7.2 SQL Data Definition Language

As discussed in Chapter 1, the data definition language (DDL) is used by the database administrator (DBA) to express the database's logical, internal, and external data models. These definitions are stored in the catalog. In what follows, we discuss the key DDL concepts and illustrate them with an example. We revisit referential integrity constraints and elaborate on how to drop or alter database objects.

7.2.1 Key DDL Concepts

A key concept to start off with is the **SQL schema**. This is a grouping of tables and other database objects such as views, constraints, and indexes which logically belong together. An SQL schema is defined by a schema name and includes an **authorization identifier** to indicate the user, or user account, who owns the schema. Such users can perform any action they want within the context of the schema. A schema is typically defined for a business process or context such as a purchase order or HR system. Here, you can see the SQL definition of a schema called purchase whereby BBAESENS is assigned as the owner:

```
CREATE SCHEMA PURCHASE AUTHORIZATION
BBAESENS
```

Once we have defined a schema, we can start creating SQL tables. An SQL table implements a relation from the relational model. It typically has multiple columns, one per attribute type, and multiple rows, one for each tuple. An SQL table can be created using the CREATE TABLE statement followed by the name of the table. Below you can see two examples. The first one creates a table PRODUCT which is assigned to the default schema. The second example creates a table PRODUCT within the PURCHASE schema. It is recommended to explicitly assign a new table to an already-existing schema to avoid any confusion or inconsistencies.

```
CREATE TABLE PRODUCT …
CREATE TABLE PURCHASE.PRODUCT …
```

An SQL table has various columns – one per attribute type. As an example, our SQL table PRODUCT can have columns such as PRODNR, PRODNAME, PRODTYPE, etc. Each of these columns has a corresponding data type to represent the format and range of possible values. Table 7.1 gives some examples of commonly used SQL data types.

Table 7.1 SQL data types

Data type	Description
CHAR(n)	Holds a fixed length string with size *n*
VARCHAR(n)	Holds a variable length string with maximum size *n*
SMALLINT	Small integer (no decimal) between –32,768 and 32,767
INT	Integer (no decimal) between –2,147,483,648 and 2,147,483,647
FLOAT(n,d)	Small number with a floating decimal point. The total maximum number of digits is *n* with a maximum of *d* digits to the right of the decimal point.
DOUBLE(n,d)	Large number with a floating decimal point. The total maximum number of digits is *n* with a maximum of *d* digits to the right of the decimal point.
DATE	Date in format YYYY-MM-DD
DATETIME	Date and time in format YYYY-MM-DD HH:MI:SS
TIME	Time in format HH:MI:SS
BOOLEAN	True or false
BLOB	Binary large object (e.g., image, audio, video)

These data types might be implemented differently in various RDBMSs, and it is recommended to check the user manual for the options available.

In SQL it is also possible for a user to define a data type or domain. This can be handy when a domain can be re-used multiple times in a table or schema (e.g., see our BillOfMaterial example in Chapter 6). Changes to the domain definition then only need to be done once, which greatly improves the maintainability of your database schema. Here you can see an example of a domain PRODTYPE_DOMAIN, which is defined as a variable number of characters of which the value is either white, red, rose, or sparkling:

```
CREATE DOMAIN PRODTYPE_DOMAIN AS VARCHAR(10)
CHECK (VALUE IN ('white', 'red', 'rose', 'sparkling'))
```

If you would decide later to also include beer, then this can be easily added to the list of admissible values. Note, however, that some RDBMSs such as MySQL do not support the concept of an SQL domain.

SQL column definitions can be further refined by imposing column constraints (see also Chapter 6). The primary key constraint defines the primary key of the table. Remember, a primary key should have unique values and null values are thus not tolerated (entity integrity constraint). A foreign key constraint defines a foreign key of a table, which typically refers to a primary key of another (or the same) table, thereby restricting the range of possible values (referential integrity constraint). A **UNIQUE** constraint defines an alternative key of a table. A **NOT NULL** constraint prohibits null values for a column. A **DEFAULT** constraint can be used to set a default value for a column. Finally, a **CHECK** constraint can be used to define a constraint on the column values. All these constraints should be set in close collaboration between the database designer and business user.

7.2.2 DDL Example

Let's now illustrate the DDL concepts discussed before with an example. Figure 7.4 shows an ER model for a purchase order administration which we will also use to illustrate both SQL DDL and SQL DML in this chapter. Let's spend some time understanding it. We have three entity types: SUPPLIER, PURCHASE ORDER, and PRODUCT. A supplier has a unique supplier number, which is its key attribute type. It is also characterized by a supplier name, supplier address, supplier city, and supplier status. A purchase order has a unique purchase order number, which is its key attribute type. It also has a purchase order date. A product has a unique product number, which is its key attribute type. It also has a product name, product type, and available quantity.

