03. Data Definition using SQL

[PM Jat, DAIICT, Gandhinagar]

Data Definition Language in SQL

What do we mean by data definition?

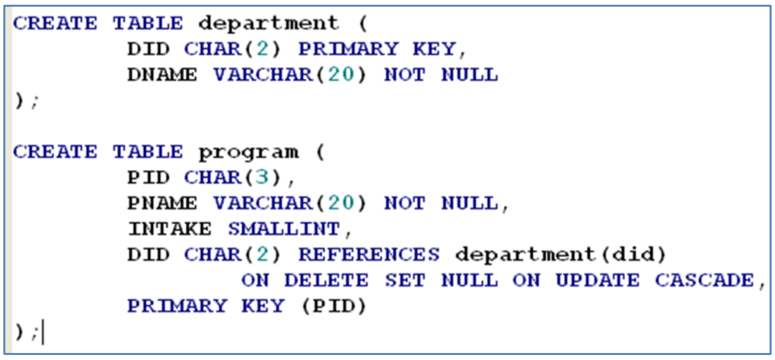
It is process of defining database schema. In RDBMS, it is done using DDL part of SQL. A data definition program is often referred as DDL script. A DDL script typically contains instructions for creating new database schema.

A data definition task has DDL script as input to DBMS, and on successful run of DDL script on DBMS, following things are done -

- New database instance (empty) is created as per the schema definition given in script.
- Database schema information is saved as data in database catalogue.

SQL provides a set of commands for this purpose and are referred as DDL part of SQL.

Given below is DDL script for XIT database.



```
CREATE TABLE department (
        DID CHAR(2) PRIMARY KEY,
        DNAME VARCHAR(20) NOT NULL
);

CREATE TABLE program (
        PID CHAR(3),
        PNAME VARCHAR(20) NOT NULL,
        INTAKE SMALLINT,
        DID CHAR(2) REFERENCES department(did)
                ON DELETE SET NULL ON UPDATE CASCADE,
        PRIMARY KEY (PID)
);
```

SQL-DDL commands

- This set of commands are used to *define* and *maintain* database schemas
- Following are commonly used commands -
  - `CREATE SCHEMA, DROP SCHEMA`
  - `CREATE TABLE, ALTER TABLE, DROP TABLE`
  - `CREATE DOMAIN/TYPE, DROP DOMAIN/TYPE`
  - `CREATE VIEWS, DROP VIEWS`
- Note that SQL is not case sensitive language.

CREATE/DROP SCHEMA

Database management systems often have two containers to store databases.

(1) Database, and
(2) Second is "Schema"

Remember both of these are container to hold databases.

Caution: DO NOT GET CONFUSED with terms!

Meaning of these two things as containers is different than terms that we have learned so far "database" and "schema".

What we have learned is "database instance", that is data and database schema that is database definition.

DBMS typically use these two names as "containers" and a particular DBMS often have specific meaning of these two.

For instance, PostgreSQL does as following:

- Database is owned by use database user.
- A database (container) can have n number of databases. Each database shall have its schema and instance.
- Schema (container) holds everything of a database. Its tables, constraints, indexes. views, indexes, stored functions, etc.
- You as a user of database, typically own a single database. You can though own multiple databases.
- You create multiple schema within a database. For each database, you create a separate schema.

Here is how you create a schema for XIT database.

```
CREATE SCHEMA XIT;
```

You can drop an existing schema, as following:

```
DROP SCHEMA XIT;
```

Command for dropping a schema would fail if the being dropped is schema non-empty. Use CASCADE option to convey your firm intention of "DELETION" as following

```
DROP SCHEMA XIT CASCADE;
```

CASCADE is used with most DROP commands to given deletion affirmation. However, CASCADE option should be used judiciously.

CREATE TABLE statement

`CREATE TABLE` is most comprehensive command for creating relations. This command is used to specify a new relation; command typically requires specifying following parameters

- Name of relation
- Its attributes, domain for attributes, and
- Constraints.

