DDL SQL Statements

CE384: Database Design

Maryam Ramezani
Sharif University of Technology
maryam.ramezani@sharif.edu

Data Definition Language (DDL)

- This is a language that allows the DBA or user to describe and name the entities, attributes and relationships required for the application, together with any associated integrity and security constraints (Begg & Connolly, 2002)

- DDL is not only used to specify new database schemas but also to modify exiting ones.

DDL

- Relation DB schema objects are created and maintained by using SQL DDL statements (such as CREATE, ALTER, DROP).

- The result of compiling DDL statements is a set of tables stored in special files collectively called the system catalog. The system catalog may also be referred to as a data dictionary.

- Example of DDL SQL statement:
  - CREATE
  - ALTER
  - DROP
  - RENAME
  - TRUNCATE
  - COMMENT

01

Create

Creating a Table

- Involves using the CREATE TABLE statement which includes:
  - Specifying a table name [mandatory]
  - Defining columns of the table [mandatory]
  - Column name [mandatory]
  - Column data type [mandatory]
  - Column constraints [optional]
  - Providing a default value [optional]
  - Specifying table constraints [optional]

Creating a Table

- Creates a table with one or more columns of the specified dataType.
  - Syntax:

```
CREATE TABLE TableName {(
{colName dataType [NOT NULL] [UNIQUE]
[DEFAULT defaultOption]
[CHECK searchCondition] [,…]}
[PRIMARY KEY (listOfColumns),]
[UNIQUE (listOfColumns),] […,]
[FOREIGN KEY (listOfFKColumns)
  REFERENCES ParentTableName [(listOfCKColumns)],
  [ON UPDATE referentialAction]
  [ON DELETE referentialAction ]] [,…]
[CHECK (searchCondition)] [,…] )}
```

DEFAULT option

-
  ```
  CREATE TABLE students
  (regNo varchar(15), name varchar(20), dob date, gender
  char(1) default ‘M’);
  ```

- Example

  ```
  CREATE TABLE students (regNo varchar(15), name varchar(20), dob date,
  gender char(1) default ‘F');
  select * from students
  insert into students values ('12','Ali',now())
  insert into students values ('120','Leili')
  insert into students values (9)
  insert into students values ('98','23','2024-04-02')
  insert into students values ('98','23','2023-02-13','Alaki')
  insert into students values ('98','23','2023-02-13','A')
  ```

Integrity Enhancement Feature

- Consider these types of integrity constraints defined in CREATE & ALTER (We will read it next part):
  1. Required data
  2. Domain constraints
  3. Entity integrity
  4. Referential integrity
  5. Inherits
- Imposed in order to protect the database from becoming inconsistent.

Integrity Enhancement Feature

Required Data

- Null is distinct from blank or zero
  - Zero is a number. Null means "no value". Blank could also be an empty string. It depends on the context.
- Syntax:

```
columnName    dataType    [NOT NULL | NULL]
```

- Example:

```
position        VARCHAR(10)     NOT NULL
```

Integrity Enhancement Feature

Domain Constraints

Every column has a domain, in other words a set of legal values. **“CHECK”**

First Syntax:

CHECK (search condition)

Example:

- sex CHAR NOT NULL CHECK (sex IN (‘M’, ‘F’))
- salary DECIMAL NOT NULL CHECK (salary > 10000);
- bno INT CHECK (bno IN(SELECT branchno FROM branch))

Integrity Enhancement Feature

Domain Constraints

- By default, PostgreSQL assigns a name to a CHECK constraint using the following format:

  {table}_{column}_check

- How to give name to the CHECK constraint?

```
CREATE TABLE Test_Check (name char(5) ,regNo serial, constraint
sharif_check check(name in ('DB') ))
```

