Data Modeling and Database Design

Chapter 10:
Database Creation

Overview

- Structured Query Language (SQL)
- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language (DCL)

- SQL is a standard developed by ANSI and ISO
- SQL statements are case-insensitive
- The book uses SQL-92 (SQL2) and SQL-99

Database Creation

The principal task includes

- creation and modification of the database tables and other related structures
- enforcement of integrity constraints
- population of the database tables
- specification of security and authorizations as a means of access control and transaction processing controls

Data Definition in SQL

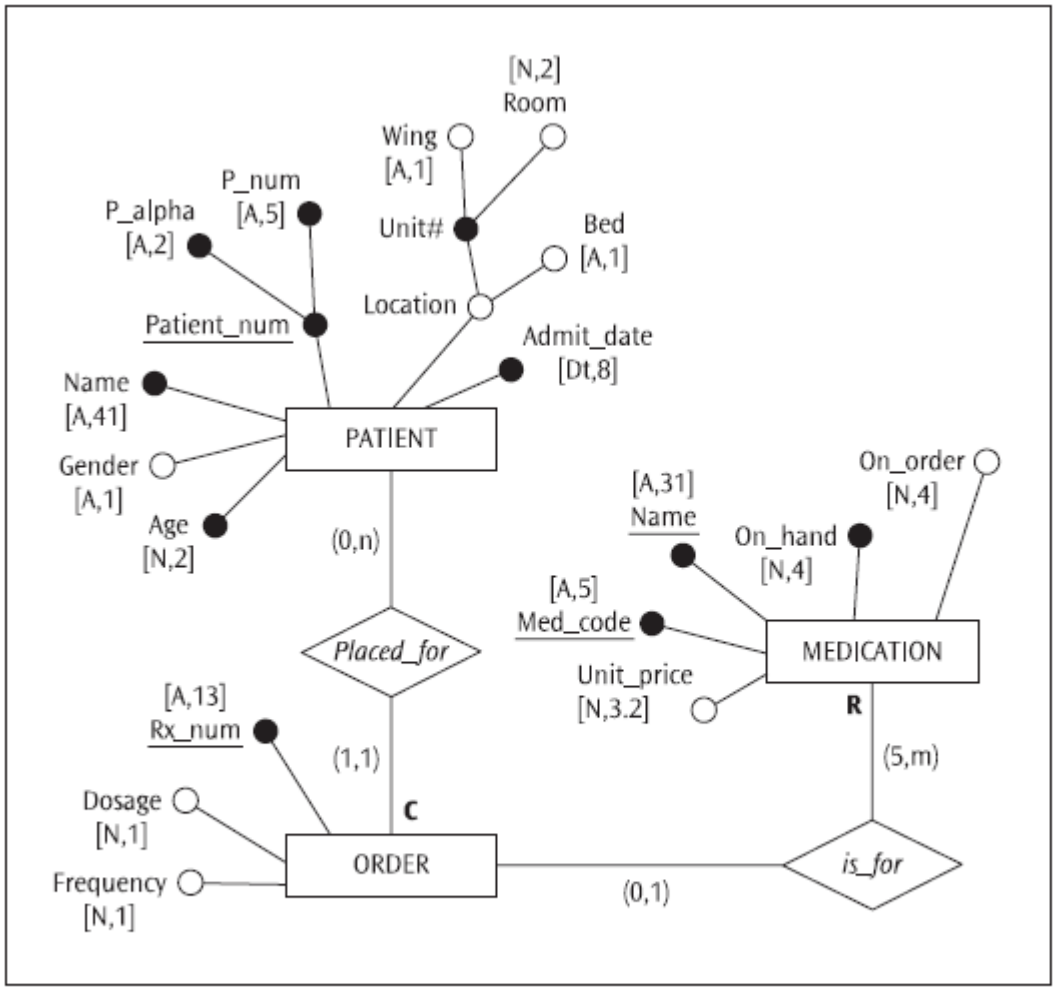
Three major constructs:

- Create
- Alter
- Drop

Applied to:

- Table
- Domain
- Schema
- Views

Example



> Constraint	Gender	IN ('M', 'F')
> Constraint	Age	IN (1 through 90)
> Constraint	Bed	IN ('A', 'B')
> Constraint	Unit_price	< 4.50
> Constraint	(Qty_onhand + Qty_onorder)	IN (1000 through 3000)
> Constraint	Dosage	DEFAULT 2
> Constraint	Dosage	IN (1 through 3)
> Constraint	Frequency	DEFAULT 1
> Constraint	Frequency	IN (1 through 3)

Figure 10.1a ER diagram: An excerpt from a hypothetical medical information system

Example (continued)

