5

Chapter 5 – Physical Data Modeling

In this chapter, you will learn about *physical data models* and how they differ from logical data models. You will also learn how to transform a logical data model into a physical data model. By the end of the chapter, you will learn how to create and use a physical data model to create database-specific DDL statements that can turn your data modeling exercises into a functional schema.

5.1 Physical data modeling: The big picture

Physical data models are data models that are specific to a database, and they are based on database specifications. With a physical data model, you can model storage that includes partitions, table spaces, or indexes, in addition to other storage objects.

You can model data by making use of specific properties available within a specific database and database platform. For example, physical data models will vary depending on whether they are modeled for DB2 systems, Oracle systems, or SQL Server systems. IBM InfoSphere Data Architect supports many different database system types and versions, and you can use a physical data model to create the database-specific data objects for these systems.

Note:

InfoSphere Data Architect only supports storage modeling for DB2 for Linux, UNIX, and Windows, DB2 for z/OS®, and DB2 for i5/OS® databases. You can still create tables, columns, and other related objects within the workbench, but data objects from other database vendors, such as table spaces and buffer pools, are not supported.

For this book, you will be working with models that are designed for a DB2 system.

Typically, when you work with a physical data model, you use the following task flow:

1. Create a physical data model, either from scratch, template, transforming from a logical data model, or importing from another system.
2. Refine the physical data model by adding storage objects, indexes, and views.
3. Generate the DDL to deploy the database and/or schema.
4. Run the DDL script that creates the data objects on the server.

So you might be asking yourself, “What are the differences between a logical and physical data model?” *Table 5.1* summarizes the differences.

Logical data models	Physical data models
Do not correspond to any database vendor	Always correspond to a database vendor and database version
Based on a higher level of abstraction; more generic	More specific; closer to the design of an actual database
Physical storage is not taken into account	Used to model physical storage for supported database vendors
Does not require database vendor construction knowledge; requires to know how the model will be used	Requires knowledge about database vendors and how models are constructed for each platform; requires information about how data

	will be stored

Table 5.1 – Differences between logical and physical data models

5.2 Creating a physical data model from scratch

Ideally, you would first create a logical data model before you create a physical data model. This helps you keep the logical and physical designs separate, allowing you to modify data models at the logical level before you design the physical storage aspects of a database or schema.

However, it is possible to create a blank physical data model within the workbench.

Note:

The steps below are provided for reference only. You do not need to perform these steps to continue with the design of the student information system. In the next section, you will learn how to transform to the physical data model.

To create a blank physical data model:

1. Click *File -> New -> Physical Data Model*. The New Physical Data Model wizard opens.
2. Complete the wizard. You can create a blank physical data model, or you can reverse-engineer an existing database, schema, or DDL script. Reverse-engineering is covered in a later chapter.
3. Use the workbench to add data objects, such as tables, schemas, or columns. Right-click on various objects within the data model to see which data objects you can add from the context menu, or create a diagram to visualize these data objects and their relationships.

5.3 Transforming a logical data model to a physical data model

Instead of creating a physical data model from scratch, you will use the existing logical data model (**`STUDENT_INFO_SYSTEM.ldm`**) that you created in the previous chapters and transform it to a physical data model. The transformation process creates the physical design of a database or schema for you, which saves you the work of manually creating tables, columns, and other data objects.

After you transform from the logical data model, you will add more specific properties to the physical data model and bring it one step closer to a functional, deployable data model.

Note:

The data types that are mapped from one data model to another depend on what you have specified in the Preferences window. To ensure that data types are mapped correctly between the logical data model and the new physical data model:

1. Open the Preferences window from the main menu by clicking *Window* > *Preferences*.
2. Expand the Data Management node and find the *Transform* > *Data Type Map* page.
3. Expand the Data Type Map node in the Preferences window to specify the options for data type mapping between logical and physical data models and the various database vendors.

To transform a logical data model into a physical data model:

1. Select the **`STUDENT_INFO_SYSTEM.ldm`** file in the data design project.
2. From the main menu, click *Data -> Transform -> Physical Data Model*. The Transform to Physical Data Model wizard opens.
3. On the Target Physical Data Model page, select the *Create new model* radio button, then click *Next*.
4. Complete the Physical Data Model File page:
   a. Specify the *University Info System* data design project as the destination folder.
   b. Specify *DB2 for Linux, UNIX, and Windows* as the database type.
   c. Select *V9.7* as the database version.
   d. Click *Next*.
5. On the Options page, in the *Schema name* field, specify **`STUDENT_LIFE`**, then click *Next*. The Options page is shown in *Figure 5.1*.

