPEN	CIS	1-11-1117
122	06-Feb-2018	PCJUN	320213	21234	10698.20	PAID	CIS	1-11-1128
123	10-Feb-2018	COMPU	320322	21234	1751.95	PAID	CIS	1-11-1120
124	10-Feb-2018	PCPAL	000000	21234	2397.74	REC	BUS	1-11-1113
125	14-Feb-2018	COMPU	320345	21234	640.98	REC	CIS	1-11-1128

The "M" side of the relationship between orders and their components will be stored in the ORDER_ITEM table. Thus, each ORDER references one or more ORDER_ITEM records, but each ORDER_ITEM entry refers to a single ORDER record. The ORDER_ITEM's structure is shown in Table C.20.

TABLE C.20

THE ORDER_ITEM ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
ORD_ID	Order number		PK, FK	ORDER
OI_LINE	Order line number		PK	
TY_GROUP	Inventory group		FK	INV_TYPE
OI_DESCRIPTION	Item description			
OI_UNIT_COST	Unit cost			
OI_QTY_ORD	Order quantity			
OI_COST	Total line cost	D		
OI_QTY_RECVD	Quantity received			
OI_DATE_IN	Last date received			

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

Using two entities, ORDER and ORDER_ITEM, results in some additional advantages, as follows:

- The ORDER_ITEM entity will contain an OI_LINE attribute to represent the order in which the INV_TYPE (items) are actually entered. If ORD_ID and TY_GROUP are used instead, any change will produce a different listing each time an item is added to

an order. In that case, if an end user were to review a previous order, it might have an arrangement quite different from that of the original entries and would likely confuse the end user. By keeping the OI_LINE attribute in ORDER_ITEM, that problem is eliminated.

- The OI_DESCRIPTION has a special purpose. Although application software can be used to read the description found in INV_TYPE when the end user enters the TY_GROUP code, that description can still be modified to clarify the order details. (See the sample data in Figure C.18) Because this description does not serve the purpose of a foreign key, the redundancy does not create structural problems. Additionally, the descriptive material can prove to be helpful if questions arise later about the precise nature of the order.

The sample data for the ORDER_ITEM are shown in Figure C.18.

FIGURE C.18 SAMPLE ORDER_ITEM DATA

ORD_ID	OI_LINE	TY_GROUP	OI_DESCRIPTION	OI_UNIT_COST	OI_QTY_ORD	OI_COST	OI_QTY_RECVD	OI_DATE_IN
121	1	HWNCTR4M	Ethernet card, NIC (BAS112B addition)	184.23	1	184.23	1	10-Feb-2018
121	2	HWPCDTP3	DELL Desktop, 800GB HD, 160GB RAM	2395.00	5	11975.00	2	
121	3	HWPCLTP3	DELL Inspiron, 200GB HD, 4GB RAM, 14" display	2587.95	2	5175.90	2	10-Feb-2018
122	1	HWPRLSBL	HP Laserjet 4550, 64MB RAM	1999.99	5	9999.95	5	11-Feb-2018
122	2	SUPPSS11	Laser printer paper, 5,000 sheet box	19.95	35	698.25	35	11-Feb-2018

As you examine the ORDER and ORDER_ITEM data presented in Figure C.17 and Figure C.18, respectively, you can easily trace all orders and their components. For example, as you look at ORDER's ORD_ID = 121 and trace it through ORDER_ITEM's ORD_ID = 121, you can draw the following conclusions:

- The order consisted of three items: one network card, five Dell desktop computers, and two Dell laptop computers. (Note that the ORDER_ITEM's OI_LINE numbers range from 1 to 3 for ORD_ID = 121.)
- The order was written on February 10, 2018 to vendor PCJUN.
- The order is still open because the ORDER_ITEM's second order line (OI_LINE = 2) shows that only two of the five computers have been received. (Note that the ORDER_ITEM's OI_DATE_IN is null.)

Keeping track of the items in inventory is a challenge in the UCI's environment because there are two different types of transactions. Some items, such as paper, ink-jet cartridges, and other consumables, are withdrawn from inventory as they are used. For example, if a faculty member needs a box of paper for the classroom or the office, the stock of "boxes of paper" is simply decreased by one box. However, faculty and staff might also check out items *temporarily*, such as flat panel displays for use in the classroom or laptop computers to take to remote classes for demonstration purposes. In that case, the checked-out item remains in inventory, but its availability status and location change. You must be able to track each item's user and location. When the item is returned, another (check-in) transaction changes the availability status and location again.

To examine the inventory transactions, let's begin with some simple withdrawals. The following scenario covers four transactions, recorded as withdrawals 325, 326, 327, and 328. (You can evaluate the transaction components by reviewing the previously shown sample data. For example, you know that user 1-11-1111 is a CIS faculty member from examining the USER table. You can find item 4238131 in the ITEM table; you determine that this item is a laser printer cartridge by looking at the INV_TYPE table, and so on.)

- Transaction 325: CIS faculty member 1-11-1111 withdrew a laser printer cartridge (ITEM_ID = 4238131) from the computer lab (KOM245A-1) on February 4, 2018.
- Transaction 326: CIS staff member 1-11-1128 withdrew three laser printer cartridges from KOM245A-1, five boxes of single-sheet 8.5" × 11" paper (ITEM_ID = 3154567) from KOM245B-1, and two boxes of 3.5" disks (ITEM_ID = 4238132) from KOM245A-1 on February 4, 2018.
- Transaction 327: CIS staff member 1-11-1130 withdrew one box of 3.5" floppy disks from KOM106-1 on February 7, 2018.
- Transaction 328: CIS faculty member 1-11-1109 withdrew one box of single-sheet 8.5" × 11" paper from KOM106-1 and one black ink-jet cartridge (ITEM_ID = 4238130) from KOM1061-1 on February 8, 2018.

Before those transactions can be tracked, there must be a set of database tables to support them. At this point, you cannot track the transactions because the database reflects withdrawals as the M:N relationship between USER and ITEM shown in Figure C.19, Panel A. For example, a qualified user can withdraw many boxes of paper, and a box of paper in inventory can be withdrawn by any number of qualified users.

Because the M:N relationship cannot be properly implemented in a relational database design, your first thought might be to transform the "withdraws" relationship into a composite entity named WITHDRAW, shown in Figure C.19, Panel B. That entity's structure is illustrated in Table C.21.

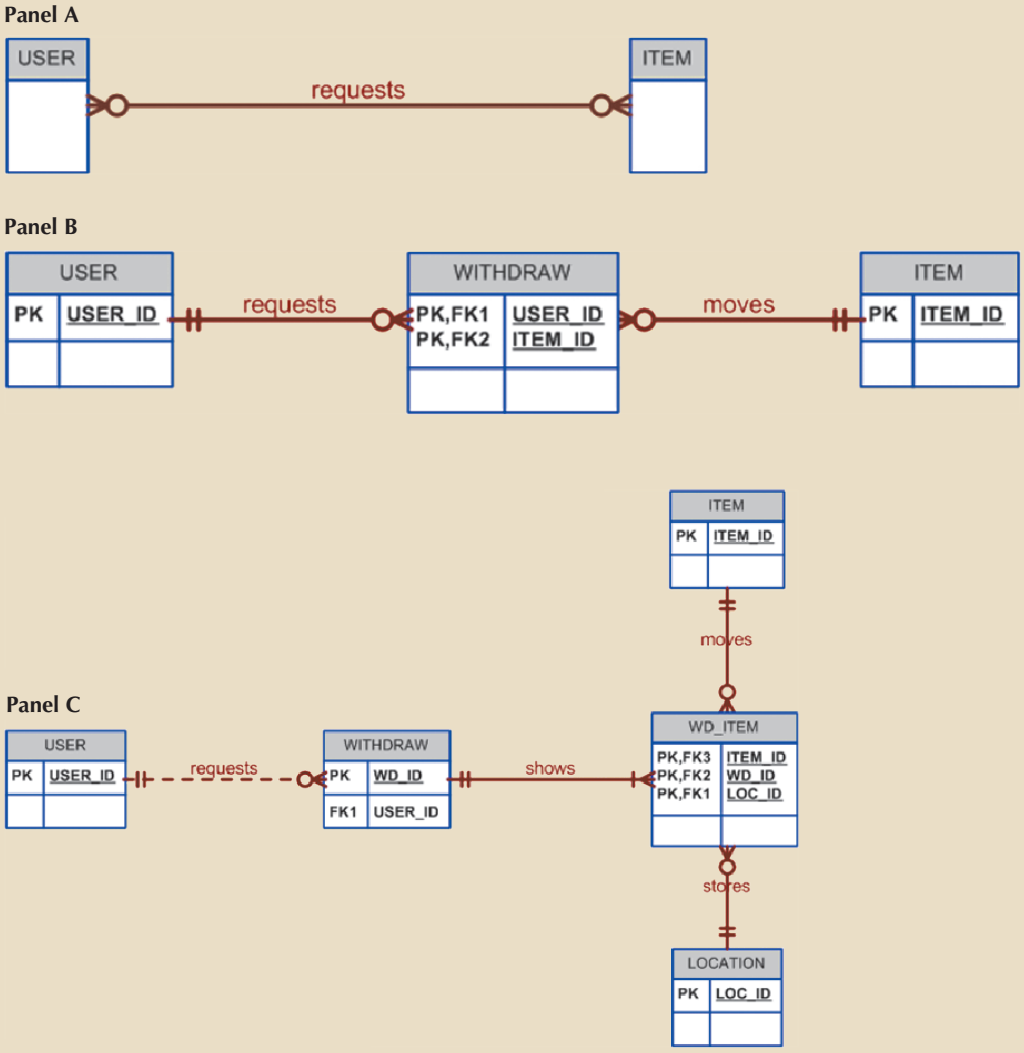
TABLE C.21

THE WITHDRAW ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
WD_DATE	Withdrawal date		PK	
USER_ID	User ID (faculty or staff)		PK, FK	USER
ITEM_ID	Item ID for withdrawn item	M	PK, FK	ITEM
LOC_ID	Location ID	M	PK, FK	LOCATION
WD_QTY	Quantity withdrawn	M		

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

FIGURE C.19 THE WITHDRAW REVISION PROCESS



The WITHDRAW entity in Table C.21 seems to have all of the required attributes. In addition, Figure C.19, Panel B indicates that the addition of the WITHDRAW entity certainly has transformed the M:N relationship between USER and ITEM into two sets of 1:M relationships. Yet in spite of the design's improvement, WITHDRAW will not perform its intended functions well. Although its components help tie USER, ITEM, and LOCATION together, it contains three multivalued attributes. To eliminate those multivalued attributes, the WITHDRAW entity in Table C.21 can be decomposed into the two entities shown in Figure C.19, Panel C.