Integrity Enhancement Feature

Domain Constraints

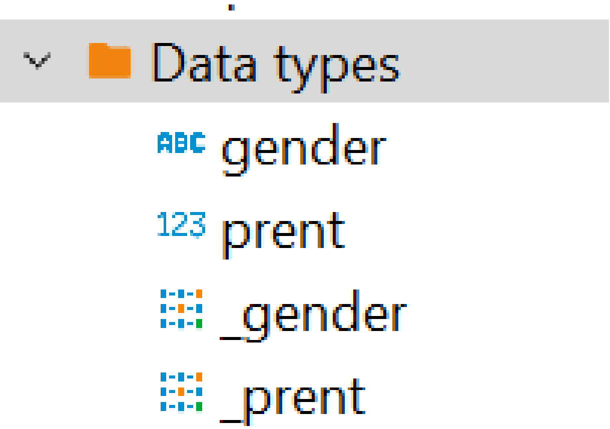
CREATE DOMAIN

- **Second Syntax:**

```
CREATE DOMAIN DomainName [AS] dataType
[DEFAULT defaultOption]
[CHECK (searchCondition)]
```

- **Example:**

  -
    ```
    CREATE DOMAIN Gender AS CHAR
               CHECK (VALUE IN ('M', 'F'));
    ```



Data types
- gender
- prent
- _gender
- _prent

Integrity Enhancement Feature

Domain Constraints

- <u>**Search Condition can involve a table lookup:**</u>

```
CREATE DOMAIN BranchNo AS CHAR(4)
CHECK (VALUE IN (SELECT branchNo FROM Branch));
```

Integrity Enhancement Feature

Entity Integrity

- Primary key of a table must contain a unique, non-null value for each row.
- Syntax:

```
PRIMARY KEY(staffNo)
```

- Example:

```
PRIMARY KEY(clientNo, propertyNo)
```

- Can only have one PRIMARY KEY clause per table.
- Can still ensure uniqueness for alternate keys using UNIQUE:

```
    (1)UNIQUE(telNo)
    (2)pno  VARCHAR(5) NOT NULL UNIQUE;
    (3)CONSTRAINT pno_check UNIQUE(pno));
```

Integrity Enhancement Feature

Referential Integrity

- FK is column or set of columns that links each row in child table containing foreign FK to row of parent table containing matching PK.

- Referential integrity means that, if FK contains a value, that value must refer to existing row in parent table.

- ISO standard supports definition of FKs with FOREIGN KEY clause in CREATE and ALTER TABLE:

- Syntax:

```
(1)  FOREIGN KEY (FK column (,…)) REFERENCES  table_name [(CK        column (,…))]
(2) (FK column (,…)) REFERENCES  table_name
```

- Example:

```
FOREIGN KEY(bNo) REFERENCES Branch (branchNo)
AGENT_CODE CHAR(6) NOT NULL REFERENCES AGENTS
```

Integrity Enhancement Feature

Referential Integrity

- Any INSERT/UPDATE attempting to create FK value in child table without matching the value in parent is rejected.
- Action taken attempting to update/delete a reference value in parent table with matching rows in child is dependent on referential action specified using ON UPDATE and ON DELETE subclauses:
  - CASCADE: Delete row from parent and delete matching rows in child, and so on in cascading manner.
  - SET NULL: Delete row from parent and set FK column(s) in child to NULL. Only valid if FK columns is NULL.
  - SET DEFAULT: Delete row from parent and set each component of FK in child to specified default. Only valid if DEFAULT specified for FK columns.
  - NO ACTION: Reject delete from parent. Default.
  - RESTRICT: Reject update or delete from parent if matching rows exist in child. This is essentially the same as NO ACTION, but with RESTRICT the check is done immediately, while NO ACTION checks are deferred until the end of the transaction.

Integrity Enhancement Feature

Referential Integrity

- Example