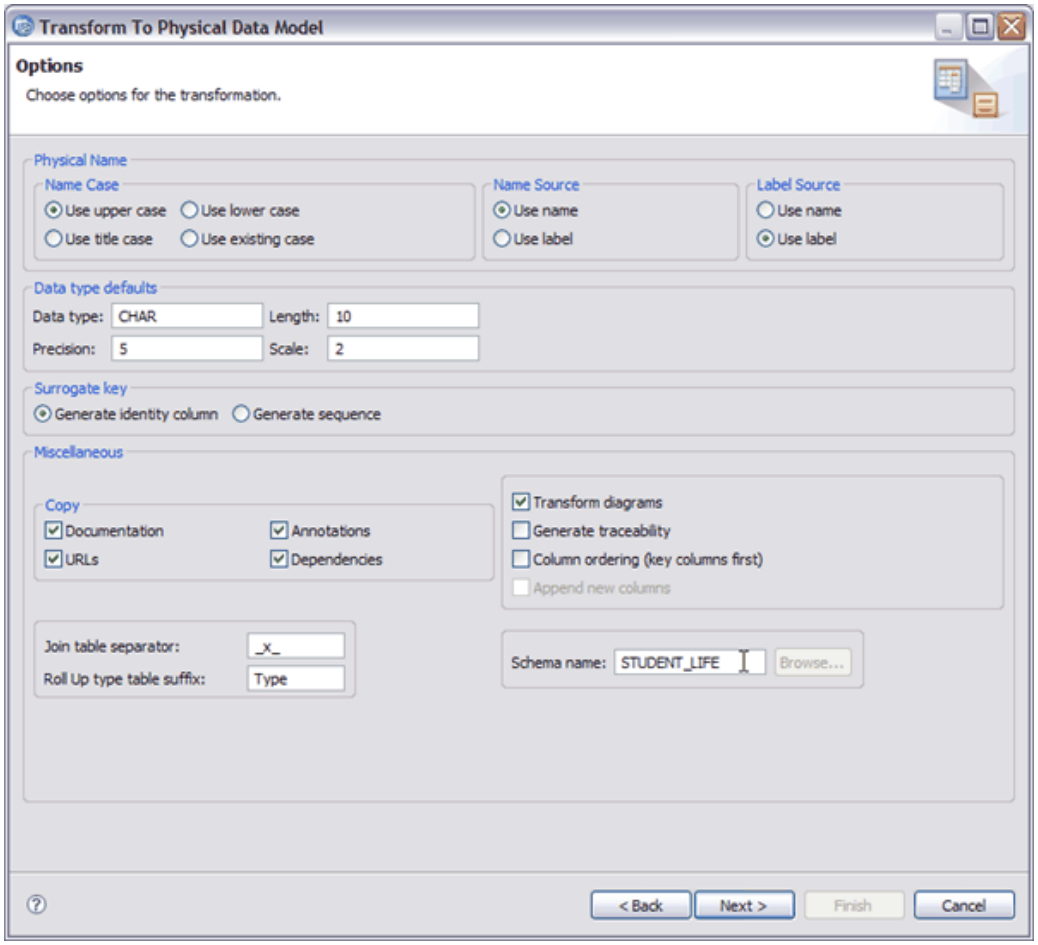

Figure 5.1 – Completing the Options page

Note:

As a best practice, you should select the *Generate traceability* option. This option allows you to identify dependencies between the logical and physical data models, which helps you perform impact analysis when refining your data models.

6. On the Output page, click *Finish*. The physical data model is created in the *Data Models* folder of the data design project, and the file opens in the editor view.
7. Give the new database a more descriptive name.
   a. Select the Database object under the physical data model.
   b. In the *General* tab of the Properties view, specify **`SAMPLE`** as the new name.
8. Save your work.

Figure 5.2 shows the data objects and folders that are created after you complete the transformation process.

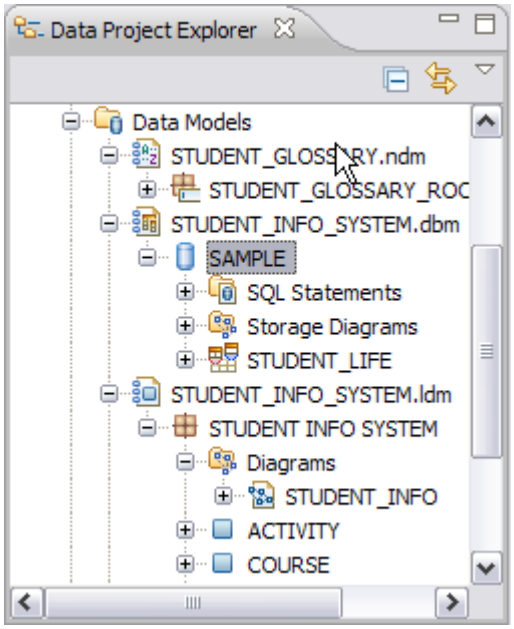

Figure 5.2 – New physical data model and data objects

5.4 Working on your physical data model

5.4.1 Anatomy of your model

Figure 5.3 shows you what a physical data model looks like.