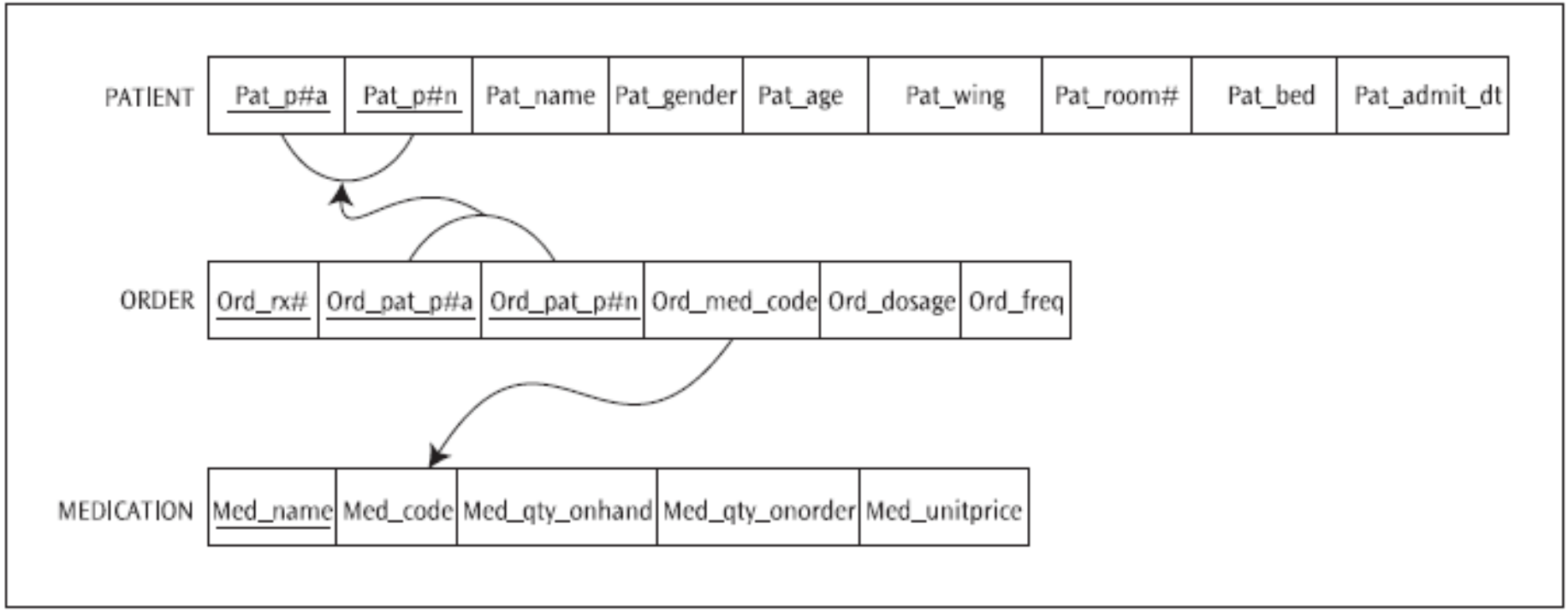


Figure 10.1c Relational schema for the ERD in Figure 10.1a

CREATE TABLE Commands: Box 1

```
CREATE TABLE patient
(Pat_p#a          char (2),
Pat_p#n           char (5),
Pat_name          varchar (41),
Pat_gender        char (1),
Pat_age           smallint,
Pat_admit_dt      date,
Pat_wing          char (1),
Pat_room#         integer,
Pat_bed           char (1)
);
```

```
CREATE TABLE medication
(Med_code         char (5),
Med_name          varchar (31),
Med_qty_onhand    integer,
Med_qty_onorder   integer,
Med_unitprice     decimal(3,2)
);
```

```
CREATE TABLE order
(Ord_rx#          char (13),
Ord_pat_p#a       char (2),
Ord_pat_p#n       char (5),
Ord_med_code      char (5),
Ord_dosage        smallint,
Ord_freq          smallint
);
```

CREATE TABLE Syntax

- CREATE TABLE *table_name* (*comma delimited list of table-elements*);
  where *table_name* is a user supplied name for the base TABLE and each *table-element* in the list is either a *column-definition* or a *constraint-definition*. The basic syntax for a *column-definition* is of the form:

  column_name representation [*default-definition*] [*column-constraint list*]

  where

  - *column_name* is a user supplied name for a COLUMN
  - *representation* specifies the relevant data type or alternatively the predefined *domain-name*
  - the optional (indicated by [ ]) *default-definition* specifies a default value for the COLUMN which overrides any default value specified in the domain definition, if applicable. In the absence of an explicit default definition (directly or via the domain definition), the implicit assumption of a NULL value for the default prevails. The *default-definition* is of the form: DEFAULT (*literal* | *niladic-function*| NULL)
  - the optional (indicated by [ ]) *column-constraint list* specifies *constraint-definition* for the column. The basic syntax for a *constraint-definition* follows the form:
    [CONSTRAINT *constraint_name*] *constraint-definition*

SQL-92 Data Types

Number Data Types	
numeric (p, s) where **p** indicates precision and **s** indicates scale	Exact numeric type—literal representation of number values—*decimal portion exactly the size dictated by the scale*—storage length = (precision + 1) when scale is > 0—most DBMSs have an upper limit on **p** (e.g., 28)
decimal (p, s) where **p** indicates precision and **s** indicates scale	Exact numeric type—literal representation of number values—*decimal portion at least the size of the scale; but expandable to limit set by the specific DBMS*—storage length = (precision + 1) when scale is > 0—most DBMSs have an upper limit on **p** (e.g., 28)
integer or **integer (p)** where **p** indicates precision	Exact numeric type—binary representation of large whole number values—often precision set by the DBMS vendor (e.g., 2 bytes)
smallint or **smallint (p)** where **p** indicates precision	Exact numeric type—binary representation of small whole number values—often precision set by the DBMS vendor (e.g., 1 byte)
float (p) where **p** indicates precision	Approximate numeric type—represents a given value in an exponential format—precision value represents the minimum size used, up to the maximum set by the DBMS
real	Approximate numeric type—represents a given value in an exponential format—has a default precision value set below that set for double precision data type by the DBMS
double precision	Approximate numeric type—represents a given value in an exponential format—has a default precision value set above that set for real data type by the DBMS

SQL-92 Data Types (continued)

String Data Types	
character (ℓ) or **char** (ℓ) where ℓ indicates length	Fixed length character strings including blanks from the defined language set SQL_TEXT within a database—can be compared to other columns of the same type with different lengths or varchar type with different maximum lengths—most DBMS have an upper limit on ℓ (e.g., 255)
character varying (ℓ) or **char** (ℓ) **varying** or **varchar** (ℓ) where ℓ indicates the maximum length	Variable length character strings except trailing blanks from the defined language set SQL_TEXT within a database—DBMS records actual length of column values—can be compared to other columns of the same type with different maximum lengths or char type with different lengths—most DBMS have an upper limit on ℓ (e.g., 2000)
bit (ℓ) where ℓ indicates length	Fixed length binary digits (0,1)—can be compared to other columns of the same type with different lengths or bit varying type with different maximum lengths
bit varying (ℓ) where ℓ indicates maximum length	Variable length binary digits (0,1)—can be compared to other columns of the same type with different maximum lengths or bit type with different lengths