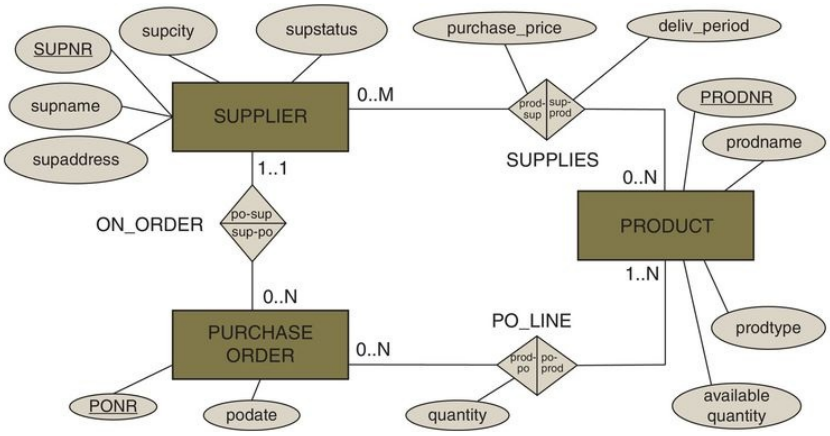

Figure 7.4 Example ER model.

Let's look at the relationship types. A supplier can supply minimum zero and maximum N products. A product is supplied by minimum zero and maximum M suppliers. The SUPPLIES relationship type has two attribute types: purchase_price and deliv_period, representing the price and period for a particular supplier to supply a particular product. A supplier has minimum zero and maximum N purchase orders on order. A purchase order is on order with

minimum one and maximum one – in other words, exactly one – supplier. PURCHASE ORDER is existence-dependent on SUPPLIER. A purchase order can have several purchase order lines, each for a particular product. This is the relationship type between PURCHASE ORDER and PRODUCT. A purchase order can have minimum one and maximum N products as purchase order lines. Vice versa, a product can be included in minimum zero and maximum N purchase orders. The relationship type is characterized by the quantity attribute type, representing the quantity of a particular product in a particular purchase order.

The corresponding relational tables for our ER model then become (primary keys are underlined; foreign keys are in italics):

SUPPLIER(SUPNR, SUPNAME, SUPADDRESS, SUPCITY, SUPSTATUS)
PRODUCT(PRODNR, PRODNAME, PRODTYPE, AVAILABLE_QUANTITY)
SUPPLIES(*SUPNR*, *PRODNR*, PURCHASE_PRICE, DELIV_PERIOD)
PURCHASE_ORDER(PONR, PODATE, *SUPNR*)
PO_LINE(*PONR*, *PRODNR*, QUANTITY)

The SUPPLIER and PRODUCT table correspond to the SUPPLIER and PRODUCT entity types. The SUPPLIES table is needed to implement the N:M relationship type between SUPPLIER and PRODUCT. Its primary key is a combination of two foreign keys: SUPNR and PRODNR. It also includes both the PURCHASE_PRICE and DELIV_PERIOD, which were the attribute types of the relationship type SUPPLIES. The PURCHASE_ORDER table corresponds to the PURCHASE_ORDER entity type in the ER model. It also has a foreign key SUPNR, which refers to the supplier number in the SUPPLIER

table. The PO_LINE table implements the N:M relationship type between PURCHASE_ORDER and PRODUCT. Its primary key is again a combination of two foreign keys: PONR and PRODNR. The QUANTITY attribute type is also included in this table.

The DDL definition for the SUPPLIER table becomes:

```
CREATE TABLE SUPPLIER
    (SUPNR CHAR(4) NOT NULL PRIMARY KEY,
    SUPNAME VARCHAR(40) NOT NULL,
    SUPADDRESS VARCHAR(50),
    SUPCITY VARCHAR(20),
    SUPSTATUS SMALLINT)
```

The SUPNR is defined as CHAR(4) and set to be the primary key. We could have also defined it using a number data type, but let's say the business asked us to define it as four characters so to also accommodate some older legacy product numbers, which may occasionally include alphanumeric symbols. SUPNAME is defined as a NOT NULL column.