Using Figure C.19, Panel C as a guide, you can revise the WITHDRAW entity to eliminate the multivalued attributes and place them in WD_ITEM. Those two entities are shown in Tables C.22 and C.23 and their sample data are shown in Figures C.20 and C.21, respectively. (The data trace the withdrawal scenario presented at the beginning of this discussion.)

TABLE C.22

THE REVISED WITHDRAW ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
WD_ID	Withdrawal number		PK	
WD_DATE	Withdrawal date			
USER_ID	User ID (faculty or staff)		FK	USER
* The attribute type may be Composite (C), Derived (D), or Multivalued (M).				

FIGURE C.20 SAMPLE WITHDRAW DATA

WD_ID	WD_DATE	USER_ID
325	04-Feb-2018	1-11-1111
326	04-Feb-2018	1-11-1128
327	07-Feb-2018	1-11-1130
328	08-Feb-2018	1-11-1109

TABLE C.23

THE WD_ITEM (WEAK) ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
WD_ID	Withdrawal ID		PK, FK	WITHDRAW
ITEM_ID	Item ID for withdrawn item		PK, FK	ITEM
LOC_ID	Location ID		PK, FK	LOCATION
WD_QTY	Quantity withdrawn			
* The attribute type may be Composite (C), Derived (D), or Multivalued (M).				

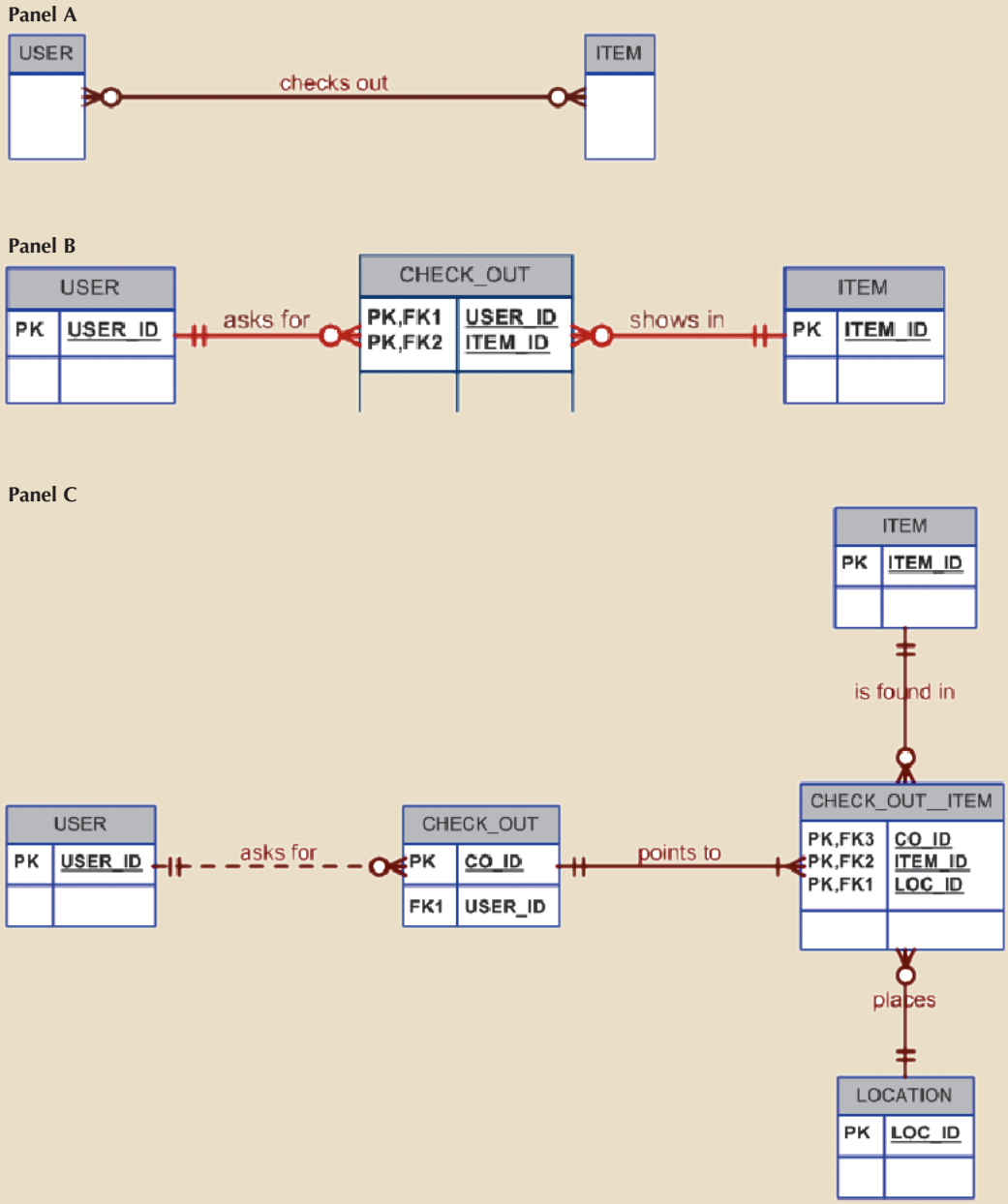
FIGURE C.21 SAMPLE WD_ITEM DATA

WD_ID	ITEM_ID	LOC_ID	WI_QTY
325	4238131	KOM245A-1	1
326	4238131	KOM245A-1	3
326	3154567	KOM245B-1	5
326	4238132	KOM245A-1	2
327	4238132	KOM106-1	1
328	3154567	KOM106-1	1
328	4238130	KOM106-1	1

WITHDRAW and WD_ITEM are capable of supporting the required withdrawal transactions, so this revision could be incorporated into the final design. However, another revision will be made later to standardize all inventory transactions.

Because the check-out transactions are subject to the same basic process as the withdrawal transactions, Figure C.22 illustrates that their design revisions mirror those of the withdrawal revision process. The difference between WITHDRAW and CHECK_OUT is that the latter yields two expected transactions for each item: one when the item is checked out and one when the item is returned.

FIGURE C.22 THE CHECK_OUT REVISION PROCESS



Because the check-out revision process basically mirrors that of the withdrawal revision, the discussion will simply note Figure C.22, Panels A and B, without providing any further revision details.

Using Figure C.22, Panel C as a design guide, Table C.24 defines the CHECK_OUT structure.

TABLE C.24

THE CHECK_OUT ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
CO_ID	Check-out ID		PK	
CO_DATE	Check-out date			
USER_ID	User ID		FK	USER
* The attribute type may be Composite (C), Derived (D), or Multivalued (M).				

To see the check-out process in action, let's trace the following transactions. (You will also see the difference between withdrawal and check-out when you look at the CO_ITEM's structure and sample data.)

- CO_ID = 101: Accounting faculty member 1-11-1117 checked out a laptop computer (4228753) from KOM245A-1 on February 2, 2018 and returned it on February 3, 2018.
- CO_ID = 102: CIS faculty member 1-11-1128 checked out a laptop computer (4358255) from KOM245B-1 and a projector panel (4358258) from KOM245B-1 on February 3, 2018. Only the laptop computer was returned on February 4, 2018.
- CO_ID = 103: CIS faculty member 1-11-1128 checked out the laptop computer (4228753) that was returned by the Accounting faculty member in transaction 101, from KOM245A-1 on February 3, 2018. The CIS faculty member has not yet returned the laptop.
- CO_ID = 104: CIS staff member 1-11-1112 checked out a laptop computer (4112151) from KOM245A-2 on February 4, 2018 and returned it on February 5, 2018.

Note how those transactions are reflected in Figures C.23 and C.24.

FIGURE C.23 SAMPLE CHECK_OUT DATA

CO_ID	CO_DATE	USER_ID
101	02-Feb-2018	1-11-1117
102	03-Feb-2018	1-11-1128
103	03-Feb-2018	1-11-1128
104	04-Feb-2018	1-11-1112

Each of the CHECK_OUT records will point to the transaction details—that is, the "Many" side, represented by the CHECK_OUT_ITEM entity shown in Table C.25.

TABLE C.25

THE CHECK_OUT_ITEM (WEAK) ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
CO_ID	Check-out ID		PK	
ITEM_ID	Item ID		PK, FK	ITEM
LOC_ID	Location ID		PK, FK	LOCATION
COI_QTY	Check-out item quantity			
COI_DATE_IN	Date the item was returned			

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

The scenario is completed as shown in the sample data in Figure C.24.

FIGURE C.24 SAMPLE CHECK_OUT_ITEM DATA

CO_ID	ITEM_ID	LOC_ID	COI_QTY	COI_DATE_IN
101	4228753	KOM245A-1	1	03-Feb-2018
102	4358255	KOM245B-1	1	04-Feb-2018
102	4358258	KOM245B-1	1	
103	4228753	KOM245A-1	1	
104	4112151	KOM245A-2	1	05-Feb-2018

As you examine Figure C.24, note that it accurately portrays the transactions described earlier. Because item 4358258 in transaction 102 and item 4228753 in transaction 103 have not yet been returned, their COI_DATE_IN values are null. As was true in the case of the withdrawal process, you can now support the check-out and check-in transactions. However, you will discover in the next section that the inventory transaction process can be streamlined further.