SQL-92 Data Types (continued)

Date/Time & Interval Data Types	
date	10 characters long—format: yyyy-mm-dd—can be compared to only other date type columns—allowable dates conform to the Gregorian calendar
time (p)	Format: hh:mi:ss—sometimes precision (p) specified to indicate fractions of a second—the length of a TIME value is 8 characters, if there are no fractions of a second. Otherwise, the length is 8, plus the precision, plus one for the delimiter: hh:mi:ss.p—if no precision is specified, it is 0 by default—TIME can only be compared to other TIME data type columns
timestamp (p)	Format: yyyy:mm:dd hh:mi:ss.p—a timestamp length is nineteen characters, plus the precision, plus one for the precision delimiter—timestamp can only be compared to other timestamp data type columns
interval (q)	Represents measure of time—there are two types of intervals: year-month (yyyy:mm) which stores the year and month; and day-time (dd hh:mi:ss) which stores the days, hours, minutes, and seconds—the qualifier (q) known in some databases as the interval lead precision, dictates whether the interval is year-month or day-time—implementation of the qualifier value varies

CREATE TABLE: Things to Remember

- Data types supported by SQL-92 are grouped under *Number, String, Date/time & Interval*. However, DBMS vendors often build their own data types based on these four categories.
- Make sure that all attributes names are unique
- Some terms are reserved words, i.e., SQL uses these words in its language (e.g., order, grant, etc.)
- Attribute-level constraint vs. table-level constraint

CREATE TABLE Commands: Box 2

```
CREATE TABLE patient
(Pat_p#a        char (2),
Pat_p#n         char (5),
Pat_name        varchar (41),
Pat_gender      char (1),
Pat_age         smallint,
Pat_admit_dt    date,
Pat_wing        char (1),
Pat_room#       integer,
Pat_bed         char (1),
CONSTRAINT pk_pat PRIMARY KEY (Pat_p#a, Pat_p#n),
CONSTRAINT chk_gender CHECK (Pat_gender IN ('M', 'F')),
CONSTRAINT chk_age CHECK (Pat_age IN (1 through 90)),
CONSTRAINT chk_bed CHECK (Pat_bed IN ('A', 'B'))
);


CREATE TABLE medication
(Med_code          char (5),
Med_name           varchar (31) CONSTRAINT pk_med  PRIMARY KEY,
Med_unitprice      decimal (3,2) CONSTRAINT chk_unitprice CHECK (Med_unitprice < 4.50),
Med_qty_onhand     integer,
Med_qty_onorder    integer,
CONSTRAINT chk_qty CHECK ((Med_qty_onhand + Med_qty_onorder) BETWEEN 1000 AND 3000)
);


CREATE TABLE orders
(Ord_rx#           char (13) CONSTRAINT pk_ord  PRIMARY KEY,
Ord_pat_p#a        char (2),
Ord_pat_p#n        char (5),
Ord_med_code       char (5) CONSTRAINT fk_med FOREIGN KEY REFERENCES medication (med_code),
Ord_dosage         smallint DEFAULT 2 CONSTRAINT chk_dosage CHECK (Ord_dosage BETWEEN 1 AND 3),
Ord_freq           smallint DEFAULT 1 CONSTRAINT chk_freq CHECK (Ord_freq IN (1,2,3)),
CONSTRAINT fk_pat FOREIGN KEY (Ord_pat_p#a, Ord_pat_p#n)
REFERENCES patient (Pat_p#a, Pat_p#n)
);
```

Still Missing…

- Pat_name, Pat_age, Pat_admit_dt, Med_code and Med_qty_onhand are mandatory attributes – i.e., cannot have null values in any tuple
- Med_code is the alternate key since Med_name has been chosen as the primary key of medication table
- Participation of order in the *Placed_for* relationship is total
- Participation of patient in the *Placed_for* relationship is partial
- Participation of order in the is_*for* relationship is partial
- Participation of medication in the *is_for* relationship is total
- The deletion rule for the *Is_for* relationship is **restrict**
- The deletion rule for the *Placed_for* relationship is **cascade**
- [Pat_wing, Pat_room] is a molecular attribute
- [Pat_wing, Pat_room, Pat_bed] is a molecular attribute

Note: The cardinality ratio of the form (1, n) in a relationship type is implicitly captured in the DDL specification via the foreign key constraint. Any (1, 1) cardinality ratio can be implemented using the UNIQUE constraint definition.