The PRODUCT table can be defined as follows:

```
CREATE TABLE PRODUCT
    (PRODNR CHAR(6) NOT NULL PRIMARY KEY,
     PRODNAME VARCHAR(60) NOT NULL,
          CONSTRAINT UC1 UNIQUE(PRODNAME),
     PRODTYPE VARCHAR(10),
          CONSTRAINT CC1 CHECK(PRODTYPE IN ('white',
'red', 'rose', 'sparkling')),
     AVAILABLE_QUANTITY INTEGER)
```

In the PRODUCT table, the PRODNR column is defined as the primary key. The PRODNAME column is defined as NOT NULL and UNIQUE. Hence, it can be

used as an alternative key. The PRODTYPE column is defined as a variable number of characters up to ten. Its values should either be white, red, rose, or sparkling. The AVAILABLE_QUANTITY column is defined as an integer.

The DDL for the SUPPLIES table is:

```
CREATE TABLE SUPPLIES
    (SUPNR CHAR(4) NOT NULL,
    PRODNR CHAR(6) NOT NULL,
    PURCHASE_PRICE DOUBLE(8,2)
        COMMENT 'PURCHASE_PRICE IN EUR',
    DELIV_PERIOD INT
        COMMENT 'DELIV_PERIOD IN DAYS',
    PRIMARY KEY (SUPNR, PRODNR),
    FOREIGN KEY (SUPNR) REFERENCES SUPPLIER
(SUPNR)
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (PRODNR) REFERENCES PRODUCT
(PRODNR)
        ON DELETE CASCADE ON UPDATE CASCADE)
```

The SUPPLIES table has four columns. First, the SUPNR and PRODNR are defined. The PURCHASE_PRICE is defined as DOUBLE(8,2). It consists of a total of eight digits with two digits after the decimal point. The DELIV_PERIOD column is assigned the INT data type. The primary key is then defined as a combination of SUPNR and PRODNR. The foreign key relationship is then also specified. Neither of the foreign keys can be null since they both make up the primary key of the table. The ON UPDATE CASCADE and ON DELETE CASCADE statements are discussed below.

Next, we have the PURCHASE_ORDER table:

```
CREATE TABLE PURCHASE_ORDER
```

(PONR CHAR(7) **NOT NULL PRIMARY KEY**,
PODATE DATE,
SUPNR CHAR(4) **NOT NULL**,
FOREIGN KEY (SUPNR) **REFERENCES** SUPPLIER (SUPNR)
ON DELETE CASCADE ON UPDATE CASCADE)

The PURCHASE_ORDER table is defined with the PONR, PODATE, and SUPNR columns. The latter is again a foreign key.

We conclude our database definition with the PO_LINE table:

CREATE TABLE PO_LINE
(PONR CHAR(7) **NOT NULL**,
PRODNR CHAR(6) **NOT NULL**,
QUANTITY INTEGER,
PRIMARY KEY (PONR, PRODNR),
FOREIGN KEY (PONR) **REFERENCES** PURCHASE_ORDER (PONR)
ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (PRODNR) **REFERENCES** PRODUCT (PRODNR)
ON DELETE CASCADE ON UPDATE CASCADE);

It has three columns: PONR, PRODNR, and QUANTITY. Both PONR and PRODNR are foreign keys and make up the primary key of the table.

7.2.3 Referential Integrity Constraints

Remember, the referential integrity constraint states that a foreign key has the same domain as the primary key it refers to and occurs as either a value of the primary key or NULL (see Chapter 6). The question now arises of what should happen to foreign keys in the case that a primary key is updated or even deleted. As an example, suppose the supplier number of a SUPPLIER tuple is updated or the tuple is deleted in its entirety. What would have to happen to all other referring SUPPLIES and PURCHASE_ORDER tuples? This can be specified by using various referential integrity actions. The ON UPDATE CASCADE option says that an update should be cascaded to all referring tuples. Similarly, the ON DELETE CASCADE option says that a removal should be cascaded to all referring tuples. If the option is set to RESTRICT, the update or removal is halted if referring tuples exist. SET NULL implies that all foreign keys in the referring tuples are set to NULL. This obviously assumes that a NULL value is allowed. Finally, SET DEFAULT means that the foreign keys in the referring tuples should be set to their default value.

This is illustrated in Figure 7.5. We have listed some tuples of the SUPPLIER table. Let's focus on supplier number 37, whose name is Ad Fundum. This supplier has four referring SUPPLIES tuples and five referring PURCHASE_ORDER tuples. Suppose now that we update the supplier number to 40. In the case of an ON UPDATE CASCADE constraint, this update will be cascaded to all nine referring tuples where the supplier number will thus also be updated to 40. In the case of an ON UPDATE RESTRICT constraint, the update will not be allowed because of the referring tuples. If we now remove supplier number 37, then an ON DELETE CASCADE option will also remove all nine referring tuples. In the case of an ON DELETE RESTRICT constraint, the

removal will not be allowed. The constraints can be set individually for each foreign key in close collaboration with the business user.

