# SQL (Structured Query Language)

## Database Design

Department of Computer Engineering

Sharif University of Technology

Maryam Ramezani maryam.ramezani@sharif.edu

# Introduction to DDL & DML

# Database Languages

- Database Languages are known as data Sublanguages.
- DBMSs have a facility for embedding the sublanguage in a high-level programming language e.g. C, C++, Java Or VB. The high-level language is then known as a host Language.
- Most data sublanguages also provide interactive commands that can be input directly from a terminal.

# Database Languages

- A data sublanguage consists of two parts:
  - Data Definition Language (DDL)
    - Used to specify the database schema.
  - Data Manipulation Language (DML)
    - Used to read and update the database.

# 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. A database schema is a logical grouping of objects that belong to a user.
- All created objects / structures (such as tables, views, indexes) are stored in a database schema.

# 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

# Data Dictionary

- The data dictionary integrates the meta-data; definitive information about the structure is recorded in a data dictionary.
  - For example: definitions about the records, data items and other objects of interest to users or required by the DBMS.

- The DBMS consults the data dictionary before accessing or manipulating the data.

- https://dataedo.com/kb/query/postgresql

# Data Manipulation Language

- This is a language that provides a set of operations to support the basic data manipulation operations on the data held in the database.

- Some of the operations include:
  - Insertion of new data into the database
  - Modification of data stored in the database
  - Retrieval of data contained in the database (Query language).
  - Deletion of data from the database.
- Examples of DML SQL statements;
  - INSERT
  - UPDATE
  - DELETE
  - MERGE

# Basic SQL Summary

# SQL Statements

| Statement | Description |
| --- | --- |
| SELECT | Retrieves data from the database |
| CREATE , ALTER, DROP, RENAME, TRUNCATE | DDL: Sets up, changes and removes data structures from tables |
| INSERT, UPDATE, DELETE<br>MERGE | DML: Enters new rows, changes existing rows and removes unwanted rows from tables in a database |
| COMMIT, ROLLBACK, SAVEPOINT | Transaction Control (TC): Manages changes made by DML. Changes to data may be grouped into logical transactions |
| GRANT, REVOKE | Data Control Language (DCL): Gives and removes access rights to both the database and the structures within it. |

# Database, Schema, Table

# Database and Schema

- **CREATE DATABASE** DatabaseName
- **DROP DATABASE** DatabaseName
- **CREATE SCHEMA** SchemaName
- **DROP SCHEMA** SchemaName

- Example: dbcourse.student means the student table in dbcourse schema

# Create table

- **CREATE TABLE** SQL syntax

  CREATE TABLE tablename
  (columnname1 data_type,
  columnname2 data_type, ⋯);

- Example:

  CREATE TABLE student
  (s_id CHAR(5),
  s_first VARCHAR2(20));

- Basic data types
  - Character
  - Number
  - Date/time
  - Large object

# Data Type

# ISO SQL Data Types

**Table 6.1** ISO SQL data types.

| Data type | Declarations | | | |
|---|---|---|---|---|
| boolean | BOOLEAN | | | |
| character | CHAR | VARCHAR | | |
| bit | BIT | BIT VARYING | | |
| exact numeric | NUMERIC | DECIMAL | INTEGER | SMALLINT |
| approximate numeric | FLOAT | REAL | DOUBLE PRECISION | |
| datetime | DATE | TIME | TIMESTAMP | |
| interval | INTERVAL | | | |
| large objects | CHARACTER LARGE OBJECT | | BINARY LARGE OBJECT | |

# Character Data Types

- VARCHAR
  - Variable-length character data (up to 4000 characters)
  - Syntax: *columnname* VARCHAR(*maximum_size*)
  - If user enters data value less than maximum_size, DBMS only stores actual character values
- CHAR
  - Fixed-length character data (default = 2000)
  - Syntax: *columnname* CHAR(*maximum_size*)
  - If user enters data value less than maximum_size, DBMS adds trailing blank spaces to the end of entry
  - The CHAR data type uses the storage more efficiently and processes data faster than the VARCHAR2 type.