C-3 Verifying the ER Model

Let's look at what you have accomplished. At this point, you have identified:

- *Entity sets, attributes, and domains.*
- *Composite attributes*. Such attributes may be (and usually are) decomposed into several independent attributes.
- *Multivalued attributes*. You implemented them in a new entity set in a l:M relationship with the original entity set.
- *Primary keys*. You ensured primary key integrity.
- *Foreign keys*. You ensured referential integrity through the foreign keys.
- *Derived attributes*. You ensured the ability to compute their values.
- *Composite entities*. You implemented them with 1:M relations.

Although you have made considerable progress, much remains to be done before the model can be implemented.

To complete the UCL conceptual database design, you must *verify* the model. Verification represents the link between the database modeling and design activities, database implementation, and database application design. Therefore, the verification process is used to establish that:

- The design properly reflects the end-user or application views of the database.
- All database transactions—inserts, updates, deletes—are defined and modeled to ensure that the implementation of the design will support all transaction-processing requirements.
- The database design is capable of meeting all output requirements, such as query screens, forms, and report formats. (Remember that information requirements may drive part of the design process.)
- All required input screens and data entry forms are supported.
- The design is sufficiently flexible to support expected enhancements and modifications.

In spite of the fact that you were forced to revise the ER diagram initially depicted in Appendix B's Figure B.19, it is still possible that:

- Some of the relationships are not clearly identified and may even have been misidentified.
- The model contains design redundancies. (Consider the similarity between the WITHDRAW and CHECK_OUT entities.)
- The model can be enhanced to improve semantic precision and to better represent the operations in the real world.
- The model must be modified to better meet user requirements (such as processing performance or security).

The following few paragraphs will demonstrate the verification process for some of the application views in the Inventory Management module. (This verification process should be repeated for all of the system's modules.)

- *Identifying the central entity*. Although the satisfaction of the UCL's end users is vital, inventory management has the top priority from an administrative point of view. The reason for that priority rating is simple: state auditors target the Lab's considerable and costly inventory to ensure accountability to the state's taxpayers. Failure to track items properly may have serious consequences; therefore, ITEM becomes the UCL's central entity.
- *Identifying each module and its components*. Table C.2 identifies the modules and their components. It is important to "connect" the components by using shared entities. For example, although USER is classified as belonging to the Lab Management module and ITEM is classified as belonging to the Inventory Management module, the USER and ITEM entities interface with both. For example, the USER is written into the LOG in the Lab Management module. USER also features prominently in the Inventory Management module's withdrawal of supplies and in the check-out/check-in processes.
- *Identifying each module transaction requirement*. You will focus your attention on one of the INVENTORY module's reporting requirements. The authors suggest that you identify other transaction requirements. Then you can validate those requirements against the UCL database for all system modules.

An examination of the Inventory Management module's reporting requirements uncovers the following problems:

- The Inventory module generates three reports, one being an inventory movement report. But the inventory movements are spread across several different entities (CHECK_OUT and WITHDRAW and ORDER). That spread makes it difficult to generate the output and reduces system performance.
- An item's *quantity on hand* is updated with an inventory movement that can represent a purchase, withdrawal, check-out, check-in, or inventory adjustment. Yet only the *withdrawals* and *check-outs* are represented in the model.

The solution to those problems is described by the database designer:

What the Inventory Management module needs is a common entry point for all movements. In other words, the system must track all inputs to and withdrawals from inventory. To accomplish that task, we must create a new entity to record all inventory movements; that is, we need an inventory transaction entity. We will name that entity INV_TRANS.

The creation of a common entry point serves two purposes:

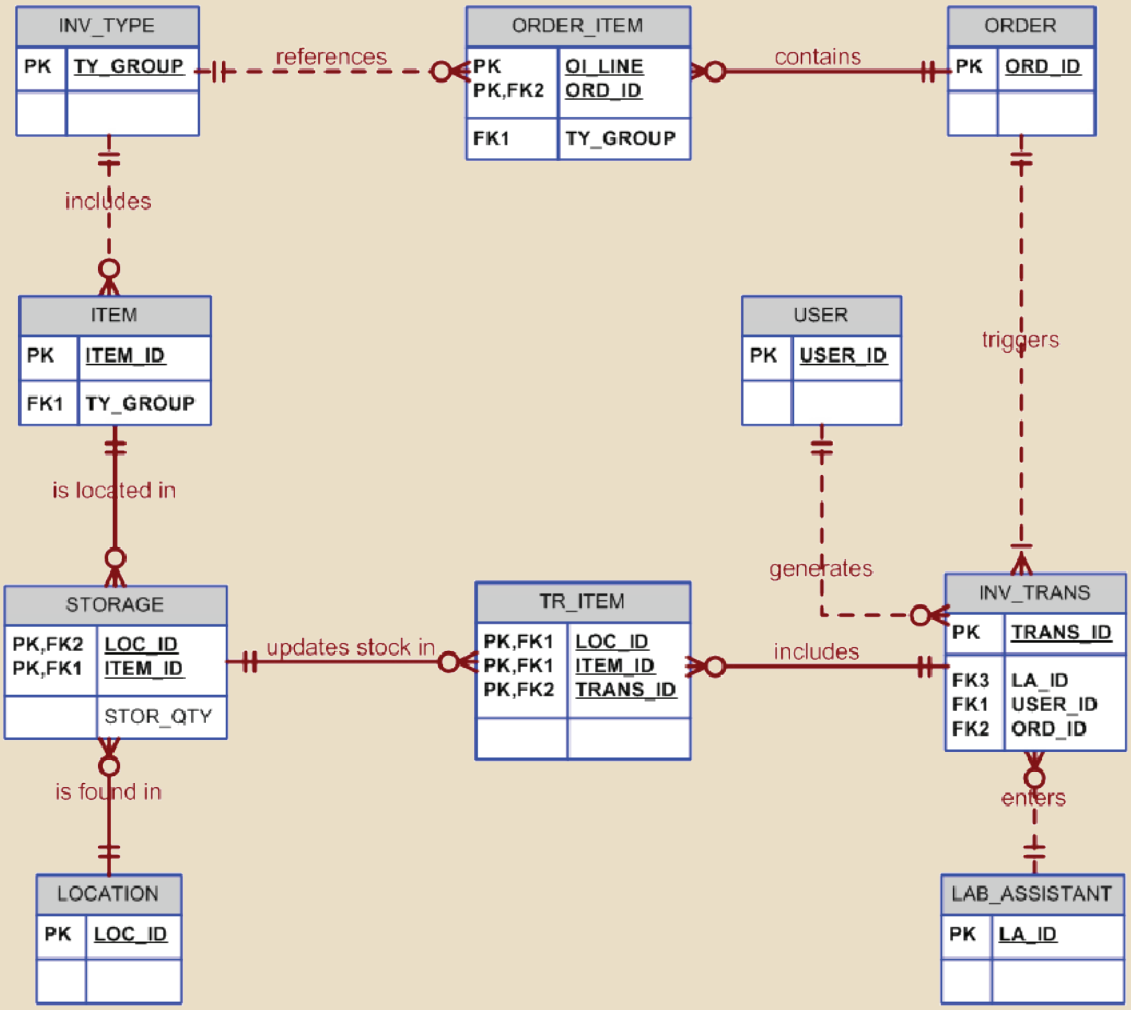
1. It standardizes the Inventory module's interface with other (external) modules. Any inventory movement (whether it adds or withdraws) will generate an inventory transaction entry.
2. It facilitates control and generation of required outputs, such as the inventory movement report.

Figure C.25 illustrates a solution to the problems just described.

The INV_TRANS entity in Figure C.25 is a simple inventory transaction log, and it will contain any inventory I/O movement. Each INV_TRANS entry represents one of two types of movement: input (+) or output (-). Each INV_TRANS entry must contain a line in TR_ITEM for each item that is added, withdrawn, checked in, or checked out.

The INV_TRANS entity's existence also enables you to build additional I/O movements efficiently. For example, when an ordered item is received, an inventory transaction entry (+) is generated. That INV_TRANS entry will update the quantity received (OI_QTY_RECVD) attribute of the ORDER_ITEM entity in the Inventory Management module. The Inventory Management module will generate an inventory transaction entry (-) to register the items checked out by a user, and it will generate another inventory transaction entry (+) to register the items checked in. The withdrawal of items (supplies) will also generate an inventory transaction entry (-) to register the items that are being withdrawn. Those relationships are depicted in Figure C.25.

FIGURE C.25 THE INVENTORY TRANSACTION PROCESS



The new INV_TRANS entity's attributes are shown in Table C.26.

TABLE C.26

THE INV_TRANS ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
TRANS_ID	Inventory transaction ID (This code is generated by the system.)		PK	
TRANS_TYPE	Inventory transaction type: I = input (add to inventory) O = output (subtract from inventory)			

TABLE C.26

THE INV_TRANS ENTITY (CONTINUED)

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
TRANS_PURPOSE	Reason for inventory transaction: PO = purchase order (add to the inventory) CC = check-out (subtract from inventory) WD = withdrawal (subtract from inventory) AD = adjustment (add to or subtract from inventory, depending on the type of adjustment)			
TRANS_DATE	Inventory transaction date			
LA_ID	Lab assistant who recorded the transaction		FK	LAB_ASSISTANT
USER_ID	Person who created the transaction		FK	USER
ORDER_ID	Order ID		FK	ORDER
TRANS_COMMENT	Comments			

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

To see how INV_TRANS in Table C.26 works, refer to the ER segment in Figure C.25 and note that its detail lines are kept in the (weak) TR_ITEM. Figure C.25 also illustrates that all of the inventory movements can now be traced. For example, any item must be stored somewhere, so its location can be accessed through STORAGE. Because INV_TRANS is related to both LAB_ASSISTANT and USER, you know who recorded the transaction and who generated it. Figure C.26 contains sample data that will allow you to trace the:

- Withdrawal transactions first examined in Figures C.20 and C.21.
- Check-in and check-out transactions first examined in Figures C.23 and C.24.
- Purchase of 2 HP laser printers and 35 boxes of paper.