```
 FOREIGN KEY (staffNo) REFERENCES Staff ON DELETE SET NULL
FOREIGN KEY (ownerNo) REFERENCES Owner ON UPDATE CASCADE
```

Cascading Actions in SQL

- If there is a chain of foreign-key dependencies across multiple relations, with **on delete cascade** specified for each dependency, a deletion or update at one end of the chain can propagate across the entire chain.
- If a cascading update to delete causes a constraint violation that cannot be handled by a further cascading operation, the system aborts the transaction. As a result, all the changes caused by the transaction and its cascading actions are undone.
- Referential integrity is only checked at the end of a transaction
  - Intermediate steps are allowed to violate referential integrity provided later steps remove the violation
  - Otherwise it would be impossible to create some database states, e.g. insert two tuples whose foreign keys point to each other (e.g. *spouse* attribute of relation *marriedperson)*

Integrity Enhancement Feature

Inherits

- Subclass is defined as a table inheriting attributes from the parent table and adding some new attibutes.

- Syntax:

```
 Create table subclass () INHERITS (superclass table)
```

- Example:

```
CREATE TABLE person(
    pid      int,
    name     text,
    address         text
  );
  CREATE TABLE student(
    major_subject text,
    study_points  int
  ) INHERITS (person);
```

Non-Atomic Values

- One of the tenets of the relational model is that the attributes of a relation are atomic
  - I.e. only a single value for a given row and column
- Postgres does not have this restriction: attributes can themselves contain sub-values that can be accessed from the query language
  - Examples include arrays and other complex data types.

Non-Atomic Values - Arrays

- Postgres allows attributes of an instance to be defined as fixed-length or variable-length multi-dimensional arrays. Arrays of any base type or user-defined type can be created. To illustrate their use, we first create a table with arrays of base types.

```
CREATE TABLE SAL_EMP (
    name                text,
    pay_by_quarter      int4[ ],
    schedule            text[ ][ ]
);
```

Non-Atomic Values - Arrays

- The preceding SQL command will create a table named SAL_EMP with a text string (name), a one-dimensional array of int4 (pay_by_quarter), which represents the employee's salary by quarter and a two-dimensional array of text (schedule), which represents the employee's weekly schedule.

- Now we do some INSERTSs; note that when appending to an array, we enclose the values within braces and separate them by commas.

Inserting into Arrays

```
INSERT INTO SAL_EMP
    VALUES ('Bill',
    '{10000, 10000, 10000, 10000}',
    '{{"meeting", "lunch"}, {"", ""}}');

INSERT INTO SAL_EMP
    VALUES ('Carol',
    '{20000, 25000, 25000, 25000}',
    '{{"talk", "consult"}, {"meeting", ""}}');
```

Querying Arrays

- This query retrieves the names of the employees whose pay changed in the second quarter:

```
SELECT name
    FROM SAL_EMP
    WHERE SAL_EMP.pay_by_quarter[1] <>
    SAL_EMP.pay_by_quarter[2];

+-------+
|name   |
+-------+
|Carol  |
+-------+
```

Querying Arrays

- This query retrieves the third quarter pay of all employees:

```
SELECT SAL_EMP.pay_by_quarter[3] FROM SAL_EMP;

+----------------+
|pay_by_quarter  |
+----------------+
|10000           |
+----------------+
|25000           |
+----------------+
```

Querying Arrays

- We can also access arbitrary slices of an array, or subarrays. This query retrieves the first item on Bill's schedule for the first two days of the week.

```
SELECT SAL_EMP.schedule[1:2][1:1]
    FROM SAL_EMP
    WHERE SAL_EMP.name = 'Bill';