# Character Example

```
create table Test_Char_Length (first char(4), second varchar(5))


insert into Test_Char_Length values ('12','123')
insert into Test_Char_Length values ('1234','1234')
insert into Test_Char_Length values ('1234','12345')
insert into Test_Char_Length values ('1234','123456')
insert into Test_Char_Length values ('12345','12345')


select * from test_char_length


SELECT CONCAT('(', first, ')'), CONCAT('(', second, ')') FROM
Test_Char_Length;
```

# Character Data Types

| CHAR | VARCHAR |
| --- | --- |
| Used to store strings of fixed size | Used to store strings of variable length |
| Can range in size from 1 to 8000 bytes | Can range in size from 1 to 8000 bytes |
| Uses a fixed amount of storage, based on the size of the column | Use varying amounts of storage space based on the size of the string stored. |
| Takes up 1 to 4 byte for each character, based on collation setting | Takes up 1 to 4 byte for each character based on collation and requires one or more bytes to store the length of the data |
| Better performance | Slightly poorer performance because length has to be accounted for. |
| Pads spaces to the right when storing strings less than the fixed size length | No padding necessary because it is variable in size |

# Number

- The NUMBER data type is used to store negative, positive, integer, fixed-decimal, and floating-point numbers.
- When a number type is used for a column, its **precision** and **scale** can be specified.
  - Precision is the total number of significant digits in the number, both to the left and to the right of the decimal point.
  - Scale is the total number of digits to the right of the decimal point.

# Number –– integer

- An **integer** is a whole number without any decimal part.
- The data type for it would be defined as NUMBER(3), where 3 represents the maximum number of digits.

# Number – fixed-point

- Decimal number has a specific number of digits to the right of the decimal point.
- The PRICE column has values in dollars and cents, which requires two decimal places – for example, values like 2.95, 3.99, 24.99, and so on.
- If it is defined as NUMBER(4,2), the first number specifies the **precision** and the second number the **scale**.

# Number – floating–point

- A **floating–point** decimal number has a variable number of decimal places
- To define such a column, do not specify the scale or precision along with the NUMBER type.
- By defining a column as a floating–point number, a value can be stored in it with very high precision

# Number Example

In postgres number types are NUMERIC(p, q), DECIMAL(p, q), REAL, INTEGER, SMALLINT, FLOAT(p), DOUBLE PRECISION. The types decimal and numeric are equivalent.

```
create table Test_Number (f1 numeric, f2 numeric (2), f3 numeric (2,1))

insert into Test_Number values (232.34,24,3.1)

select * from Test_Number
```

# Date And Time Data Types

- Datetime data subtypes
  - Store actual date and time values
  - DATE
  - TIMESTAMP
- Interval data subtypes
  - Store elapsed time interval between two datetime values
  - INTERVAL YEAR TO MONTH
  - INTERVAL DAY TO SECOND

# Auto Increment

- Use the PostgreSQL pseudo-type SERIAL to create an auto-increment column for a table.

Behind the scenes, the following statement:

```
CREATE TABLE table_name(
    id SERIAL
);
```

is equivalent to the following statements:

```
CREATE SEQUENCE table_name_id_seq;

CREATE TABLE table_name (
    id integer NOT NULL DEFAULT nextval('table_name_id_seq')
);

ALTER SEQUENCE table_name_id_seq
OWNED BY table_name.id;
```

# Auto Increment

- Mysql> Create table grocery_inventory ( id int not null primary key auto_increment, item_name varchar (50) not null, item_desc text, item_price float not null, curr_qty int not null);

- Auto_Increment is a table modifier/constraint that will request MySQL to add the next available number to the ID field for you.

- Postgres Example:

```
CREATE TABLE CountNum (name char(5),regNo serial)
insert into CountNum values('DB'),('DS');
select * from CountNum
```

# Date And Time Data Types (continued)

- DATE
  - Stores dates from Dec 31, 4712 BC to Dec 31, AD 4712
  - Default date format: DD-MON-YY
  - Default time format: HH:MI:SS AM
  - Syntax: *columnname* DATE
- TIMESTAMP
  - Stores date values similar to DATE data type
  - Also stores fractional seconds
  - Syntax: *columnname* TIMESTAMP (*fractional_seconds_precision*)

    If omitted, default is 6 decimal place
  - Example: *shipment_date* TIMESTAMP(2)

# Date And Time Data Types (continued)

- INTERVAL YEAR TO MONTH
  - Stores time interval expressed in years and months using the following syntax:
  - Example:
    - `create table Interval_Time (time_enrolled INTERVAL YEAR TO MONTH)`
    - `insert into Interval_Time values (INTERVAL '13' MONTH),(INTERVAL '1' MONTH),(INTERVAL '18' MONTH);`
    - `select * from Interval_Time`

# Large Object (LOB) Data Types

- Store binary data such as:
  - Digitized sounds or images
  - References to binary files from word processor or spreadsheet
- How? Additional topic for study.

# Create Table

# 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 'M');


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
- Imposed in order to protect the database from becoming inconsistent

## 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
```

## Domain Constraints

- Every column has a domain, in other words a set of legal values. **"CHECK"**
- **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))`

## 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

## 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:

```
UNIQUE(telNo)
pno  VARCHAR(5) NOT NULL UNIQUE;
```

# 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:

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

- Example:

```
FOREIGN KEY(bNo) REFERENCES Branch (branchNo)
```

# References

- ref