FIGURE C.26 SAMPLE INV_TRANS DATA

TRANS_ID	TRANS_TYPE	TRANS_PURPOSE	TRANS_DATE	LA_ID	USER_ID	ORDER_ID	TRANS_COMMENT
325	O	WD	04-Feb-2018	1-11-2003	1-11-1117		Laser printer cartridge
326	O	WD	04-Feb-2018	1-11-2008	1-11-1128		Laser printer cartridge
327	O	WD	07-Feb-2018	1-11-2003	1-11-1128		Box of 20 3.5" floppy disks, HD/DS
328	O	WD	08-Feb-2018	1-11-2008	1-11-1111		Two reams of 8.5x11" paper & ink-jet cartridge
401	O	CC	02-Feb-2018	1-11-2008	1-11-1120		Laptop check-out
402	I	CC	03-Feb-2018	1-11-2009	1-11-1120		Laptop returned
403	O	CC	03-Feb-2018	1-11-2003	1-11-1111		Laptop & projector check-out
404	O	CC	03-Feb-2018	1-11-2009	1-11-1112		Laptop check-out
405	I	CC	04-Feb-2018	1-11-2008	1-11-1111		Laptop returned
406	O	CC	04-Feb-2018	1-11-2021	1-11-1133		Laptop check-out
407	I	CC	05-Feb-2018	1-11-2003	1-11-1133		Laptop returned
408	I	PO	11-Feb-2018	1-11-2008	1-11-1128	122	Two HP printers, 35 boxes of paper order

For example, the first INV_TRANS row reveals that on February 4, 2018, a laser printer cartridge was withdrawn from inventory by user 1-11-1117. The transaction was recorded by LA 1-11-2003, and the transaction decreased (TRANS_TYPE = O) the stock in inventory.

The transaction details in Figure C.26 are stored in TR_ITEM, so before you can examine those details, you must examine the TR_ITEM structure in Table C.27.

TABLE C.27

THE TR_ITEM (WEAK) ENTITY

ATTRIBUTE NAME	CONTENTS	ATTRIBUTE TYPE (*)	PRIMARY KEY (PK) AND/OR FOREIGN KEY (FK)	REFERENCES
TRANS_ID	Inventory transaction ID (This code is generated by the system in the INV_TRANS entity.)		PK, FK	INV_TRANS
ITEM_ID	Item ID		PK, FK	ITEM
LOC_ID	Location ID		PK, FK	LOCATION
TRANS_QTY	Quantity withdrawn			

* The attribute type may be Composite (C), Derived (D), or Multivalued (M).

By examining the sample data shown in Figure C.27, you can trace the transaction details in Figure C.26.

FIGURE C.27 SAMPLE TR_ITEM DATA

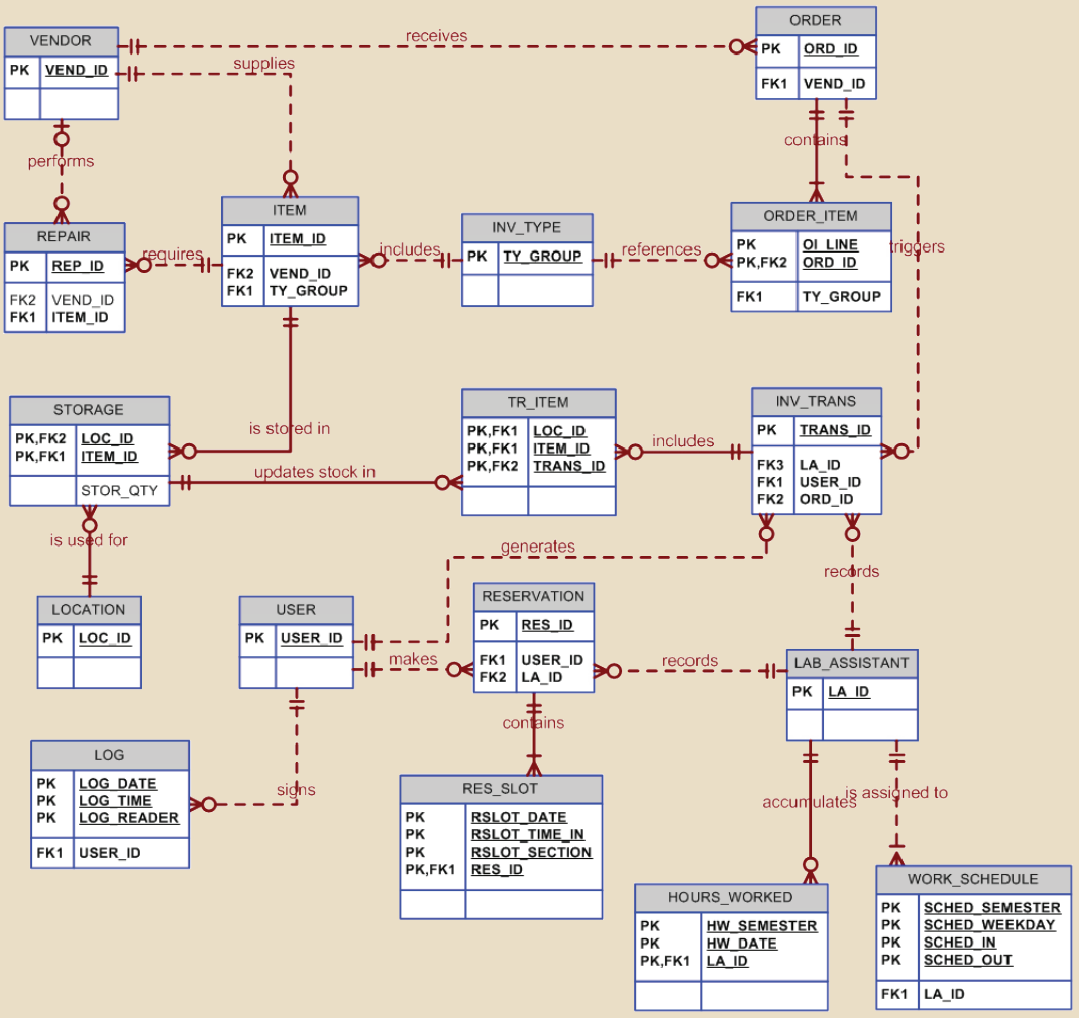
TRANS_ID	ITEM_ID	LOC_ID	TRANS_QTY
325	4238131	KOM245A-1	1
326	3154567	KOM245B-1	5
326	4238131	KOM245A-1	3
326	4238132	KOM245A-1	2
327	4238132	KOM106-1	1
328	3154567	KOM106-1	1
328	4238130	KOM106-1	1
401	4228753	KOM245A-1	1
402	4228753	KOM245A-1	1
403	4358255	KOM245B-1	1
403	4358258	KOM245B-1	1
404	4228753	KOM245A-1	1
405	4358255	KOM245B-1	1
406	4112151	KOM245A-2	1
407	4112151	KOM245A-2	1
408	3154567	KOM245A-2	35
408	4567920	KOM245B-2	1
408	4567921	KOM245B-2	1

For example, note that the first INV_TRANS row's TRANS_ID = 325 entry (see Figure C.26) now points to the TR_ITEM's TRANS_ID = 325 entry shown in Figure C.27, thus allowing you to conclude that that transaction involved the withdrawal of a single unit of item 4238131, a laser printer cartridge. (You can conclude that item 4238131 is a laser printer cartridge by examining the INV_TYPE and ITEM data in Figures C.11 and C.12, respectively, and noting that ITEM_ID = 4238131 corresponds to TY_GROUP = SUCALPXX.) Transaction 326 involved three items, so the TR_ITEM table contains three detail lines for that transaction.

Examine how check-outs and check-ins are handled. In Figure C.26, INV_TRANS transaction 401 records TRANS_PURPOSE = CC and TRANS_TYPE = O, indicating that a check-out was made. That transaction recorded the following: check-out of a laptop, ITEM_ID = 4228753 (see Figure C.21), on February 2, 2018 by an Accounting faculty member, USER_ID = 1-11-1120. The laptop was returned on February 3, 2018, and that transaction was recorded as TRANS_ID = 402, whose TRANS_PURPOSE = CC and TRANS_TYPE = I, indicating that this particular laptop was returned to the available inventory. Incidentally, because the department owns several laptops, faculty members need not wait for a laptop to be returned before checking one out, *as long as there are laptops in inventory*. However, if no additional laptops are available, the system can trace who has them and when they were checked out. If the CLD wants to place restrictions on the length of time an item can be checked out, this design makes it easy to notify users to return the items in question.

The final entity relationship diagram reflects the changes that have been made. Although the original ER diagram is easier to understand from the user's point of view, the revised ER diagram has more meaning from the *procedural* point of view. For example, the changes made are totally *transparent* (invisible) to the user because the user never sees the INV_TRANS entity. The final ER diagram is shown in Figure C.28.

FIGURE C.28 THE REVISED UNIVERSITY COMPUTER LAB ERD



C-4 Logical Design

When the conceptual design phase is completed, the ERD reflects—at the conceptual level—the business rules that, in turn, define the entities, relationships, optionalities, connectivities, cardinalities, and constraints. (Remember that some of the design elements cannot be modeled and are, therefore, enforced at the application level. For example, the constraint, "a checked-out item must be returned within five days" cannot be reflected in the ERD.) In addition, the conceptual model includes the definition of the attributes that describe each of the entities and that are required to meet information requirements.

Keep in mind that the conceptual model's entities must be normalized before they can be properly implemented. The normalization process may yield additional entities and relationships, thus requiring the modification of the initial ERD. Because the focus was on the verification of the conceptual design to produce an *implementable* design, the model verified in this appendix was certain to meet the requisite normalization requirements. In short, design and normalization processes were used concurrently. In fact, such concurrent use of design and normalization reflects real-world practice. The logical design process is used to translate the conceptual design into the internal model for the selected DBMS. To the extent that normalization helps establish the appropriate attributes, their characteristics, and their domains, normalization moves you to the logical design phase. Nevertheless, because the conceptual modeling process does not preclude the definition of attributes, you can reasonably argue that normalization occasionally straddles the line between conceptual and logical modeling.

It bears repeating that the logical design translates the conceptual model in order to match the format expected of the DBMS that is used to implement the system. Because you will be using a relational database model, the logical design phase sets the stage for creating the relational table structures, indexes, and views.