CREATE TABLE Commands: Box 3

```
CREATE TABLE patient
(Pat_p#a        char (2),
Pat_p#n         char (5),
Pat_name        varchar (41) constraint nn_Patnm not null,
Pat_gender      char (1),
Pat_age         smallint constraint nn_Patage not null,
Pat_admit_dt    date constraint nn_Patadmdt not null,
Pat_wing        char (1),
Pat_room#       integer,
Pat_bed         char (1),
CONSTRAINT pk_pat PRIMARY KEY (Pat_p#a, Pat_p#n),
CONSTRAINT chk_gender CHECK (Pat_gender IN ('M', 'F')),
CONSTRAINT chk_age CHECK (Pat_age IN (1 through 90)),
CONSTRAINT chk_bed CHECK (Pat_bed IN ('A', 'B'))
);


CREATE TABLE medication
(Med_code        char (5) CONSTRAINT nn_medcd not null CONSTRAINT unq_med UNIQUE,
Med_name         varchar (31) CONSTRAINT pk_med  PRIMARY KEY,
Med_unitprice    decimal (3,2) CONSTRAINT chk_unitprice CHECK (Med_unitprice < 4.50),
Med_qty_onhand   integer CONSTRAINT nn_medqty not null,
Med_qty_onorder integer,
CONSTRAINT chk_qty CHECK ((Med_qty_onhand + Med_qty_onorder) BETWEEN 1000 AND 3000)
);


CREATE TABLE orders
(Ord_rx#         char (13) CONSTRAINT pk_ord  PRIMARY KEY,
Ord_pat_p#a      char (2) CONSTRAINT nn_ord_pat_p#a not null,
Ord_pat_p#n      char (5) CONSTRAINT nn_ord_pat_p#n not null,
Ord_med_code     char (5) CONSTRAINT fk_med REFERENCES medication (Med_code)
ON DELETE RESTRICT ON UPDATE RESTRICT,
Ord_dosage       smallint DEFAULT 2 CONSTRAINT chk_dosage CHECK (Ord_dosage BETWEEN 1 AND 3),
Ord_freq         smallint DEFAULT 1 CONSTRAINT chk_freq CHECK (Ord_freq IN (1, 2, 3)),
CONSTRAINT fk_pat FOREIGN KEY (Ord_pat_p#a, Ord_pat_p#n)
REFERENCES patient (Pat_p#a, Pat_p#n) ON DELETE CASCADE ON UPDATE CASCADE
);
```

ALTER TABLE Syntax

```
ALTER TABLE table_name action;
```

- where *table_name* is the name of the base TABLE being altered and the actions possible are:
  - Actions pertaining to alteration of a column via the syntax:
    - ADD [COLUMN] *column_definition*
    - ALTER [COLUMN] *column_name* { SET *default-definition* | DROP DEFAULT }
      - (Adds the *default-definition* or replaces an existing *default-definition*) or
      - (removes an existing *default-definition*)
    - DROP [COLUMN] *column_name* { RESTRICT | CASCADE }

  Or

  - Alteration of a previously specified table constraint in force via the syntax
    - ADD *table_constraint_definition*
      - (Permits addition to existing set of constraints, if any)
    - DROP CONSTRAINT *constraint_name* { RESTRICT | CASCADE }
      - (Removes the named constraint)

ALTER TABLE Examples

- Suppose we want to add a column to the base table patient to store the phone number of every patient. The DDL/SQL code to do this is:
  ALTER TABLE patient ADD Pat_phone# char (10);
- In order to delete the column from the base table, the following code applies:
  ALTER TABLE patient DROP Pat_phone# CASCADE;
  or
  ALTER TABLE patient DROP Pat_phone# RESTRICT;

ALTER TABLE Examples (continued)

- Suppose we want to specify a default value of $3.00 for the unit price of all medications. This can be done as follows:
  ALTER TABLE medication ALTER Med_unitprice SET DEFAULT 3.00;
- The default clause can be removed by:
  ALTER TABLE medication ALTER Med_unitprice DROP DEFAULT;

Best Practices

- Method 1:
  Pat_age smallint not null
- Method 2:
  Pat_age smallint constraint nn_Patage not null,
  CONSTRAINT *chk_age* CHECK *(*Pat_age IN (1 through 90))
- Method 3:
  Pat_age smallint not null CHECK *(*Pat_age IN (1 through 90))

Best Practices (continued)

- If we decide to permit null value for Pat_age, in Method 3, the whole column definition has to be re-specified. In Method 2, we simply drop the “not null” constraint as shown below:
-

```
ALTER TABLE patient  DROP CONSTRAINT nn_patage
CASCADE;   or
ALTER TABLE patient  DROP CONSTRAINT nn_patage
RESTRICT;
```

DROP TABLE

```
DROP TABLE table_name drop behavior;
```

- Where *table_name* is the name for the base TABLE being deleted and the drop behaviors possible are: CASCADE or RESTRICT.
- Example: DROP TABLE medication CASCADE;

CREATE, ALTER, DROP DOMAIN

- The SQL-92 standard provides for the formal specification of a *Domain*. A domain specification can be used to define a constraint over one or more columns of a table with a formal name so that the domain name can be used wherever that constraint is applicable.

CREATE DOMAIN Syntax

```
CREATE DOMAIN domain_name [ AS ] data type [ default-definition ]
     [ domain-constraint-definition list ];
```

- where
  - *domain_name* is a user supplied name for the domain
  - the optional *default-definition* specifies a default value for the DOMAIN. In the absence of an explicit default definition, the domain has **no** default value
  - the optional *domain-constraint-definition* is of the form: [CONSTRAINT *constraint_name*] CHECK (VALUE (*conditional-expression*))

Example 1

- Specify a domain to capture integer values 1, 2 and 3 with a default value of 2:
  CREATE DOMAIN measure smallint DEFAULT 2 CONSTRAINT *chk_measure* CHECK (VALUE IN (1,2,3));

- Since [CONSTRAINT *constraint_name*] is optional the above statement can also be stated as:
  CREATE DOMAIN measure smallint DEFAULT 2 CHECK (VALUE IN (1,2,3));

- Now, for instance, the column definition in the orders table
  Ord_dosage smallint DEFAULT 2 CONSTRAINT *chk_dosage* CHECK (Ord_dosage BETWEEN 1 AND 3) can be coded as
  Ord_dosage measure

- Likewise, Ord_freq smallint DEFAULT 1 CONSTRAINT *chk_freq* CHECK (Ord_freq IN (1, 2, 3)) can also be coded as
  Ord_freq measure DEFAULT 1