This command does following things:

- Creates empty relation in database instance.
- Makes appropriate entries in database catalogue; that is schema definition is put into database catalogue

CREATE TABLE – General Form

```
CREATE TABLE R (A1 dom(A1), A2 dom(A2), ..., An dom(An),
                  (integrity-constraint1),
                  ...,
                  (integrity-constraintk))
```

– *R* is the name of the relation

– each A_i is an attribute name in the schema of relation *r*

– dom(A_i) is the data type(domain) for attribute A_i

note that in SQL-DDL domains are specified by SQL data types

Example CREATE TABLE

```
CREATE TABLE department (
    DID CHAR(2) PRIMARY KEY,
    DNAME VARCHAR(30) NOT NULL
);

CREATE TABLE program (
    PID CHAR(3),
    PNAME VARCHAR(20) NOT NULL,
    INTAKE SMALLINT,
    DID CHAR(2) REFERENCES department(did),
    PRIMARY KEY (PID)
);

CREATE TABLE student (
    StudID CHAR(3),
    Name VARCHAR(20) NOT NULL,
    ProgID CHAR(3) REFERENCES program(pid),
    "class" smallint,
    cpi decimal(4,2)
);
```

Note: "`class`" is reserved word is PostgreSQL. Reserved words can be used for attribute names (or for any object for that matter) but need to be put in double quotes, (" "), for example relation student has an attribute named "class".

Also note that all names here table, attribute etc. are case insensitive. However, they are case sensitive when put in quotes like the case above; "class" here is not same as "Class", where as StudID is same as studid; student and STUDENT are same, like wise.

Default Value [for an attribute]

You can specify default value for an attribute; as shown below

```
CREATE TABLE EMPLOYEE (
    …
    DNO SMALLINT DEFAULT 4,
    …
)
```

This value is assigned to the attribute in a new tuple, when we do not specify its value.

Data Types in SQL

DDL uses SQL data types for specifying domain of attributes. Here is broad category of PostgreSQL data types-

- Numeric
- Character strings
- Boolean
- DATE/TIME
- Binary

Numeric Data types

Whole Numbers:

- `INTEGER`, `SMALLINT`, `BIGINT` – though implementation dependent, are typically 4 byte, 2 byte, 8 byte integers respectively.

Approximate (Floating point) Numbers:

- `REAL`, `DOUBLE PRECISION` – typically 4 byte and 8 byte floating numbers with precision of 6 and 15 digits precision, respectively
- `FLOAT(p)`, lets you specify precision in terms of “bits - binary digits”; it can be anything between 1 to 53; specifying float with no p is taken as double precision.
- Values are rounded off if have higher precision; Value might overflow and under flow.
- Special values associated with: Infinity, - Infinity, NaN (Not-a-Number)

Exact fractional numbers

- Given inaccuracies with floating point numbers, SQL provides an alternative data type `Numeric`, where accuracy is important for example in case of Amount, and other quantities or so. You would not want Rs 2000 to be shown up as 1999.9999.
- Numeric data types can store numbers with up to 1000 digits of precision and perform calculations exactly.
- However, arithmetic on numeric values is slow compared to the integer types, and floating-point types.
- Numeric are declared as `NUMERIC(i, j)`, or `Numeric(i)`; where i is precision, and j is scale (default scale is 0)
- Another keyword Decimal is also used for numeric data type declarations: `DECIMAL(i,j)`, `DEC(i,j)`, where meaning of (i,j) remain same.
- Example: value 23.2134 has precision 6 and scale 4. If no scale is specified, then it is taken as zero

Character Strings

PostgreSQL provides following three-character data types. First two types require specifying the length of string (in terms of number of characters) while third one does not.

Name	**Description**
`CHARACTER VARYING(N)`, `VARCHAR(N)`	variable-length with limit
`CHARACTER(N), CHAR(N)`	fixed-length, blank padded
`TEXT`	variable unlimited length

First two types can store strings up to n characters in length. An attempt to store a longer string into a column of these types will result in an error, unless excess characters are all spaces, in which case the string will be truncated to the maximum length.

Fixed length characters are typically used for IDs/Codes. Extra spaces are padded towards the end if actual data length is less than specified length.

While character and character varying have maximum length limit as 1GB (take note; this is storage size and not length of string); text data type allows storing strings of unlimited.

Notable remarks from PostgreSQL documentation:

> There is no performance difference among these three types, apart from increased storage space when using the blank-padded type, and a few extra CPU cycles to check the length when storing into a length-constrained column. While character(n) has performance advantages in some other database systems, there is no such advantage in PostgreSQL; in fact, `CHARACTER(N)` is usually the slowest of the three because of its additional storage costs. In most situations text or character varying should be used instead.

Some RDBMS, like oracle have a data type called Character Large Objects (CLOB) having limit of 4GB. [Oracle has limits of 2000 bytes and 4000 bytes on character and character varying respectively]

Boolean

Valid values for Boolean type are

- True: `TRUE`, `'TRUE'`
- False: `FALSE`, `'FALSE'`

Values are not case sensitive

In some implementations you may also find ‘y’, ‘yes’, 1 are accepted for true and ‘n’, ‘no’, and 0 for false. PostgreSQL also allows this.