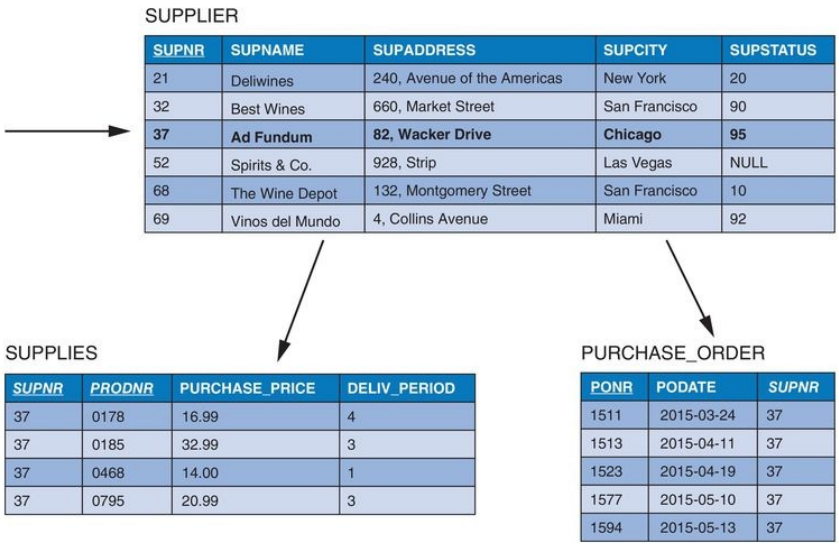

Figure 7.5 Referential integrity actions.

7.2.4 DROP and ALTER Command

The **DROP** command can be used to drop or remove database objects (e.g., schemas, tables, views, etc.). It can also be combined with the CASCADE and RESTRICT options. Some examples are:

DROP SCHEMA PURCHASE **CASCADE**
DROP SCHEMA PURCHASE **RESTRICT**
DROP TABLE PRODUCT **CASCADE**
DROP TABLE PRODUCT **RESTRICT**

The first statement drops the purchase schema. The CASCADE option indicates that all referring objects such as tables, views, indexes, etc. will also be automatically dropped. If the option had been RESTRICT, such as in the second example, the removal of the schema will be refused if there are still referring objects (e.g., tables, views, indexes). The same reasoning applies when dropping tables.

The **ALTER** statement can be used to modify table column definitions. Common actions are: adding or dropping a column; changing a column definition; or adding or dropping table constraints. Here you can see two examples:

ALTER TABLE PRODUCT **ADD** PRODIMAGE **BLOB**
ALTER TABLE SUPPLIER **ALTER** SUPSTATUS **SET DEFAULT** '10'

The first one adds the PRODIMAGE column to the PRODUCT table and defines it as **a binary large object**, or **BLOB**. The second example assigns a default value of 10 to the SUPSTATUS column.

Once we have finalized the data definitions, we can compile them so that they can be stored in the catalog of the RDBMS. The next step is to start populating the database. Figure 7.6 shows some examples of tuples listed for the various tables we defined earlier. We continue with the relational database for a wine purchase administration whereby the products represent wines.

SUPPLIER

SUPNR	SUPNAME	SUPADDRESS	SUPCITY	SUPSTATUS
21	Deliwines	240, Avenue of the Americas	New York	20
32	Best Wines	660, Market Street	San Francisco	90
...				

PRODUCT

PRODNR	PRODNAME	PRODTYPE	AVAILABLE_QUANTITY
0119	Chateau Miraval, Cotes de Provence Rose, 2015	rose	126
0154	Chateau Haut Brion, 2008	red	111
...		red	5

SUPPLIES

SUPNR	PRODNR	PURCHASE_PRICE	DELIV_PERIOD
21	0289	17.99	1
21	0327	56.00	6
...			

PURCHASE_ORDER

PONR	PODATE	SUPNR
1511	2015-03-24	37
1512	2015-04-10	94
...		

PO_LINE

PONR	PRODNR	QUANTITY
1511	0212	2
1511	0345	4
...		

Figure 7.6 Example relational database state.

Retention Questions

- Discuss how SQL DDL can be used to define schemas, tables, and domains.
- What types of referential integrity actions are supported in SQL?
- What is the purpose of the SQL DROP and ALTER commands?