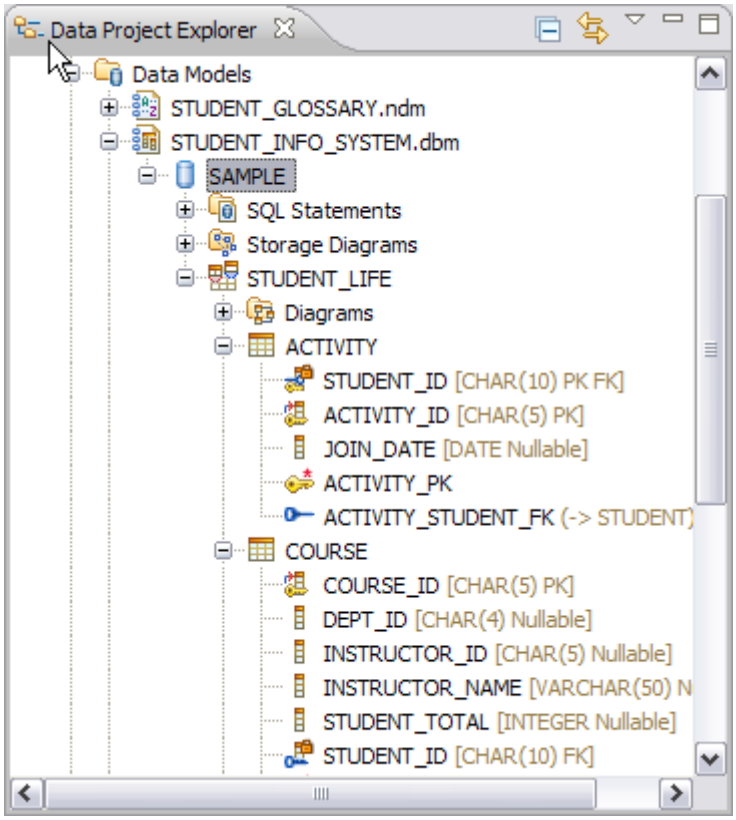

Figure 5.3 – The physical data model, displayed in the workbench

Every physical data model has the following characteristics:

- File extension: The file extension at the end of each physical data model file name is *.dbm*.
- Schema and database design:
  - The structure is more or less the same as a logical data model.
  - As you have learned previously, logical data models contain a single package, where you can store subpackages, entities, and diagrams. Physical data models, on the other hand, store information in databases. Each database can have multiple schemas, and each schema can contain multiple tables.
- Diagrams:
  - When you transform a diagram from a logical data model, your design is intact, including notes, formatting, and relationships. The only difference is in the icons used to depict the new tables, columns, and other data objects that you design.

- As with logical data models, you can create physical data model diagrams to help you visualize and understand the design of complex data models.

- Storage diagrams: You can create diagrams to help you model the storage of your DB2 databases. This includes partitions, table spaces, buffer pools, and other storage objects.
- SQL statements: The *SQL Scripts* folder stores SQL scripts that you create within the physical data model.

If you explore the physical data model, you will see that even the atomic domain elements that you specified are carried over into the new physical data model.

5.4.2 Storage modeling in DB2

Storage modeling for DB2 involves creating *table spaces*, *partitions*, and *partition groups* of a database setup.

Note:

With IBM InfoSphere Data Architect, you can only model the storage of a DB2 for Linux, UNIX, and Windows database, or DB2 for z/OS database.

For more information about the DB2 storage model, refer to the book *Getting started with DB2 Express-C*, that is part of this book series.

This chapter does not cover how to model storage in detail, because the schema we are creating is very small. However, this section does explain the basic concepts behind storage modeling and the information that you should keep in mind when you design the storage for complex data models.

5.4.2.1 Table spaces

The data for databases is stored in *table spaces*. Table spaces are database vendor-specific.