Binary data types in SQL

Images, Doc, Excel, PDFs, etc are typically stored as binary data. PostgrSQL provides following binary data types:

`BIT(N)`: fixed length of N bits

`BIT VARYING(n)`: variable length bits of max n

`BYTEA`: to store large binary objects like images and video etc.; this is equivalent to Binary Large Objects (BLOB) in some implementations like oracle.

DATE data types

- Stores year, month, and day
- Literals are typically specified as DATE 'YYYY-MM-DD' when you communicate with database. For example, `DATE'1989-05-13'`

  Note: This is casting syntax as well in ANSI SQL; in example above string data is gets casted to date type. General syntax is: `<data-type> data_in_other_type;`

TIME data types

- Stores hours, minutes, and seconds

- Literals are specified as TIME 'HH:MI:SS', seconds may contain fractional values
- For example- `TIME '15:25:18.34'`

TIMESTAMP data types

- Stores Y,M,D,H, Min, Second
- Literals are specified as
  `TIMESTAMP 'YYYY-MM-DD HH:MI:SS';`
  for example: `TIMESTAMP '2004-10-19 10:23:54'`
- Time and Timestamps can be specified time zone info too, in the form of +13:00 to –12:59 in the format of HH:MI; for example: `TIMESTAMP '2004-10-19 10:23:54+02'`

Specifying Constraints in SQL-DDL

```
NOT NULL

UNIQUE

PRIMARY KEY

FOREIGN KEY

CHECK
```

NOT NULL constraint

Attribute cannot be NULL

UNIQUE constraint

No two tuple can have same value for UNIQUE constrained attribute. Unique constraint can be specified for a group of attributes too.

PRIMARY KEY

Technically, a primary key constraint is simply a combination of a unique constraint and a not-null constraint. So, the following two table definitions accept the same data:

Primary Key constraint may also influence the way data on the disk organized. PostgreSQL automatically creates b+-tree index on Primary Key attributes of a relation.

Normally attribute level constraints (involves only one attribute) are specified inline against the attribute definition, while

Constraints involving multiple attributes must be specified as separate fragment (separated by comma) in CREATE table statement.

Attribute level constraints are in-lined with attribute declaration

Example below:

```
CREATE TABLE PROGRAM (
      PNAME VARCHAR(20) NOT NULL,
      PID SMALLINT PRIMARY KEY,
      DID CHAR(2) NOT NULL REFERENCES DEPARTMENT(DID),
      UNIQUE (PNAME)
);
```

Constraints could be specified as separate fragment (normally done when involves multiple attributes). Example below shows the same.

```
CREATE TABLE PROGRAM (
      PNAME VARCHAR(20) NOT NULL,
      PID SMALLINT,
      DID CHAR(2) NOT NULL,
      UNIQUE (PNAME),
      PRIMARY KEY (PID),
      FOREIGN KEY (DID) REFERENCES DEPARTMENT(DID),
);
```

CHECK constraints:

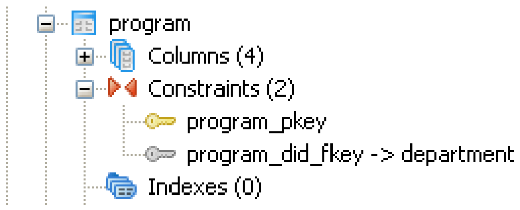
Two example below should be self-explanatory and conveying the intuition.

```
CREATE TABLE STUDENT (
      …
      CPI REAL CHECK (cpi >=0 AND cpi <= 10)
      …
);
```

```
CREATE TABLE COURSE_OFFERING (
      …
      semester varchar(6) CHECK(semester in ('Autumn','Winter',
'Summer')),
      …
);
```

Constraints have name in schema

Below is a snap from pgAdmin accessing the "XIT" schema



You may not always be giving names to constraints. PostgreSQL automatically gives names and follows certain conventions while giving names. In constraints shown in above snapshot, you should able to observe following patterns-

PK: <rel-name>_pkey, for example- program_pkey

FK: <rel-name>_<fk-attr-name>_fkey; for example- program_did_fkey

You can give names to constraints explicitly, and should be done for complex constraints, so that you can refer them for dropping or altering purposes

```
CREATE TABLE  works_on (
      essn  CHAR(9),
      ...
      PRIMARY KEY  (essn, pno),
      CONSTRAINT work_proj_check CHECK  ( <predicate> )
)
```

Constraints can be added later too using ALTER TABLE command; example follows.