C-4a Tables

The following few examples illustrate the design of the logical model, using SQL. (Make sure that the tables conform to the ER model's structure and that they obey the foreign key rules if your DBMS software allows you to specify foreign keys.)

Using SQL, you can create the table structures within the database you have designated. For example, the STORAGE table would be created with:

```
CREATE TABLE STORAGE (
      LOC_ID          CHAR(12)  NOT NULL,
      ITEM_ID         CHAR(10)  NOT NULL,
      STOR_QTY        NUMBER,
      PRIMARY KEY (LOC_ID, ITEM_ID),
      FOREIGN KEY (LOC_ID) REFERENCES LOCATION
            ON DELETE RESTRICT
            ON UPDATE RESTRICT,
      FOREIGN KEY (ITEM_ID) REFERENCES ITEM
            ON DELETE CASCADE
            ON UPDATE CASCADE);
```

Most DBMSs now use interfaces that allow you to type the attribute names into a template and to select the attribute characteristics you want from pick lists. You can even insert comments that will be reproduced on the screen to prompt the user for input. For example, the preceding STORAGE table structure might be created in a Microsoft Access template, as shown in Figure C.29.

When all of the tables required by the design have been created, the relationships specified in the design are established. A good CASE tool will let you accomplish those

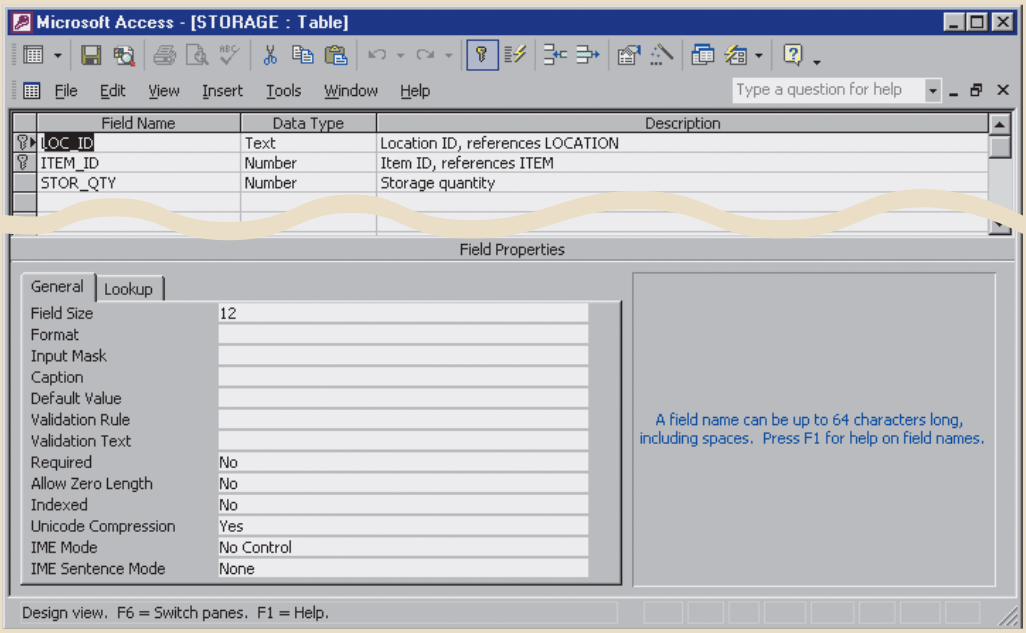
tasks directly from the design. For example, the design shown in Figure C.28 can be written into the specified database structure by the CASE tool. The advantages of letting the CASE tool write all of the table structures and relationships are that:

- The database will match the design precisely.
- All of the relationships have already been tested by the CASE tool to ensure that they are logically correct and that they are implementable as designed.
- All of the FK attribute definitions and characteristics match those of the referenced PKs.

Regardless of how you translate the design shown in Figure C.28 into the matching database structure, the database's relational schema must match the design. For example, Figure C.30 shows the relational schema in MS Access format.

As you examine the relational diagram in Figure C.30, note that all of its tables and relationships match the design specifications in Figure C.29 Note also that the relational diagram shows the addition of attributes that serve the end-user information and data management requirements.

FIGURE C.29 THE STORAGE TABLE STRUCTURE DEFINED IN MICROSOFT ACCESS



C-4b Indexes and Views

In the logical design phase, the designer can specify appropriate indexes to enhance operational speed. Indexes also enable the production of logically ordered output sequences. For example, if you want to generate the LA schedule shown in Table C.6, you need data from two tables, LAB_ASSISTANT and WORK_SCHEDULE. Because the report output is ordered by semester, LA, weekday, and time, indexes must be available for the primary key fields in each table. Using SQL, you would type:

```
CREATE UNIQUE INDEX LA_DEX
        ON LAB_ASSISTANT (LA_ID);
```

and

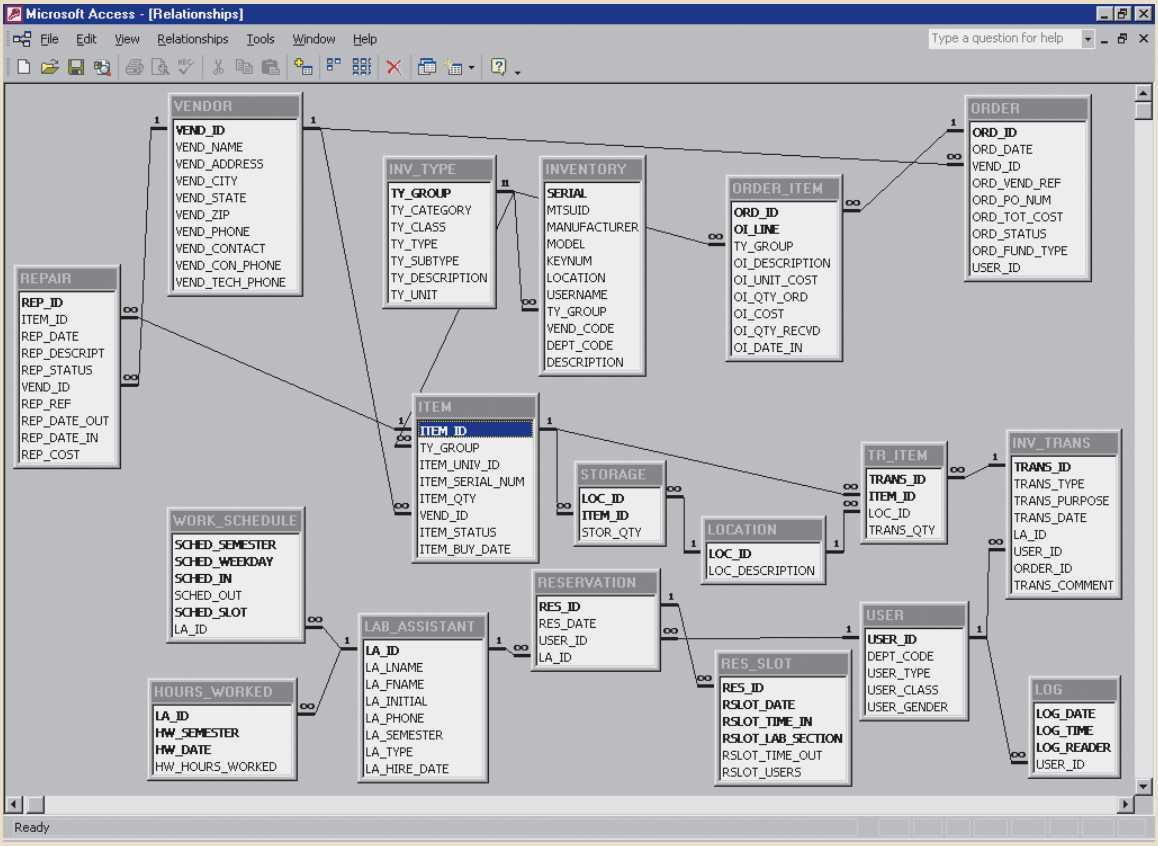
```
CREATE UNIQUE INDEX WS_DEX
        ON WORK_SCHEDULE (SCHED_SEMESTER, LA_ID, SCHED_WEEK-
        DAY, SCHED _IN);
```

Most modern DBMSs automatically index on the primary key components.

Views (see Chapter 8, Advanced SQL) are often used for security purposes. However, views are also used to streamline the system's processing requirements. For example, output limits may be defined efficiently by specifying appropriate views. To define the view necessary for the LA schedule report for the Spring semester of 2012, the CREATE VIEW command is used:

```
CREATE VIEW LA_SCHED AS
        SELECT  LA_ID,  LA_NAME,  SCHED_WEEKDAY,  SCHED  _IN,
        SCHED _OUT      FROM  WORK_SCHEDULE
        WHERE   SCHED_SEMESTER='SPRING12';
```


FIGURE C.30 THE UCL RELATIONAL DIAGRAM



The designer creates the views necessary for each database output operation.

Note

Unlike some other databases, the relational database model does not require the use of views in order to access the database. However, using views yields security benefits and greater output efficiency.

C-5 Physical Design

Physical design requires the definition of specific storage or access methods that will be used by the database. Within the DBMS's confines, the physical design must include an estimate of the space required to store the database. The required space estimate is translated into the space to be reserved within the available storage devices.

Physical storage characteristics are a function of the DBMS and the operating systems being used. Most of the information necessary to define the physical storage characteristics is found in the technical manuals of the software you are using. For example, if you use the newest SQL Server database, an estimate of the physical storage required for database creation (empty database) is provided by a table such as the one shown in Table C.28.

TABLE C.28

FIXED SPACE CLAIMED PER DATABASE

	DISK SPACE IN KB
Fixed space per table created within the database	535
17 tables 2-4 KB per table	68
Total fixed overhead used by database	603

Next, you need to estimate the data storage requirements for each table. Table C.29 shows the calculation for the USERS table only.