Note:

To learn more about table spaces in DB2 for Linux, UNIX, and Windows, visit the following URL:
http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.dbobj.doc/doc/c0004935.html

To learn more about table spaces in DB2 for z/OS:
http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=/com.ibm.db2z.doc.gloss/src/gloss/db2z_gloss.htm

Table spaces are typically managed in one of two ways:

- *System-managed space*: The operating system will manage the table spaces. Containers are defined as operating system files, and they are accessed using operating system calls. This means that the operating system handles the following actions:
  - Buffering of input and output
  - Space allocation
  - Automatic extension of the table space
- *Database-managed space*: The database manages the table spaces. Containers are defined as files, which will be fully allocated with the size given when the space is created, or as devices. When you use DB2 to manage your table spaces, it will handle the following actions:
  - Buffering of input and output (as much as the allocation method and operating system allows)
  - Extending containers when using ALTER TABLESPACE
  - Releasing unused portions of database-managed containers

Each table space has at least one container. A *container* belongs to a single table space (but a table space can have more than one container). You can only add a container to a system-managed table space when the database is partitioned, and the partition does not yet have a container allocated for that table space. When you add a container to a table space, the data is distributed across all of the containers.

When you specify a table space, you need to consider the following settings (found on the *General* tab of the Properties view of the table space):

- *Page size*: Defines the size of pages that are used for the table space. For DB2, the supported sizes include 4K, 8K, 16K, and 32K. The page size limits the row length and column count of the tables that can be placed in the table space. Table spaces are limited to 16,777,216 pages. Choosing a larger page size increases the capacity of the table space.
- *Extent size*: Specifies the number of pages that will be written to a container before the next container is used. The database manager cycles repeatedly through the containers as data is stored. This setting only has an effect when there are multiple containers for each table space.
- *Prefetch size*: Specifies the number of pages that are read from the table space when data prefetching is performed. Prefetching draws data that is needed by a query before they are actually referenced by the query. This allows the query to have the data ready when data input or output is performed. Prefetching is selected by the database manager when it determines that it needs to use sequential input and output, and prefetching the data will improve performance.

Best practice:

Set the prefetch size as a multiple of the extent size value and the number of table space containers. For example, if your extent size is 32 and there are 4 containers, then the prefetch size should be 128, 256, or other multiples of 128. If a table is used often, create a separate table space for that table to improve performance.

- *Overhead and transfer rate*: Determines the cost of input and output during query optimization. Values are measured in milliseconds, and they should be the average for all containers. *Overhead* is the time that is associated with input-output controller activity, disk seek time, and rotational latency. The *transfer rate* is the amount of time that is needed to read a page into memory. These values can be calculated based on hardware specifications.

5.4.2.2 Buffer pools

A *buffer pool* is associated with a single database, and it can be used by more than one table space. When you create a buffer pool for a table space, you must ensure that the table space size and the buffer pool page size are the same for all of the table spaces that the buffer pool services. A table space can use only one buffer pool.

Adequate buffer pool size is essential to good database performance, because it reduces the amount of input and output by the disk, which is the most time-consuming database operation. Large buffer pools also have an effect on query optimization, because more of the work can be done in memory.

5.4.2.3 Performance implications of storage modeling

When you design the storage of a schema or database, you should focus primarily on utilizing buffers and maximizing the amount of parallelism for input and output. Use the following task flow to ensure that your storage models are efficient:

1. Determine the constraints given by the table designs. You might have to use more than one regular table space.
2. Consider whether having the tables in large table spaces with different settings is likely to increase performance.
3. Design a tentative table space.
4. Consider using buffer pools, which might help you improve upon your table space design.
5. Allocate containers to the table spaces.
6. Refine the model, and verify that your design is solid by thoroughly testing and benchmarking the model.

As a general rule, start out with the simplest design. Only allow complexity if and when performance is impacted.

5.5 Refining the physical data model

Now that we have transformed a logical data model into a physical data model, we should make some final refinements before we generate the DDL to deploy the database. First, we will rearrange the columns of the STUDENT table into a more logical order. Then, we will create a role and add a user to the database.

5.5.1 Rearranging columns in a physical data model

When you created the logical data model, you were more concerned with modeling *what* attributes and data characteristics must go into each entity. Let's rearrange the columns of the STUDENT table to make the object more easily understood to an end user.

To rearrange the columns, perform the following steps:

1. Select the STUDENT table under the STUDENT_LIFE schema. The properties of the table display in the Properties view.
2. Open the *Columns* tab of the Properties view.
3. Rearrange the columns to identify vital student information first:
    a. Make the FIRST_NAME column the first column in the table. Select the FIRST_NAME column, then press the Move Column Up arrow ( ) to place it first in the table. The columns should now look like the image in *Figure 5.4*.