+-------------------+
|schedule           |
+-------------------+
|{{"meeting"},{""}} |
+-------------------+
```

Creating a Table by Using a Subquery Syntax

- Create a table and insert rows by combining the CREATE TABLE statement and the AS *subquery* option.

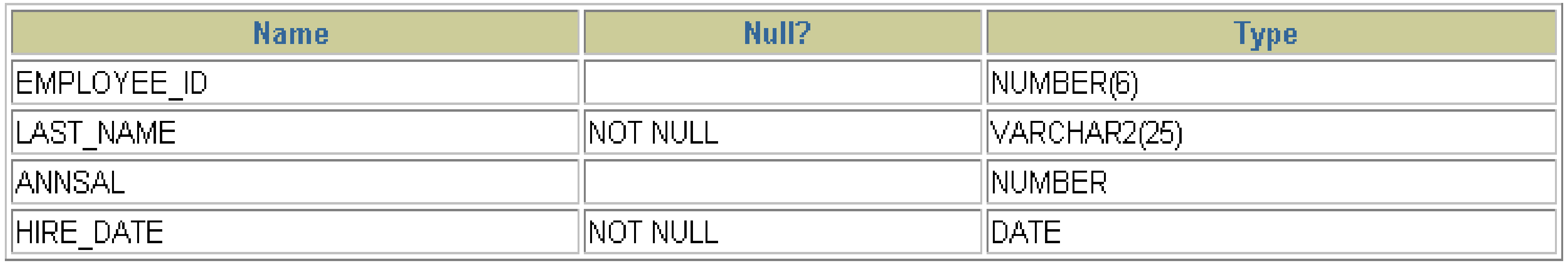
```
CREATE TABLE table
        [(column, column...)]
AS subquery;
```

- Match the number of specified columns to the number of subquery columns.
- Define columns with column names and default values.
- The table is created with the specified column names, and the rows retrieved by the SELECT statement are inserted into the table.
- The column definition can contain only the column name and default value.
- If column specifications are given, the number of columns must equal the number of columns in the subquery SELECT list.
- If no column specifications are given, the column names of the table are the same as the column names in the subquery.
- The integrity rules are not passed onto the new table, only the column data type definitions.

Creating a Table by Using a Subquery

```
CREATE TABLE  dept80
  AS
    SELECT   employee_id, last_name,
             salary*12 ANNSAL,
             hire_date
    FROM     employees
    WHERE    department_id = 80;
```

```
Table created.
```



Name	Null?	Type
EMPLOYEE_ID		NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
ANNSAL		NUMBER
HIRE_DATE	NOT NULL	DATE

What is this? ANNASAL is the alias for column after multiplying. Without the alias, , this error is generated:

```
ERROR at line
3: ORA-00998:
must name this
expression with
a column alias
```

Temporary Table

- A temporary table is a base table that is not stored in the database, but instead exists only while the database session in which it was created is active.
  - Direct create same as create table but with this syntax:

    ```
    CREATE TEMPORARY TABLE temp_people (NID int, name text)
    ```

  - Create from another existing table using subquery.

    ```
    CREATE TEMPORARY TABLE temp_people as select * from people
    ```

03

Alter

Alter Table

- Add a new column to a table.
- Drop a column from a table.
- Add a new table constraint.
- Drop a table constraint.
- Set a default for a column.
- Drop a default for a column.
- Modify an existing column

```
table           is the name of the table
ADD|MODIFY|DROP    is the type of modification
column                is the name of the new column
datatype        is the data type and length of the new column
DEFAULT expr          specifies the default value for a new column
```

Example

- Change Staff table by removing default of ‘Assistant’ for position column and setting default for sex column to female (‘F’).

  -
    ```
    ALTER TABLE Staff
    ALTER position DROP DEFAULT;
    ```

  -
    ```
    ALTER TABLE Staff
    ALTER sex SET DEFAULT ‘F’;
    ```

Adding a new column

```
ALTER TABLE dept80 ADD (job_id VARCHAR(9))
```

- You cannot specify where the column is to appear. The new column becomes the last column.
- If a table already contains rows when a column is added, then the new column is initially null for all the rows.

Modifying a Column

```
ALTER TABLE dept80 MODIFY     (last_name VARCHAR2(30))
```

- You can change a column’s data type, size, and default value.
- You can increase the width or precision of a numeric column.
- You can increase the width of numeric or character columns.
- You can decrease the width of a column **only if** the column contains only null values or if the table has no rows.
- You can change the data type **only if** the column contains null values.
- You can convert a CHAR column to the VARCHAR data type or convert a VARCHAR column to the CHAR data type **only if** the column contains null values or **if** you do not change the size.
- A change to the default value of a column **affects only** subsequent insertions to the table.

Dropping a Column