TABLE C.29

PHYSICAL STORAGE REQUIREMENTS: THE USER TABLE

ATTRIBUTE NAME	DATA TYPE	STORAGE REQUIREMENT (BYTES)
USER_ID	CHAR(11)	11
DEPT_CODE	CHAR(7)	7
USER_TYPE	CHAR(5)	5
USER_CLASS	CHAR(5)	5
USER_GENDER	CHAR(1)	1
Row length 29 Number of rows 15,950 Total space required 462,550		

If the DBMS does not automate the process of determining storage locations and data access paths, physical design requires well-developed technical skills and a precise knowledge of the physical-level details of the database, operating system, and hardware used by the database. Fortunately, the more recent versions of relational DBMS software hide most of the complexities inherent in the physical design phase.

cluster tables
A data storage structure that physically stores related rows from different tables together to improve the speed at which related data can be accessed.

You might store the database within a single volume of permanent storage space, or you can use several volumes, distributing the data in order to decrease data-retrieval time. Some DBMSs also allow you to create cluster tables and indexes. **Cluster tables** store rows of different tables together, in consecutive disk locations. That arrangement speeds up data access; it is mainly used in master/detail relationships such as ORDER and ORDER_ITEM or INV_TRANS and TR_ITEM.

The database designer must make decisions that affect data access time by fine-tuning the buffer pool size, the page frame size, and so on. Those decisions are based on the selected hardware platform and the DBMS. Consult the hardware and DBMS software manuals for the specific storage and access methodologies.

In the UCLMS, several indexes can be created to improve access time:

- Indexes created for all primary keys will increase access speed when you use foreign key references in tables. This is done automatically by the DBMS.
- Indexes can also be created for all alternative search keys. For example, if you want to search the LAB_ASSISTANT table by username, you should create an index for the LA_LNAME attribute; for example:
  CREATE INDEX LA001 ON LAB_ASSISTANT (LA_LNAME);
- Indexes can be created for all secondary access keys used in reports or queries. For example, an inventory movement report is likely to be ordered by inventory type and item ID. Therefore, an index is created for the ITEM table:
  CREATE INDEX INV002 ON ITEM (TY_GROUP, ITEM_ID);
- Indexes can be created for all columns used in the WHERE, ORDER BY, and GROUP BY clauses of a SELECT statement.

C-6 Implementation

One of the significant advantages of using a database is that it enables users to share data. When data are held in common, rather than being "owned" by various organizational divisions, data management becomes a more specialized task. Thus, the database environment favors the creation of a new organizational structure designed to manage the database resources. Database management functions are controlled by the database administrator (DBA). The DBA must define the standards and procedures required to interact with the database. (See Chapter 16, Database Administration and Security.)

Once the database designer has completed the conceptual, logical, and physical design phases, the DBA adopts an appropriate implementation plan. The plan includes formal definitions of the processes and standards to be followed, the chronology of the required activities (creation, loading, testing, and evaluation), the development of adequate documentation standards, the specific documentation needed, and the precise identification of responsibilities for continued development and maintenance.

Keep in mind that the technical details of the implementation are of little concern to the end user. Once the design has been implemented, the end users must be able to use the database and its contents—according to their work requirements and clearances—with relative ease and great utility. Therefore, the hard work of developing a user-friendly interface remains. Figures C.31 through C.34 show a sample main menu, a selection from that menu, some sample data entries, and the completed record based on a Microsoft Access database. Note that the end user interface shown in those figures uses several techniques to ensure appropriate data entries.

- *Drop-down lists* to limit the input selections. As you examine Figure C.32, note that the customer data have already been entered. In this case, the customer number 10011 was selected from a drop-down list of existing customers. The drop-down list is triggered by clicking on the downward-facing arrow button at the right margin of the customer input field. (Naturally, if the customer is new, a customer record must first be created.) Note that the customer financial data show up after the selection from the customer list, enabling the end user to authorize charges or to require full payment of the charter charges. Similarly, clicking on the downward-facing arrow button located on the right of the aircraft input field produces a drop-down list that shows all of the available aircraft and the relevant data for each aircraft. Those features enable the end user to answer customer questions without having to leave the input screen.
- *Automatic data entry* completions based on the input selections. For example, once an aircraft has been selected from the drop-down list, all appropriate field values for the

selected aircraft—such as the charge per mile and charge per waiting hour—are automatically written into the entry blanks. That feature eliminates end-user input errors and improves efficiency. (The end user does not need to type the values.)

- *System-generated computations* to avoid end-user computational errors. Once the distance flown and the waiting hours have been entered, all charges are calculated by the system, thus avoiding end-user calculation errors. (For example, Hours flown = Hobbs return – Hobbs out. A Hobbs meter is an instrument that records time.) Similarly, once the amount paid is entered, the balance is automatically calculated and entered into the "Balance owed" input field.

Many of the data entries in Figure C.33 are computed automatically. For example, the flight hours are computed after you have entered the Hobbs' time in and you leave that field. The charges and the unpaid balance, if any, are also computed automatically. When the data entry is complete and you press the Update button, the affected tables

FIGURE C.31 THE RC-CHARTER2 COMPANY MAIN MENU

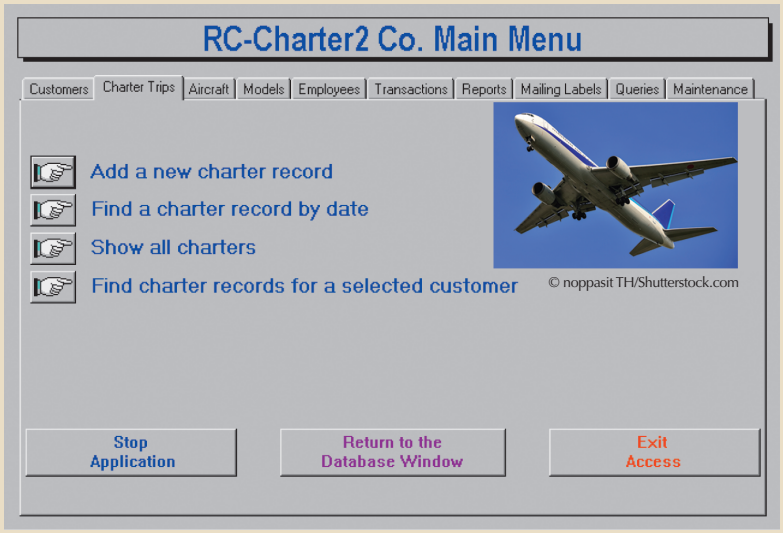
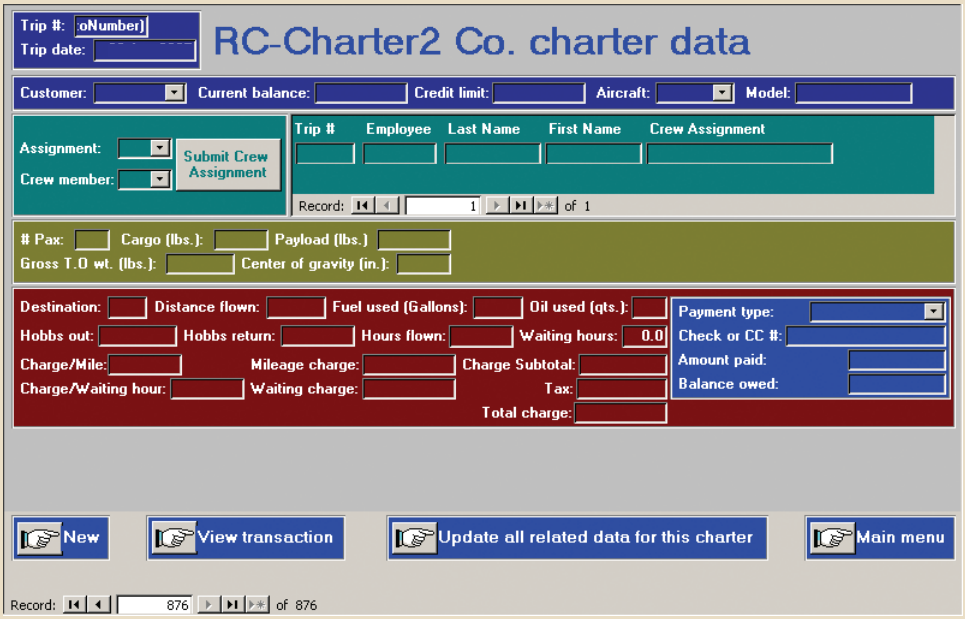
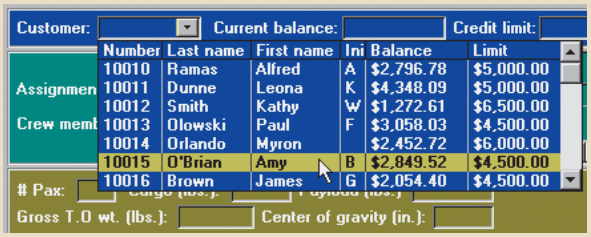


FIGURE C.32 THE RC-CHARTER2 COMPANY NEW CHARTER RECORD SELECTION



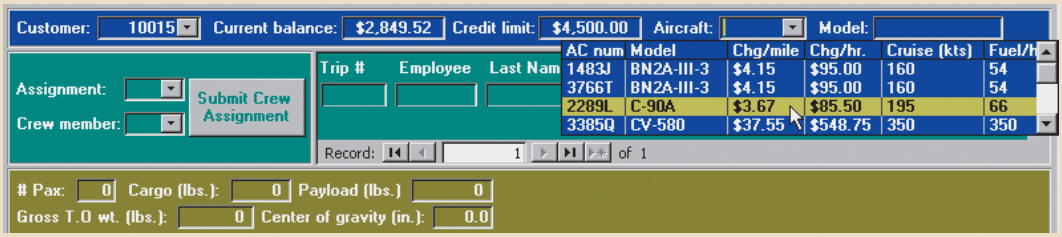
are updated, too. Note, for example, that the unpaid balance shown in Figure C.33 has been added to show the new customer balance. (Compare the customer balance value in Figure C.33 with its counterpart in Figure C.34.)