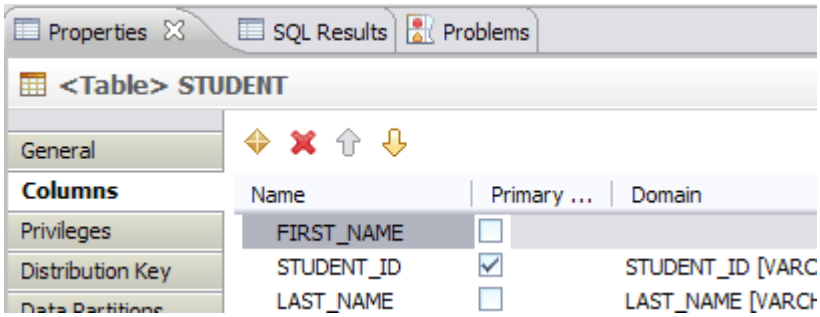

Figure 5.4 – Rearranging columns in a table

    b. Make the LAST_NAME column the second column in the table by moving it under the FIRST_NAME column.
    c. Make the STUDENT_SSN column the fourth column in the table.
    d. Make the EMAIL_ADDR column the fifth column in the table.

The columns should be in the order displayed in *Figure 5.5*.

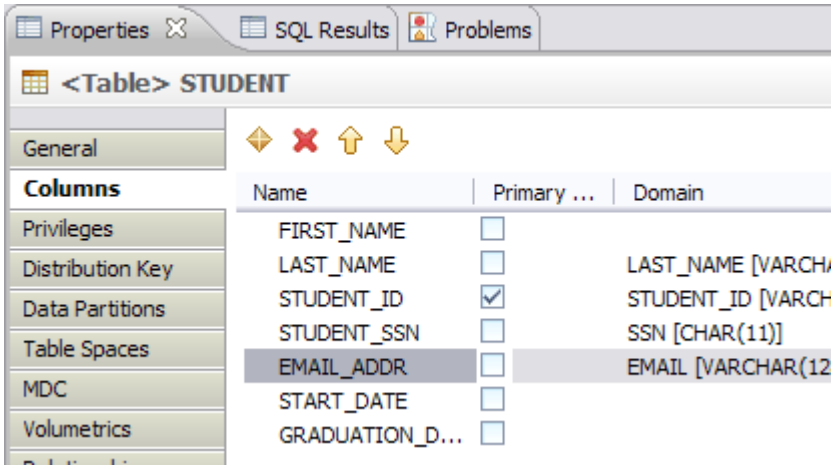

Figure 5.5 – The proper order of the columns in the table

4. Save your work.

5.5.2 Creating roles within the physical data model

Roles simplify the administration and management of privileges. A *role* is a database object that groups together one or more privileges and can be assigned to users, groups, PUBLIC, or other roles. Roles provide several advantages that make it easier to manage privileges in a database system:

- *Security administrators* (SECADM) can control access to their databases in a way that mirrors the structure of their organizations. They can create roles in the database that map directly to the job functions in their organizations.
- *Users* are granted membership in the roles that reflect their job responsibilities. As their job responsibilities change, their membership in roles can be easily *granted* and *revoked.*
- The assignment of privileges is simplified. Instead of granting the same set of privileges to each individual user in a particular job function, the administrator can grant this set of privileges to a role that represents that job function and then grant that role to each user in that job function.
- A role's privileges can be updated and all users who have been granted that role receive the update. The administrator does not need to update the privileges for every user on an individual basis.
- The privileges and authorities granted to roles are always used when you create views, triggers, materialized query tables (MQTs), static SQL and SQL routines, whereas privileges and authorities granted to groups (directly or indirectly) are not used. Because roles are managed inside the database, the DB2 database system can determine when authorization changes and act accordingly.

- All of the roles assigned to a user are enabled when that user establishes a connection, so all privileges and authorities granted to roles are taken into account when a user connects. Roles cannot be explicitly enabled or disabled.
- The security administrator can delegate management of a role to others.

All DB2 privileges and authorities that can be granted within a database can be granted to a role. For example, a role can be granted any of the following authorities and privileges:

- DBADM, SECADM, LOAD, and IMPLICIT_SCHEMA database authorities
- CONNECT, CREATETAB, CREATE_NOT_FENCED, BINDADD, CREATE_EXTERNAL_ROUTINE, or QUIESCE_CONNECT database authorities
- Any database object privilege (including CONTROL)

A user's roles are automatically enabled and considered for authorization when a user connects to a database; you do not need to activate a role by using the SET ROLE statement. For example, when you create a view, a materialized query table (MQT), a trigger, a package, or an SQL routine, the privileges that you gain through roles apply.

A role does not have an owner.

After speaking with your information system team, it is determined that you should create a role for application developers, so that they can connect to your system. This role that you create can be used to grant certain elevated privileges to the application developers so that they can deploy stored procedures, routines, and other data objects that easily access your schemas and databases.