```
ALTER TABLE  dept80 DROP COLUMN  job_id
```

- Use the DROP COLUMN clause to drop columns you no longer need from the table.
- The column may or may not contain data.
- Using the ALTER TABLE statement, only one column can be dropped at a time.
- The table must have at least one column remaining in it after it is altered.
- Once a column is dropped, it cannot be recovered.

04

Drop Table

Drop Table

- `DROP TABLE TableName [RESTRICT | CASCADE]`

```
        e.g.  DROP TABLE PropertyForRent;
```

- Removes named table and all rows within it.
- Any pending transactions are committed.
- All indexes are dropped.
- You cannot roll back the DROP TABLE statement.

- With RESTRICT, if any other objects depend for their existence on continued existence of this table, SQL does not allow request.

- With CASCADE, SQL drops all dependent objects (and objects dependent on these objects).

05

Rename

Rename

```
RENAME dept TO detail_dept;
```

- To change the name of a table, view, sequence, or synonym, you execute the RENAME statement.
- You must be the owner of the object.

```
ALTER TABLE table_name RENAME COLUMN old_name to new_name;
```

06

Truncate

Truncate

```
TRUNCATE TABLE detail_dept;
```

- The `TRUNCATE TABLE` statement:
  - Removes all rows from a table
  - Releases the **storage space** used by that table

- You cannot roll back row removal when using `TRUNCATE`.
- Truncating a table does not fire the delete triggers of the table.
- If the table is the parent of a referential integrity constraint, you cannot truncate the table. Disable the constraint before issuing the TRUNCATE statement.

Delete in DML

```
Delete from detail_dept;
```

- You must be the owner of the table or have DELETE TABLE system privileges to truncate a table.
- The DELETE statement can also remove all rows from a table, but it does not release storage space.

Delete (DML) vs Truncate (DDL)

- Truncate:
  - Removes all rows.
  - Releases storage space (frees the space used by deleted rows).
  - Resets auto-increment values (depending on DBMS).
  - Does not fire DELETE triggers.
  - Is faster than DELETE for large datasets.
- Delete:
  - Removes rows one by one.
  - Does not release storage space immediately.
  - Fires DELETE triggers.
  - Is slower than TRUNCATE, especially for large tables.

07

Comment

Adding Comments to a Table

```
COMMENT ON TABLE employees IS 'Employee Information’;
COMMENT ON Column name IS ‘Name of Employee’;
```

- You can add a comment of up to 2,000 bytes about a column, table, view, or snapshot by using the COMMENT statement. The comment is stored in the data dictionary and can be viewed in one of the following data dictionary views in the COMMENTS column:
- You can drop a comment from the database by setting it to empty string (''):

```
COMMENT ON TABLE  employees IS ' ‘;
```

- Retrieve all comments of database:

```
select * from pg_description
join pg_class on pg_description.objoid = pg_class.oid
```

Conclusion

Statement	Description
CREATE TABLE	Creates a table
ALTER TABLE	Modifies table structures
DROP TABLE structure	Removes the rows and table
RENAME view,	Changes the name of a table, sequence, or synonym
TRUNCATE and	Removes all rows from a table releases the storage space
COMMENT view	Adds comments to a table or

Conclusion

CREATE TABLE

- Create a table.
- Create a table based on another table by using a subquery.

ALTER TABLE

- Modify table structures.
- Change column widths, change column data types, and add columns.

DROP TABLE

- Remove rows and a table structure.
- Once executed, this statement cannot be rolled back.

RENAME

- Rename a table, view, sequence, or synonym.

TRUNCATE

- Remove all rows from a table and release the storage space used by the table.
- The DELETE statement removes only rows.

COMMENT

- Add a comment to a table or a column.
- Query the data dictionary to view the comment.