FIGURE C.33 CHARTER RECORD SAMPLE DATA ENTRIES

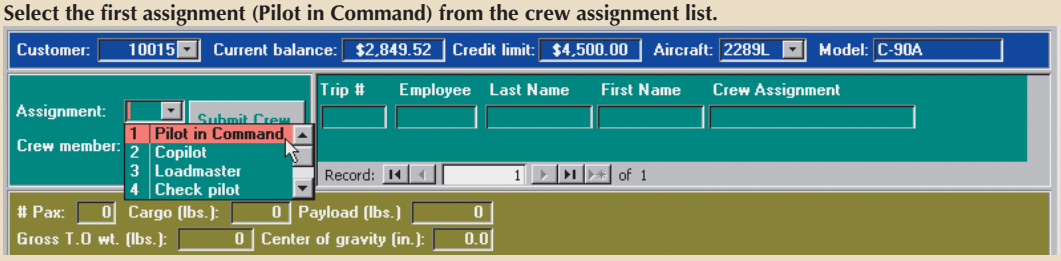


Select the customer.

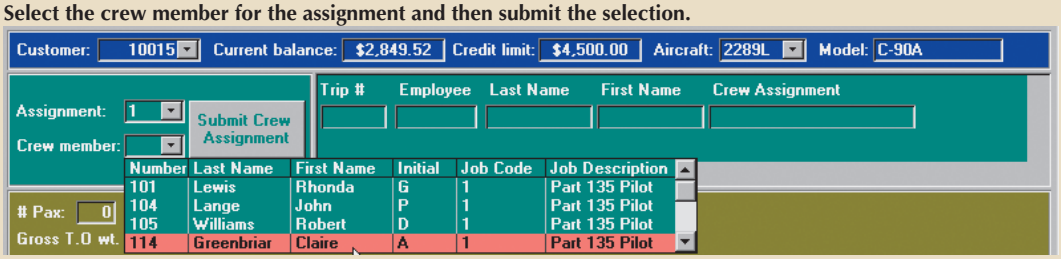
Select the aircraft.



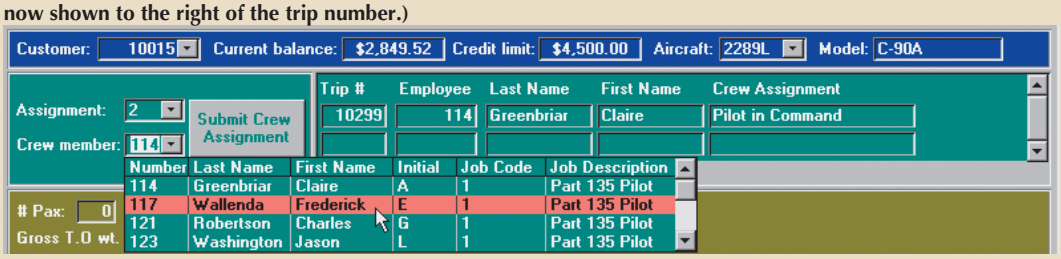
Select the first assignment (Pilot in Command) from the crew assignment list.



Select the crew member for the assignment and then submit the selection.

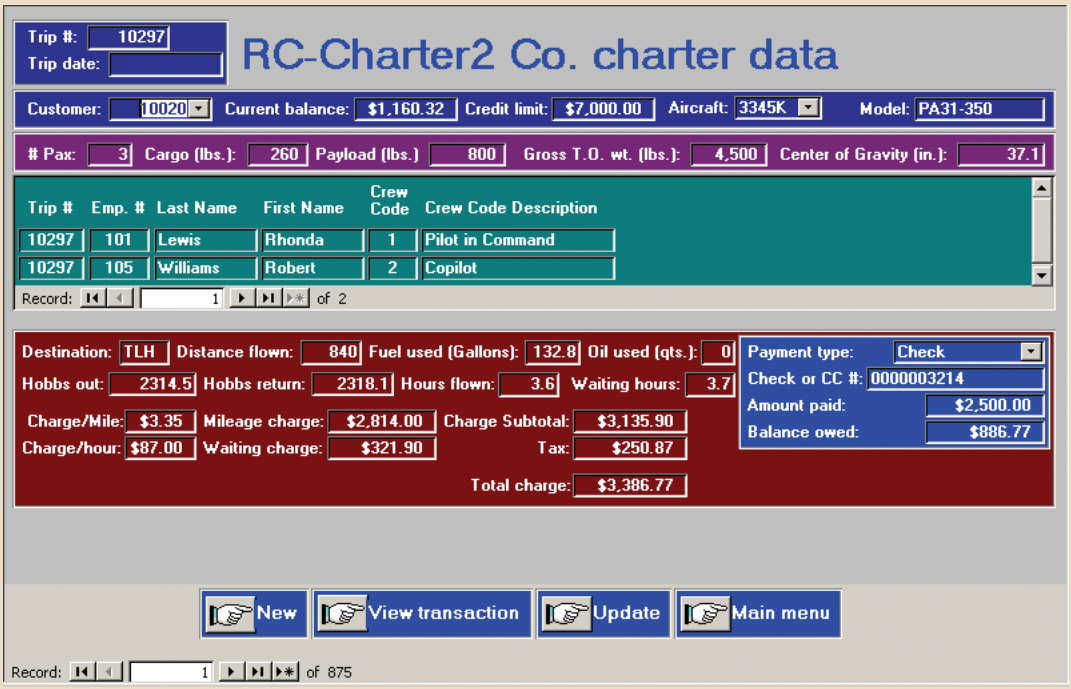


Select the next assignment and a crew member and then submit the selection. (Note that the first assignment is now shown to the right of the trip number.)



Complete at least the passenger and loading information and the destination. The remaining information is supplied at the conclusion of the trip. (See the bottom half of the form in F igure C.32.)

FIGURE C.34 SAMPLE COMPLETED CHARTER RECORD



Note

Keep in mind that even a beautifully crafted interface cannot overcome a poor database design. Unfortunately, too many organizations try to use applications development to overcome the limitations produced by poor database designs. Such attempts will lead to the inevitable failure of the organization's information requirements. (To use an analogy, you cannot overcome the problems of a poor building design by hiring better bricklayers—you just wind up with a poor building with beautiful brickwork.) That point is worth stressing—over and over and over!

As organizations become increasingly Internet-oriented, most of the database transaction interfaces tend to become Web interfaces. You were introduced to Web development issues in Chapter 15, "Database Connectivity and Web Technologies." Appendix J, "Web Database Development with ColdFusion," also discusses that topic.

C-6a Database Creation

All of the tables, indexes, and views that were defined during the logical design phase are created during this phase. The commands (or use utility programs) to create storage space and the access methods that were defined by the physical design are also issued at this time.

C-6b Database Loading and Conversion

The newly created database contains the (still empty) table structures. Those table structures can be filled by entering (typing) the data one table at a time or by copying the data

from existing databases or files. If the new table structures and formats are incompatible with those used by the original DBMS or file system software, the copy procedure requires the use of special loading or conversion utilities.

Because many processes require a precise sequencing of data activities, data are loaded in a specific order. Because of foreign key requirements, the University Computer Lab database must be initially loaded in the following order:

1. User, vendor, and location data.
2. Lab assistant and work schedule data.
3. Inventory type data.
4. Item data.

After those main entities have been loaded, the system is ready for testing and evaluation.

C-6c System Procedures

System procedures describe the steps required to manage, access, and maintain the database system. The development of those procedures constitutes a concurrent and parallel activity that started in the early stages of the system's design.

A well-run database environment requires and enforces strict standards and procedures to ensure that the data repository is managed in an organized manner. Although operational and management activities are logically connected, it is important to define distinct operations and management procedures.

In the case of the University Computer Lab Management System, procedures must be established to:

- Test and evaluate the database.
- Fine-tune the database.
- Ensure database security and integrity.
- Back up and recover the database.
- Access and use the database system.

Several databases may exist within a database environment. Each database must have its own set of system procedures and must be evaluated in terms of how it fits into the organization's information system.

C-7 Testing and Evaluation

The purpose of testing and evaluation is to determine how well the database meets its goals. Although testing and evaluation constitute a distinct DBLC phase, the implementation, testing, and evaluation of the database are concurrent and related. Database testing and evaluation should be ongoing. A database that is acceptable today may not be acceptable a few years from now because of rapidly changing information needs. In fact, an important reason for the relational database's growing dominance is its flexibility. (Because relational database tables are independent, changes can be made relatively quickly and efficiently.)

C-7a Performance Measures

Performance refers to the ability to retrieve information within a reasonable time and at a reasonable cost. A database system's performance can be affected by factors such as communication speeds, number of concurrent users, and resource limits. Performance,

measured primarily in terms of database query response time, is generally evaluated by computing the number of transactions per second. Performance issues are addressed in Chapter 11, "Database Performance Tuning and Query Optimization."

C-7b Security Measures

Security measures seek to ensure that data are safely stored in the database. Security is especially critical in a multiuser database environment, in which someone might deliberately enter inconsistent data. The DBA must define (with the help of end users) a company information policy that specifies how data are stored, accessed, and managed within a data security and privacy framework.

Access may be limited by using access rights or access authorizations. Such rights are assigned on the basis of the user's need to know or the user's system responsibilities. In the case of the UCL database, access rights must be assigned to LAs, the CLD, and the CLS. But those rights are limited. For example, the LAs may read their work schedules, but they are not able to modify the data stored in the LAB_ASSISTANT or HOURS_WORKED tables.

The database administrator may, for example, grant the use of a previously created LA_SCHED view to the lab assistant Anne Smith by using the following (SQL) syntax:

```
GRANT SELECT ON LA_SCHED TO LA_ASMITH;
```

In this case, *only* the LA identified as LA_ASMITH may use the view LA_SCHED to check the LA schedules. A similar procedure is used to enable other LAs to check the Lab schedules.

Physical security deals with controlling access to rooms or buildings where data reside or are processed. Imagine someone unplugging the main computer while several update transactions are being executed! Physical security also includes protection of the database environment against fire, earthquakes, and other calamities.

C-7c Backup and Recovery Procedures

Database backup is crucial to the continued availability of the database system after a database or hardware failure has occurred. Backup must be a formal and frequent procedure, and backup files should be located in predetermined sites. Recovery procedures must delineate the steps to ensure full recovery of the system after a system failure or physical disaster.