To create the APP_DEV role:

1. Right-click on the SAMPLE database object, then select *Add Data Object -> Role*. A new role is created in the Data Project Explorer.
2. Name the role **APP_DEV**.
3. Add the privileges to the role:
   a. Open the *Privileges* tab of the Properties view. Make sure the *Database* tab is selected.
   b. Click the New ( ) button. The Grant New Privileges window opens.
   c. Select the BINDADD, CONNECT, CREATE_EXTERNAL_ROUTINE, and CREATE_NOT_FENCED_ROUTINE options, then click *OK*. The database privileges are added to the role.
4. Save your work.

The role is created within the physical data model. Once you deploy the DDL that you generate in section *5.6 DDL generation*, you can assign users to this role.

5.5.3 Adding a user ID to the physical data model

To ensure that specific users can access the database once it is deployed, you should add those user IDs to the physical data model. You can also add other users, but let's start with the default user1 ID that is created for most DB2 databases:

1. Right-click on the SAMPLE database and select *Add Data Object -> User*. A new user object is created in the physical data model.
2. Specify **`user1`** as the name of the user. An example of this is shown in *Figure 5.6*.

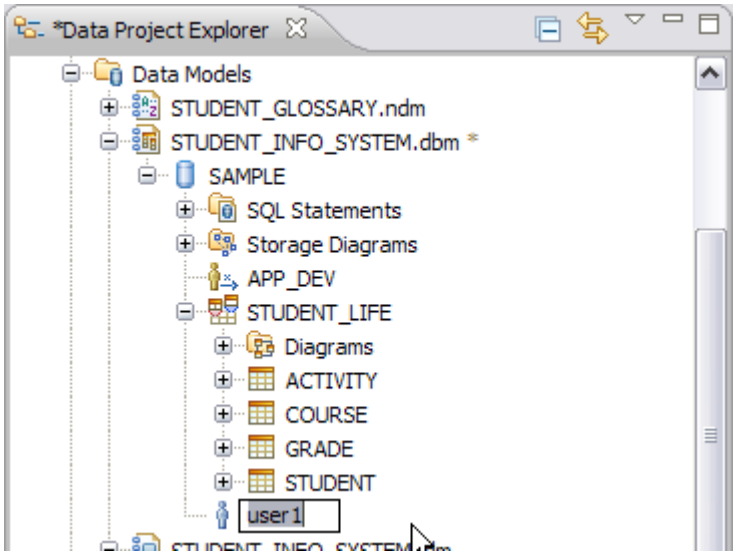

Figure 5.6 – Specifying a user

3. In the *Privileges* tab of the Properties view, make sure that you are in the *Database* section.
4. Click the New ( ) button. The Grant New Privileges window opens.
5. Select all of the privileges, then click *OK*.
6. Save your work.

The privileges are added to the user1 user ID. Now that user will have full access to the database.

5.5.4 Validating the physical data model

Before you deploy the physical data model, you should validate it. When you validate the model, the workbench checks the model to ensure that valid DDL can be generated from the data model. The validation process also ensures that your model not only adheres to the design standards that you have created, but that it adheres to the common data modeling standards. These “best practice” rules are built into the product, making it easier for you to ensure that your model is well-designed before you deploy it.

To validate the physical data model:

1. Right-click on the SAMPLE database in the Data Project Explorer, then select *Analyze Model*. The Analyze Model wizard opens. By default, all of the *Physical data model* rules are selected.
2. Deselect the *Dimensional Modeling* and *Naming Standard* options, then click *Finish*.

Note:

Dimensional modeling is not a topic that is covered in this book. For more information about dimensional modeling and its benefits, visit the IBM InfoSphere Data Architect information center at http://publib.boulder.ibm.com/infocenter/rdahelp/v7r5/index.jsp.

The model is analyzed, and you see a list of warnings and informational messages in the Problems view, as shown in *Figure 5.7*.