Create Domain

```
CREATE DOMAIN CPI_TYPE AS REAL CHECK
      (value >= 0 AND value <= 10);
```

This in turn can be used as data type for cpi in student table

```
CREATE TABLE student(
      sid char(9) PRIMARY KEY,
      name varchar(30),
      cpi CPI_TYPE
);
```

DROP TABLE

```
DROP TABLE EMPLOYEE;
```

Drop table not only deletes table rows, but also deletes its definition from the catalog.

Drop table is rejected if table being dropped is referenced by some another relation. To forcefully drop such table using CASCADE option

```
DROP TABLE EMPLOYEE CASCADE;
```

ALTER TABLE

```
ALTER TABLE EMPLOYEE ADD COLUMN JOB VARCHAR(12);
```

```
ALTER TABLE EMPLOYEE DROP COLUMN Address;
```

If column being deleted is part of some constraints, dropping of column would be rejected; use the cascade option to force the drop, in that case all constraints using the column will also be dropped.

```
ALTER TABLE EMPLOYEE DROP COLUMN Address CASCADE;
```

Drop default value for an attribute

```
ALTER TABLE STUDENT ALTER COLUMN INTAKE DROP DEFAULT;
```

Set Default for a column

```
ALTER TABLE PROGRAM ALTER COLUMN INTAKE SET DEFAULT 60;
```

Add a constraint

```
ALTER TABLE STUDENT
    ADD CONSTRAINT VALID_CPI_RANGE CHECK (cpi >=0 and cpi <= 10);
```

Drop a constraint

```
ALTER TABLE EMPLOYEE DROP CONSTRAINT VALID_CPI_RANGE;
```

General form of alter table

```
ALTER TABLE <table-name> <action>
```

Where action can be-

```
ADD COLUMN
ALTER COLUMN
DROP COLUMN
ADD CONSTRAINT
DROP CONSTRAINT
```

Referential Actions

- Consider following relation instances "PROGRAM" and "DEPARTMENT". What if we allow deleting tuple with DID='EE' from Department relation?
  Does it have any cause of concern?

Department

DID	DName
CS	Computer Engineering
EE	Electrical Engineering
ME	Mechanical Engineering

Program

ProgID	ProgName	Intake	DID
BCS	BTech(CS)	40	CS
BIT	BTech(IT)	30	CS
BEE	BTech(EE)	40	EE
BME	BTech(ME)	40	ME

- What if we Electrical Engineering department becomes "Electrical and Electronics Engineering", and its code is changed to 'EC'?
- There are some concerns
  (1) when a tuple referred by some FK value is deleted, or
  (2) When a value referred by some FK value is modified.
- Should such operation be allowed? If this is permitted, the database will be inconsistent?
- We can specify appropriate course of action to deal with such a situation. We use ON UPDATE … ON DELETE clause for this purpose. Following is general syntax of this command:

```
ON UPDATE <action> ON DELETE <action>
```

  Here basically we specify the action that is to be applied on referencing FK value, when UPDATE or DELETE operation is applied on referred value.

  `ON UPDATE ACTION` specifies the action that is applied on referencing FK value when referred value get changed? `ON DELETE ACTION` specifies the action that is taken when the referred tuple is getting deleted!

Following are the action that are specified:

- `SET NULL` – sets any referencing FK to null.
  - `ON DELETE SET NULL` implies that FK value referring to deleted tuple becomes NULL
  - `ON UPDATE SET NULL` implies that if referred value get changed, make the referring value to NULL.
- `SET DEFAULT` - sets any referencing FK to default value

- `CASCADE` –
    - `ON DELETE CASCADE` shall delete all tuples that are referred by deleted tuple. For example, if tuple with DID EE gets deleted in department relation, then all tuples in having EE value in did attribute of program relation shall also be deleted.
    - `ON UPDATE CASCADE` if referred value changes, change all update new value at all references too. In same example if EE becomes EC in department relation, all occurrences EE in program relation shall also become EC.
- `NO ACTION`- rejects any update that cause referential integrity violation
- `RESTRICT`- same as NO ACTION with the additional restriction that the integrity check cannot be deferred. Note that constraint check can be deferred to the point of transaction commit.

Summary SQL-DDL

```
CREATE TABLE

ALTER TABLE

CREATE DOMAIN

DROP ...
```

Add/Drop Constraints

```
NOT NULL

UNIQUE

PRIMARY KEY

FOREIGN KEY

CHECK
```

Giving Name to constraints

You can create a table from existing table (or from a result of any SQL query), as following-

```
CREATE TABLE emp_tmp_dno5 AS
        SELECT * FROM employee WHERE dno = 5;
```