The UCL system's backup and recovery scenario includes the following procedures:

- Each computer in the system has an uninterrupted power supply to protect the computers against electrical interruptions.
- Periodic backups are made: daily for the most active tables and weekly for the less active tables. The backups are created during low system-use periods and are stored in different places to ensure physical safety.
- The database management system uses a transaction log to permit the recovery of the database from a given state when a disaster occurs.

C-8 Operation

Database operation, also called database management, is an ongoing venture, including all of the DBA's administrative and technical functions, designed to ensure the database's continuity. Before a database is declared operational, it must pass all operational tests and evaluations. The test and evaluation results must be formally approved by company management.

C-8a Database is Operational

An operational database provides all necessary support for the system's daily operations and maintains all appropriate operational procedures. If the database is properly designed and implemented, its existence not only enhances management's ability to obtain relevant information quickly, but also strengthens the entire organization's operational structure.

C-8b Operational Procedures

Database operational procedures are written documents in which the activities of the daily database operations are described. The operational procedures delineate the steps required to carry out specific tasks, such as data entry, database maintenance, backup and recovery procedures, and security measures.

C-8c Managing the Database: Maintenance and Evolution

After the database has become operational, management and control measures must be established for the database to be effective. The DBA is responsible for coordinating all operational and managerial aspects of the new DBMS environment. The DBA's responsibilities include:

- Monitoring and fine-tuning the database.
- Planning for and allocating resources for changes and enhancements.
- Planning for and allocating resources for periodic system upgrades.
- Providing preventive and corrective maintenance (backup and recovery procedures).
- Providing end-user management services by creating and defining users, passwords, privileges, and so on.
- Performing periodic security audits.
- Performing necessary training.
- Establishing and enforcing database standards.
- Marketing the database to the organization's users.
- Obtaining funding for database operations, upgrades, and enhancements.
- Ensuring completion of database projects within time and budget constraints.

In short, the DBA performs technical and managerial duties that ensure the proper operation of the database to support the organization's mission. Therefore, the DBA's activities are designed to support the end-user community through the creation and enforcement of database-related policies, procedures, standards, security, and integrity that, in turn, foster the generation and proper use of information. A more detailed discussion of the database administration function is provided in Chapter 16.

Key Terms

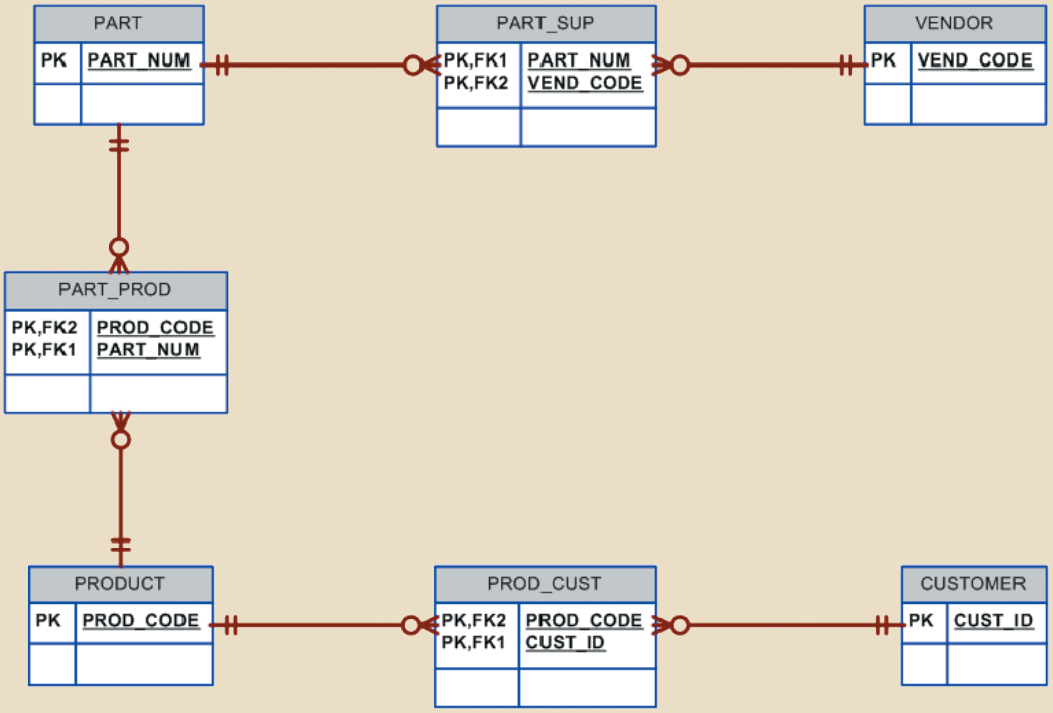
cluster tables, C-44

verification, C-1

Review Questions

1. Why must a conceptual model be verified? What steps are involved in the verification process?
2. What steps must be completed before the database design is fully implemented? (Make sure that you list the steps in the correct sequence and discuss each step briefly.)
3. What major factors should be addressed when database system performance is evaluated? Discuss each factor briefly.
4. How would you verify the ER diagram shown in Figure QC.4? Make specific recommendations.

FIGURE QC.4 THE ERD FOR QUESTION 4

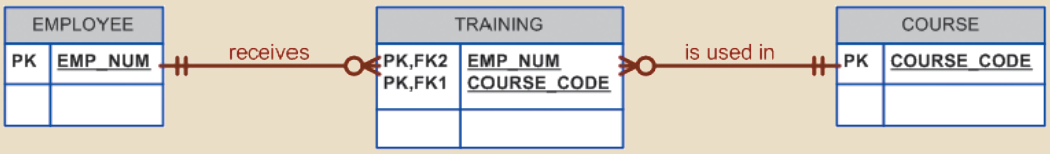


5. Describe and discuss the ER model's treatment of the UCL's inventory/order hierarchy:
   a. Category
   b. Class
   c. Type
   d. Subtype
6. Modern businesses tend to provide continuous training to keep their employees productive in a fast-changing and competitive world. In addition, government regulations often require certain types of training and periodic retraining. (For example, pilots must take semiannual courses involving weather, air regulations, and so on.) To make sure that an organization can track all training received by each of its employees, trace the development of the ERD segment in Figure QC.6 from the initial business rule that states:

An employee can take many courses, and each course can be taken by many employees.

Once you have traced the development of the ERD segment, verify it and then provide sample data for each of the three tables to illustrate how the design would be implemented.

FIGURE QC.6 THE ERD FOR QUESTION 6



7. You read in this appendix that *an examination of the UCL's Inventory Management module reporting requirements uncovered the following problems:*
   - The Inventory module generates three reports, one of which is an inventory movement report. But the inventory movements are spread across two different entities (CHECK_OUT and WITHDRAW). That spread makes it difficult to generate the output and reduces the system's performance.
   - An item's quantity on hand is updated with an inventory movement that can represent a purchase, a withdrawal, a check-out, a check-in, or an inventory adjustment. Yet only the withdrawals and check-outs are represented in the system.

   What solution was proposed for that set of problems? How would such a solution be useful in other types of inventory environments?

Problems

1. Verify the conceptual model you created in Appendix B, Problem 3. Create a data dictionary for the verified model.
2. Verify the conceptual model you created in Appendix B, Problem 4. Create a data dictionary for the verified model.
3. Verify the conceptual model you created in Appendix B, Problem 5. Create a data dictionary for the verified model.
4. Verify the conceptual model you created in Appendix B, Problem 6. Create a data dictionary for the verified model.
5. Verify the conceptual model you created in Appendix B, Problem 7. Create a data dictionary for the verified model.
6. Design (through the logical phase) a student-advising system that will enable an advisor to access a student's complete performance record at the university. A sample output screen should look like the one shown in Table PC.6.
7. Design and verify a database application for one of your local not-for-profit organizations (for example, the Red Cross, the Salvation Army, your church or synagogue). Create a data dictionary for the verified design.

TABLE PC.6

THE STUDENT TRANSCRIPT FOR PROBLEM 6

Name: Xxxxxxxxxxxxxxxx X Xxxxxxxxxxxxxxxx			**Page** # of ##
Department: Xxxxxxxxxxxxxxxxxxx	**Major:** Xxxxxxxxxxxxxxxxxxx		
Social Security Number: ###-##-####	**Report Date:** ## Xxxxxxxxxxxxxx ####		
Fall 2012			
Course	**Hours**	**Grade**	**Grade Points**
CIS 200 (Intro to Microcomputers)	3	B	##
XXX #### (Xxxxxxxxxxxxxxxxx)	#	X	##
XXX #### (Xxxxxxxxxxxxxxxxx)	#	X	##
XXX #### (Xxxxxxxxxxxxxxxxx)	#	X	##
XXX #### (Xxxxxxxxxxxxxxxxx)	#	X	##
Total this semester:	##	**GPA**	#.##
Total to date:	###	**Cumulative GPA**	#.##
Spring 2013			
Course	**Hours**	**Grade**	**Grade Points**
CIS 300 (Computers in Society)	3	B	##
XXX #### (Xxxxxxxxxxxxxxxxx)	#	X	##
XXX #### (Xxxxxxxxxxxxxxxxx)	#	X	##
XXX #### (Xxxxxxxxxxxxxxxxx)	#	X	##
XXX #### (Xxxxxxxxxxxxxxxxx)	#	X	##
Total this semester:	##	**GPA**	#.##
Total to date:	###	**Cumulative GPA**	#.##
Summer 2013			
Course	**Hours**	**Grade**	**Grade Points**
CIS 400 (Systems Analysis)	3	B	##
XXX #### (Xxxxxxxxxxxxxxxxx)	#	X	##
XXX #### (Xxxxxxxxxxxxxxxxx)	#	X	##
XXX #### (Xxxxxxxxxxxxxxxxx)	#	X	##
Total this semester:	##	**GPA**	#.##
Total to date:	###	**Cumulative GPA**	#.##

8. Using the information given in the physical design section (C-5), estimate the space requirements for the following entities:

   - RESERVATION
   - INV_TRANS
   - TR_ITEM
   - LOG
   - ITEM
   - INV_TYPE

   Hint: You may want to check Appendix B, Table B.3, A Sample Volume of Information Log.