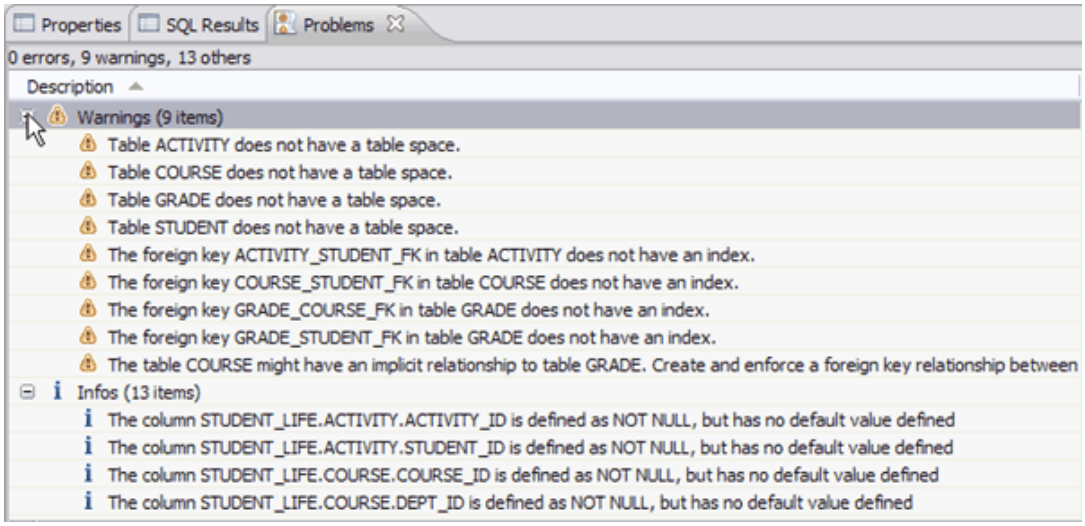

Figure 5.7 – List of warnings in the Problems view

For the purposes of this exercise, you do not have to take any steps to correct warnings and informational messages; these messages simply let you know of potential issues that can impact performance or alert you of where SQL statements might fail. For example, when information is INSERTed for the data objects in *Figure 5.7*, the update might fail, because no default value is explicitly set for the columns that are set to NOT NULL. Since your data model is relatively small and will not go into a real production environment, you do not need to correct these warnings by creating table spaces and indexes.

However, it is considered a best practice to evaluate each warning to ensure the best performance of your schemas and databases.

5.6 DDL generation

Now that you have generated a complete physical data model, you are aware of the process of creating, editing, and updating data models. Now that you've completed this process, how do you deploy the database to a test or production environment?

IBM InfoSphere Data Architect is capable of creating database-specific *data definition language* (*DDL*) scripts from the model that you have created. After you generate the DDL script, you can run it on the server, and users can access the model as part of the database.

5.6.1 Generating the DDL script from the database object

You generate the DDL script from the database object of the physical data model. The Generate DDL wizard will create the SQL statements to deploy your database on a server.

Let's create a DDL script to deploy your new database:

1. Connect to the SAMPLE database by right-clicking on the connection in the Data Source Explorer and selecting *Connect*.

   Note:

   If you are not deploying the DDL as part of the DDL generation, you do not need to connect to the database as the first step.

2. Right-click on the SAMPLE database in the Data Project Explorer, then select *Generate DDL*. The Generate DDL wizard opens.
3. On the Options page of the wizard, make sure that all of the options are selected, then click *Next*.

Table 5.1 explains the options on the Options page:

Option	Explanation
Check model	The model is validated before the script is created.
Fully qualified names	Any object is referenced by its full name. For example, the STUDENT table of the STUDENT_LIFE schema is referenced as STUDENT_LIFE.STUDENT.
Quoted identifiers	Specifies if you need to address object within quotes. This is done if you want to consider “Table A” and “TABLE A” as two different objects.
DROP statements	Includes DROP statements at the beginning of the DDL script to drop objects if they are

	already present.
CREATE statements	Includes CREATE statements in the DDL to create or re-create data objects.
COMMENT ON statements	Includes the statements from the *Documentation* tab of the Properties view of the data objects.
IN TABLESPACE clause	Takes info from the storage model or the default table space and adds it to the DDL script.

Table 5.1 – Available model elements that you can add to the DDL script

4. Keep the default settings on the Objects page, then click *Next*.
5. On the Save and Run DDL page, specify the following file name: **`DEPLOY_SYS.sql`**.

   Note:

   The DDL script is displayed in the *Preview DDL* section of the page. Use this section to verify that your script contains all of the SQL that you need to deploy the information system to the server. The first part of the SQL makes sure that none of the data objects exist on the server. Then, the SQL will re-create these data objects and make sure that the role you have specified is, in fact, created on the server.

6. Select the *Run DDL on server* option, then click *Next*.
7. Complete the Select Connection page by selecting the SAMPLE database.
8. Review your choices on the Summary page. You cannot change the options on this page. Once you are ready to generate the script, click *Next*.
9. On the Select Connection page, select the SAMPLE database connection, then click Next.
10. On the Summary page, click *Finish*.

The DDL script is generated and saved in the *SQL Scripts* folder of the data design project, as shown in *Figure 5.8*.