Example 1 (continued)

```
CREATE TABLE orders
 (Ord_rx#           char (13) CONSTRAINT pk_ord  PRIMARY KEY,
Ord_pat_p#a         char (2), CONSTRAINT nn_ord_pat_p#a not null,
Ord_pat_p#n         char (5), CONSTRAINT nn_ord_pat_p#n not null,
Ord_med_code        char (5) CONSTRAINT fk_med REFERENCES medication (Med_code)
ON DELETE RESTRICT ON UPDATE RESTRICT,
Ord_dosage          measure,
Ord_freq            measure DEFAULT 1,
CONSTRAINT fk_pat FOREIGN KEY (Ord_pat_p#a, Ord_pat_p#n)
REFERENCES patient (Pat_p#a, Pat_p#n) ON DELETE CASCADE ON UPDATE CASCADE
);
```

Example 2

- Specify a domain with mandatory values for the U.S. postal abbreviation for the list of states OH, PA, IL, IN, KY, WV, MI. Also, designate OH as the default state.

```
CREATE DOMAIN valid_states CHAR (2) DEFAULT 'OH'
CONSTRAINT nn_states CHECK (VALUE IS NOT NULL)
CONSTRAINT chk_states CHECK ( VALUE IN ('OH', 'PA', 'IL',
'IN', 'KY', 'WV', 'MI'));
```

Example 3

- Remove the ‘not null’ constraint from the domain specification for valid_states.

  ALTER DOMAIN valid_states DROP CONSTRAINT *nn_states*;

- **Note**: *Observe that had we not named the constraint, the only recourse we have is to get rid of the complete domain definition and create it over again.*

ALTER DOMAIN Syntax

- ALTER DOMAIN *domain_name action*;
  where *domain_name* is the name of the DOMAIN being altered and the actions possible are:
  - Alteration of domain default value via the syntax:
    - SET *default-definition*
      - (Adds the *default-definition* or replaces an existing *default-definition*), or
    - DROP DEFAULT
      - (Copies the default definition to the columns defined on the domain which do not have explicitly specified default values of their own and then removes *default-definition* from the domain definition.

  Or
  - Alteration of a previously specified domain constraint in force via the syntax
    - ADD *domain_constraint_definition*
      - (Permits addition to existing set of constraints, if any)
    - DROP CONSTRAINT *constraint_name*
      - (Removes the named constraint)

Example 4

- Suppose we want to add Maryland (MD) and Virginia (VA) to the domain valid_states. This is done by adding a new constraint to the domain valid_states as shown below:

  ALTER DOMAIN valid_states CONSTRAINT *chk_2more* CHECK (VALUE IN ('MD', 'VA'));

Example 5

- Assuming that the scripts in Example 1 and Box 4 have been executed, remove the default value for the DOMAIN measure. As of now, the default for measure is **2**, the default for Ord_dosage is **2**, and the default for ord_freq is **1**. The DDL/SQL syntax follows:

  ALTER DOMAIN measure DROP DEFAULT;

Example 6

- Change the default value of State in the valid_states domain to PA.
  ALTER DOMAIN valid_states SET DEFAULT 'PA';
- A domain definition can be eliminated using the DDL/SQL syntax:
  DROP DOMAIN *domain_name* CASCADE| RESTRICT
- Note:
  With the restrict option any attempt to drop a domain definition fails if any column in the database tables, views and/or integrity constraints references the domain name. With the *cascade* option, however, dropping a domain entails dropping of any referencing views and integrity constraints only. The columns referencing the domain are **not** dropped. Instead the domain constraints are effectively converted into base table constraints and are attached to every base table that has a column(s) defined on the domain.

Example 7

- The domain named measure is no longer needed

  DROP DOMAIN measure RESTRICT;

- Since the columns Ord_dosage and Ord_freq are defined on the DOMAIN measure, the DROP DOMAIN operation will fail with the drop behavior specification of RESTRICT. Therefore:

  DROP DOMAIN measure *cascade*;

Data Population Using SQL

Sample Tables

```
CREATE TABLE patient
(Pat_p#a        char (2),
Pat_p#n         char (5),
Pat_name        varchar (41) constraint nn_Patnm not null,
Pat_gender      char (1),
Pat_age         smallint constraint nn_Patage not null,
Pat_admit_dt    date constraint nn_Patadmdt not null,
Pat_wing        char (1),
Pat_room#       integer,
Pat_bed         char (1),
CONSTRAINT pk_pat PRIMARY KEY (Pat_p#a, Pat_p#n),
CONSTRAINT chk_gender CHECK (Pat_gender IN ('M', 'F')),
CONSTRAINT chk_age CHECK (Pat_age IN (1 through 90)),
CONSTRAINT chk_bed CHECK (Pat_bed IN ('A', 'B'))
);


CREATE TABLE medication
(Med_code        char (5) CONSTRAINT nn_medcd not null CONSTRAINT unq_med UNIQUE,
Med_name         varchar (31) CONSTRAINT pk_med  PRIMARY KEY,
Med_unitprice    decimal (3,2) CONSTRAINT chk_unitprice CHECK (Med_unitprice < 4.50),
Med_qty_onhand   integer CONSTRAINT nn_medqty not null,
Med_qty_onorder integer,
CONSTRAINT chk_qty CHECK ((Med_qty_onhand + Med_qty_onorder) BETWEEN 1000 AND 3000)
);


CREATE TABLE orders
(Ord_rx#         char (13) CONSTRAINT pk_ord  PRIMARY KEY,
Ord_pat_p#a      char (2) CONSTRAINT nn_ord_pat_p#a not null,
Ord_pat_p#n      char (5) CONSTRAINT nn_ord_pat_p#n not null,
Ord_med_code     char (5) CONSTRAINT fk_med REFERENCES medication (Med_code)
ON DELETE RESTRICT ON UPDATE RESTRICT,
Ord_dosage       smallint DEFAULT 2 CONSTRAINT chk_dosage CHECK (Ord_dosage BETWEEN 1 AND 3),
Ord_freq         smallint DEFAULT 1 CONSTRAINT chk_freq CHECK (Ord_freq IN (1, 2, 3)),
CONSTRAINT fk_pat FOREIGN KEY (Ord_pat_p#a, Ord_pat_p#n)
REFERENCES patient (Pat_p#a, Pat_p#n) ON DELETE CASCADE ON UPDATE CASCADE
);
```