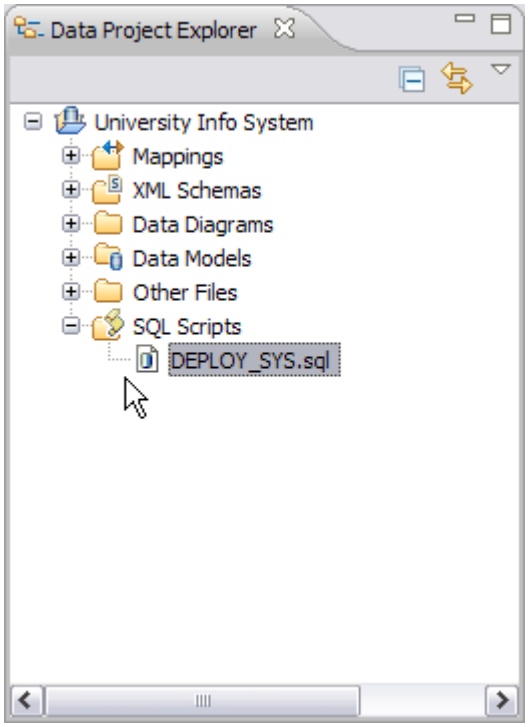

Figure 5.8 – The DDL script, saved in the *SQL Scripts* folder

The DDL script also runs on the server. This deploys your new database and schema to the DB2 system, where it can be populated with data, and application developers can create applications to query and report on the data. To verify that the schema has deployed, open the Data Source Explorer view, and expand the SAMPLE node and database. Locate and expand the *Schemas* folder and look for the STUDENT_LIFE schema, now created in the database. This is shown in *Figure 5.9*.

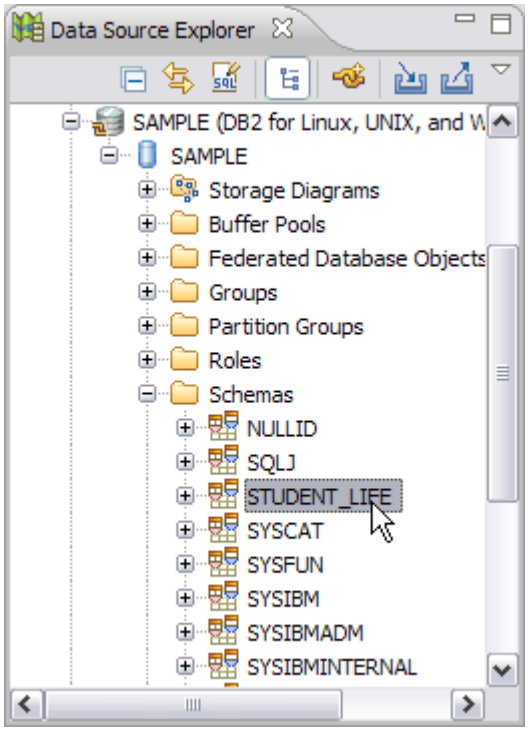

Figure 5.9 – Newly-deployed schema

5.7 Exercise

Generate a new DDL script for the physical data model:

1. On the Options page of the Generate DDL wizard, make sure the following items are not selected:
   - *DROP statements*
   - *COMMENT ON statements*
   - *Use domain if exists*
2. On the Objects page of the Generate DDL wizard, only generate statements for the following objects:
   - *Tables*
   - *Privileges*
   - *Primary key constraint*
   - *Roles*
3. Complete the Save and Run DDL page of the wizard:
   a. Specify **`EXERCISE_5_7.sql`** as the file name.

b. Select the *Open DDL file for editing* option.

4. Complete the wizard.
5. Select both SQL files that you have created (DEPLOY_SYS.sql and EXERCISE_5_7.sql), then right-click and select *Compare With -> Each Other*. The compare editor opens. What are the differences in the files?

5.8 Summary

In this chapter, you learned about physical data models. Physical data models are the next step in the data design process; they take the logical design and help you draft the physical storage (tables, columns, indexes, and other data objects) for the database and server.

You also learned about data definition language (DDL) files and how they can be used to deploy the data models that you create with IBM InfoSphere Data Architect. DDL scripts contain the queries that are necessary to drop, create, or modify data objects.

5.9 Review questions

1. What is a physical data model?
2. Physical data models can model one or all of the following objects:
   A. Tables
   B. Roles
   C. Schemas
   D. All of the above
3. From which objects of the physical data model do you generate DDL scripts?
4. True or false: You cannot preview the DDL script before you create it in your workspace.
5. What is the purpose of storage modeling?
6. What storage model objects can you create in a physical data model?
7. True or false: You must have a logical data model before you create a physical data model. You cannot create a physical data model from scratch.

5.10 What's next?

Congratulations! You have designed and deployed a complete data model.

In the next chapter, you will learn how to generate informative reports about your data models.