INSERT

- Single-row INSERT – adds a single row of data to a table
- Multi-row INSERT – extracts rows of data from another part of the database and adds them to a table.

```
INSERT INTO <table-name> [(column-name {, column-name})]
VALUES (expression {, expression})
```

```
INSERT INTO <table-name> [(column-name {, column-name})]
<select-statement>[9]
```

INSERT Example

```
INSERT INTO PATIENT VALUES ('DB','77642','Davis, Bill', 'M', 27, '2007-07-
07', 'B', 108, 'B');10

1 row created.

INSERT INTO MEDICATION VALUES ('TAG', 'Tagament', 3.00, 3000, 0);

1 row created.

INSERT INTO ORDERS VALUES ('104', 'DB', '77642', 'TAG', 3, 1);

1 row created.
```

DELETE

- The DELETE statement removes selected rows of data from a single table
- Syntax:

- ```
  DELETE FROM <table-name> [WHERE <search-condition>]
  ```
  DELETE statement of the form DELETE FROM <table-name> can be used to delete all rows in a table.
- When used in this manner, while the target table has no rows after execution of the deletion, the table still exists and new rows can still be inserted into the table with the INSERT statement. To erase the table definition from the database, the DROP TABLE statement must be used.
  ```

DELETE Example

```
SELECT * FROM PATIENT;

Pat_p#a  Pat_p#n  Pat_name          Pat_gender    Pat_age Pat_admit_dt Pat_wing  Pat_room# Pat_bed
-------- -------- ------------------- ---------- ---------- ------------ -------- ---------- ---
DB       77642    Davis, Bill       M                  27 2007-07-07   B               108 B
GD       72222    Grimes, David                        44 2007-07-12

SELECT * FROM MEDICATION;

Med_code Med_name                        Med_unitprice Med_qty_onhand Med_qty_onorder
 -------- ------------------------------ ------------ -------------- ---------------
TAG      Tagament                                   3           3000               0
SELECT * FROM ORDERS;
Ord_rx        Ord_pat_p#a  Ord_pat_p#n  Ord_med_code Ord_dosage   Ord_freq
 ------------ ------------ ------------ ------------ ---------- ----------
 104          DB           77642        TAG                   3          1
```

DELETE Example (continued)

```
DELETE FROM PATIENT WHERE PATIENT.PAT_NAME LIKE '%Davis, Bill%';[12]

1 row deleted.
```

Content of Tables After Deletion

```
SELECT * FROM PATIENT;

Pat_p#a  Pat_p#n Pat_name         Pat_gender    Pat_age Pat_admit_dt Pat_wing  Pat_room# Pat_bed
------- ------- ----------------- ---------- ---------- ------------ -------- ---------- ------
GD       72222   Grimes, David                      44 2007-07-12

 SELECT * FROM MEDICATION;

 Med_code Med_name                       Med_unitprice Med_qty_onhand Med_qty_onorder
 -------- ------------------------------ ------------- -------------- ---------------
 TAG      Tagament                                   3           3000               0

 SELECT * FROM ORDERS;

 no rows selected
```

UPDATE

- The UPDATE statement modifies the values of one or more columns in selected rows of a single table.
- Syntax:

```
UPDATE <table-name>
SET column-name = expression
        {, column-name = expression}
[WHERE <search-condition>]
```

- The SET clause specifies which columns are to be updated and calculates the new values for the columns.
- It is important that an UPDATE statement not violate any existing constraints.

Access Control

Access Control

- SELECT – permission to retrieve data from a table or view;
- INSERT – permission to insert rows into a table or view;
- UPDATE – permission to modify column values of a row in a table or view;
- DELETE – permission to delete rows of data in a table or view;
- REFERENCES – permission to reference columns of a named in integrity constraints;
- USAGE – permission to use domains, collation sequences, character sets, and translations

GRANT

- The GRANT statement is used to grant privileges on database objects to specific users. The format of the GRANT statement is:

```
GRANT {Privilege-list |ALL PRIVILEGES}
ON Object-name
TO  {User-list | PUBLIC}
[WITH GRANT OPTION]
```

- *Privilege-list* can consists of one or more of the following privileges, separated by commas:

  SELECT
  DELETE
  INSERT [(*Column-name* [, …])]
  UPDATE [(*Column-name* [, …])]
  REFERENCES [(*Column-name* [, …])]
  USAGE

REVOKE

- The REVOKE statement is used to take away all or some of the privileges previously granted to another user or users. The format of the REVOKE statement is:

```
REVOKE [GRANT OPTION FOR] {Privilege-list | ALL PRIVILEGES}
ON Object-name
FROM [{User-list | PUBLIC} RESTRICT | CASCADE]
```

Example 1

- USER_A grants an assortment of privileges on the ORDERS, PATIENT, and MEDICATION tables to USER_B, USER_C, and USER_D.

```
GRANT SELECT, INSERT, DELETE, UPDATE
ON ORDERS
TO USER_B;

Grant succeeded.

GRANT SELECT, INSERT
ON PATIENT
TO USER_C;

Grant succeeded.

GRANT INSERT, DELETE
ON MEDICATION
TO USER_D;

Grant succeeded.
```

USER_A's table privileges granted as recorded in the System Catalog

GRANTEE	OWNER	TABLE_NAME	GRANTOR	PRIVILEGE	GRANTABLE
USER_B	USER_A	ORDERS	USER_A	DELETE	NO
USER_B	USER_A	ORDERS	USER_A	INSERT	NO
USER_B	USER_A	ORDERS	USER_A	SELECT	NO
USER_B	USER_A	ORDERS	USER_A	UPDATE	NO
USER_C	USER_A	PATIENT	USER_A	INSERT	NO
USER_C	USER_A	PATIENT	USER_A	SELECT	NO
USER_D	USER_A	MEDICATION	USER_A	DELETE	NO
USER_D	USER_A	MEDICATION	USER_A	INSERT	NO

Example 2

- USER_A grants all privileges on the MEDICATION table to the PUBLIC.

```
GRANT ALL PRIVILEGES
ON MEDICATION
TO PUBLIC;

Grant succeeded.
```

USER_A's table privileges granted as recorded in the System Catalog (PUBLIC's privileges have been added)

GRANTEE	OWNER	TABLE_NAME	GRANTOR	PRIVILEGE	GRANTABLE
PUBLIC	**USER_A**	**MEDICATION**	**USER_A**	**DELETE**	**NO**
PUBLIC	**USER_A**	**MEDICATION**	**USER_A**	**INSERT**	**NO**
PUBLIC	**USER_A**	**MEDICATION**	**USER_A**	**SELECT**	**NO**
PUBLIC	**USER_A**	**MEDICATION**	**USER_A**	**UPDATE**	**NO**
PUBLIC	**USER_A**	**MEDICATION**	**USER_A**	**REFERENCES**	**NO**
USER_B	USER_A	ORDERS	USER_A	DELETE	NO
USER_B	USER_A	ORDERS	USER_A	INSERT	NO
USER_B	USER_A	ORDERS	USER_A	SELECT	NO
USER_B	USER_A	ORDERS	USER_A	UPDATE	NO
USER_C	USER_A	PATIENT	USER_A	INSERT	NO
USER_C	USER_A	PATIENT	USER_A	SELECT	NO
USER_D	USER_A	MEDICATION	USER_A	DELETE	NO
USER_D	USER_A	MEDICATION	USER_A	INSERT	NO

Twice for User D

Example 3

- USER_A has not granted USER_B any privileges on the PATIENT table. The following GRANT statement allows USER_B to retrieve (i.e., select) rows from USER_A's PATIENT table and also grant the SELECT privilege to other users.

```
GRANT SELECT
ON PATIENT
TO USER_B
WITH GRANT OPTION;

Grant succeeded.
```

Example 4

- At this point assume USER_B is connected to the database and attempts to grant the SELECT privilege received from USER_A to USER_D.

```
GRANT SELECT
ON PATIENT
TO USER_D;

ERROR at line 2:  table PATIENT does not exist

GRANT SELECT
ON USER_A.PATIENT
TO USER_D;

Grant succeeded.

USER_B's table privileges granted as recorded in the System Catalog
GRANTEE    OWNER      TABLE_NAME GRANTOR    PRIVILEGE            GRANTABLE
---------- ---------- ---------- ---------- -------------------- -------
USER_D     USER_A     PATIENT    USER_B     SELECT               NO
```

Example 4 (continued)

USER_A's table privileges granted as recorded in the System Catalog

GRANTEE	OWNER	TABLE_NAME	GRANTOR	PRIVILEGE	GRANTABLE
PUBLIC	USER_A	MEDICATION	USER_A	DELETE	NO
PUBLIC	USER_A	MEDICATION	USER_A	INSERT	NO
PUBLIC	USER_A	MEDICATION	USER_A	SELECT	NO
PUBLIC	USER_A	MEDICATION	USER_A	UPDATE	NO
PUBLIC	USER_A	MEDICATION	USER_A	REFERENCES	NO
USER_B	**USER_A**	**PATIENT**	**USER_A**	**SELECT**	**YES**
USER_B	USER_A	ORDERS	USER_A	DELETE	NO
USER_B	USER_A	ORDERS	USER_A	INSERT	NO
USER_B	USER_A	ORDERS	USER_A	SELECT	NO
USER_B	USER_A	ORDERS	USER_A	UPDATE	NO
USER_C	USER_A	PATIENT	USER_A	INSERT	NO
USER_C	USER_A	PATIENT	USER_A	SELECT	NO
USER_D	**USER_A**	**PATIENT**	**USER_B**	**SELECT**	**NO**
USER_D	USER_A	MEDICATION	USER_A	DELETE	NO
USER_D	USER_A	MEDICATION	USER_A	INSERT	NO

Example 5

- Privileges can be granted on specific columns of a table as well as on all columns. Here, USER_A grants USER_E the UPDATE privilege on the three columns of the PATIENT table.

Example 5 (continued)

```
GRANT UPDATE (PAT_WING, PAT_ROOM#, PAT_BED)
ON PATIENT
TO USER_E;

Grant succeeded.
```

USER_A's table privileges granted as recorded in the System Catalog

GRANTEE	OWNER	TABLE_NAME	GRANTOR	PRIVILEGE	GRANTABLE
PUBLIC	USER_A	MEDICATION	USER_A	DELETE	NO
PUBLIC	USER_A	MEDICATION	USER_A	INSERT	NO
PUBLIC	USER_A	MEDICATION	USER_A	SELECT	NO
PUBLIC	USER_A	MEDICATION	USER_A	UPDATE	NO
PUBLIC	USER_A	MEDICATION	USER_A	REFERENCES	NO
USER_B	USER_A	PATIENT	USER_A	SELECT	YES
USER_B	USER_A	ORDERS	USER_A	DELETE	NO
USER_B	USER_A	ORDERS	USER_A	INSERT	NO
USER_B	USER_A	ORDERS	USER_A	SELECT	NO
USER_B	USER_A	ORDERS	USER_A	UPDATE	NO
USER_C	USER_A	PATIENT	USER_A	INSERT	NO
USER_C	USER_A	PATIENT	USER_A	SELECT	NO
USER_D	USER_A	PATIENT	USER_B	SELECT	NO
USER_D	USER_A	MEDICATION	USER_A	DELETE	NO
USER_D	USER_A	MEDICATION	USER_A	INSERT	NO

Example 5 (continued)

USER_A's column privileges granted as recorded in the System Catalog

GRANTEE	OWNER	TABLE_NAME	COLUMN_NAME	GRANTOR	PRIVILEGE	GRANTABLE
USER_E	USER_A	PATIENT	PAT_WING	USER_A	UPDATE	NO
USER_E	USER_A	PATIENT	PAT_ROOM#	USER_A	UPDATE	NO
USER_E	USER_A	PATIENT	PAT_BED	USER_A	UPDATE	NO

Example 6

- USER_A is granting the UPDATE privilege on all columns of the patient table to USER_F.

Example 6 (continued)

```
GRANT UPDATE
ON PATIENT
TO USER_F;

Grant succeeded.
```

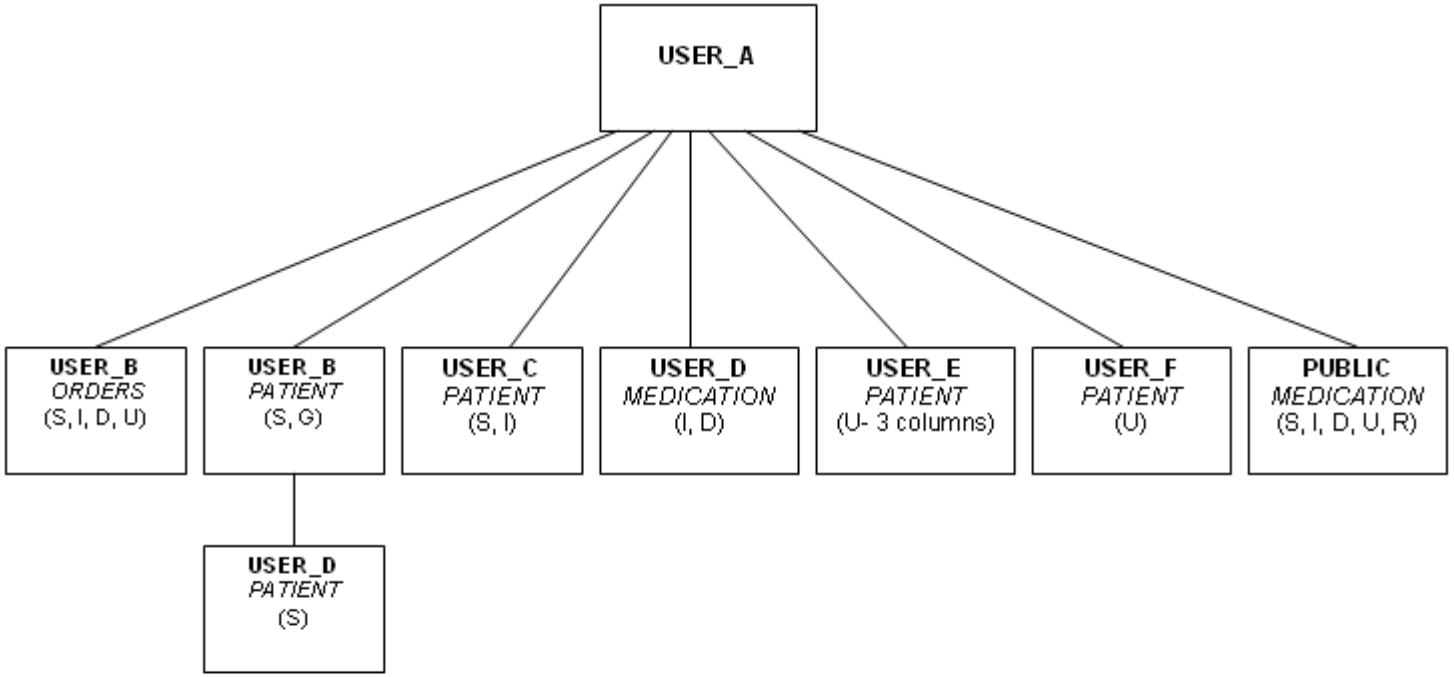
USER_A's table privileges granted as recorded in the System Catalog (note addition of USER_F's UPDATE privilege)

GRANTEE	OWNER	TABLE_NAME	GRANTOR	PRIVILEGE	GRANTABLE
PUBLIC	USER_A	MEDICATION	USER_A	DELETE	NO
PUBLIC	USER_A	MEDICATION	USER_A	INSERT	NO
PUBLIC	USER_A	MEDICATION	USER_A	SELECT	NO
PUBLIC	USER_A	MEDICATION	USER_A	UPDATE	NO
PUBLIC	USER_A	MEDICATION	USER_A	REFERENCES	NO
USER_B	USER_A	PATIENT	USER_A	SELECT	YES
USER_B	USER_A	ORDERS	USER_A	DELETE	NO
USER_B	USER_A	ORDERS	USER_A	INSERT	NO
USER_B	USER_A	ORDERS	USER_A	SELECT	NO
USER_B	USER_A	ORDERS	USER_A	UPDATE	NO
USER_C	USER_A	PATIENT	USER_A	INSERT	NO
USER_C	USER_A	PATIENT	USER_A	SELECT	NO
USER_D	USER_A	PATIENT	USER_B	SELECT	NO
USER_D	USER_A	MEDICATION	USER_A	DELETE	NO
USER_D	USER_A	MEDICATION	USER_A	INSERT	NO
USER_F	**USER_A**	**PATIENT**	**USER_A**	**UPDATE**	**NO**

USER_F's table privileges received as recorded in the System Catalog

OWNER	TABLE_NAME	GRANTOR	PRIVILEGE	GRANTABLE
USER_A	PATIENT	USER_A	UPDATE	NO

Privileges Granted by User_A and USER_B



S – SELECT Privilege; **I** – INSERT Privilege; **D** – DELETE Privilege; **U** – UPDATE Privilege; **R** – REFERENCES Privilege; **G** – GRANT Option

Figure 11.5
Privileges Granted By USER_A and USER_B