Creating Databases and Tables

Section Overview

- Data Types
- Primary and Foreign Keys
- Constraints
- CREATE
- INSERT
- UPDATE
- DELETE, ALTER, DROP

Data Types

1. Boolean (T/F)
2. Character (char, varchar, text)
3. Numeric (integer, floating-point number)
4. Temporal (date, time timestamp, interval)
5. Not common types:
    a. UUID (University Unique Identifiers
    b. Array (Stores an array of strings, numbers, etc)
    c. JSON
    d. Hstore key-value pair
    e. Special types such as network address and geometric data

- When creating databases and tables, you should carefully consider which data types should be used for the data to be stored.
- https://www.postgresql.org/docs/9.6/datatype.html

- For example:
    - Store a phone number, should it be stored as numeric? If so, which type of numeric
    - It makes sense to store it as a BIGINT data type, but we should really be thinking what is best for the situation. We don't perform arithmetic with phone numbers, so it probably makes more sense as a **VARCHAR** data type instead.
    - In fact, searching for the best practice online, you will discover its usually recommended to store as a text-based data type due to a variety of issues
        - No arithmetic performed
        - Leading zeros could cause issues, 7 and 07 treated same numerically, but are not the same phone number

- When creating a database and table, take you time to plan for long term storage

- Remember you can always remove historical information you've decided you aren't using, but you can't go back in time to add in information!

Primary and Foreign Key

- A primary key is a column or a group of columns used to identify a row uniquely in a table

    - For example, in our dvdrental database, we saw customers has a **unique**, **non-null** customer_id column as their primary key.

    - Primary keys are also important since they allow us to easily discern what columns should be used for joining tables together

- A foreign key is a field or group of fields in a table that uniquely identifies a row in another table (a foreign key is defined in a table that references to the primary key of the other table)

    - The table that contains the foreign key is called **referencing table** or **child table**

    - The table to which the foreign key references is called **referenced table** or **parent table**

    - A table can have **multiple foreign keys** depending on its relationships with other tables

    - For example, in the dvdrental database payment table, each payment row had its unique payment_id (a primary key) and identified the customer that made the payment through the customer_id (a foreign key since it references the customer table's primary key)

- When creating tables and defining columns, we can use constraints to define columns as being a primary key or attaching a foreign key relationship to another table

Constraints

- Constraints are the rules enforced on data columns on table
- These are used to prevent invalid data from being entered into the database
- This ensures the accuracy and reliability of the data in the database
- Constraints can be divided into two main categories:
    - Column Constraints
        - Constraints the data in a column to adhere to certain conditions
    - Table Constraints
        - Applied to the entire table rather than to an individual column
- The most common column constraints used:
    - **NOT NULL** Constraint
        - Ensure that a column cannot have NULL value
    - **UNIQUE** Constraint
        - Ensure that all values in a column are different
    - **PRIMARY** key
        - Uniquely identifies each row/record in a database
    - **FOREIGN** key
        - Constrains data based on columns in other tables
    - **CHECK** Constraint
        - Ensures that all values in a column satisfy certain conditions
    - **EXCLUSION** Constraint
        - Ensures that if any two rows are compared on the specified column or expression using the specified operator, not all of these comparisons will return TRUE
- Table constrains:
    - **CHECK** (condition)
        - to check a condition when inserting or updating data
    - **REFERENCES**
        - To constrain the value stored in the column that must exist in a column in another table
    - **UNIQUE** (column_list)
        - Forces the values stored in the columns listed inside the parentheses to be unique
    - **PRIMARY KEY** (column_list)
        - Allows you to define the primary key that consists of multiple columns

CREATE table

Full General Syntax:
CREATE TABLE table_name (
    column_name **TYPE** column_constraint,
    column_name **TYPE** column_constraint,
    table_constraint table_constraint
) **INHERITS** existing_table_name;

Common Simple Syntax:
CREATE TABLE table_name (
    column_name **TYPE** column_constraint,
    column_name **TYPE** column_constraint,
)

EX:
CREATE TABLE players (
    Player_id **SERIAL PRIMARY KEY**,
    Age **SMALLINT NOT NULL**
    );

SERIAL

- In PostgreSQL, a sequence is a special kind of database object that generates a sequence of integers
- A sequence is often used as the primary key column in a table
- It will create a sequence object and set the next value generated by the sequence as the default value for the column
- This is perfect for a primary key, because it logs unique integer entries for you automatically upon insertion
- If a row is later removed, the column with the SERIAL data type will **not** adjust, making the fact that a row was removed from the sequence, for example:
    - 1, 2, 3, 5, 6, 7
        - You know row 4 was removed at some point

`smallserial`	2 bytes	small autoincrementing integer	1 to 32767
`serial`	4 bytes	autoincrementing integer	1 to 2147483647
`bigserial`	8 bytes	large autoincrementing integer	1 to 9223372036854775807

EX:

1. Create table

```
CREATE TABLE account(
    user_id SERIAL PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(50) NOT NULL,
    email VARCHAR(250) UNIQUE NOT NULL,
    created_on TIMESTAMP NOT NULL,
    last_login TIMESTAMP
)
```

2. Create another table and then link them through a foreign key

```
CREATE TABLE job(
    job_id SERIAL PRIMARY KEY,
    job_name VARCHAR(200) UNIQUE NOT NULL
)
```

(SERIAL is integer)

```
CREATE TABLE account_job(
    user_id INTEGER REFERENCES account(user_id),
    job_id INTEGER REFERENCES job(job_id),
    hire_date TIMESTAMP
)
```

INSERT table

- INSERT allows you to add in rows to a table
- The inserted row values must match up for the table, including constraints
- SERIAL columns do not need to be provided a value

General Syntax:
INSERT INTO table (coulmn1, column2, …)
VALUES
(value1, value2, …),
(value1, value2, …),
…;

Inserting values from another table:
INSERT INTO table (column1, column2, …)
SELECT column1, column2, …
FROM another_table
WHERE condition;

Ex:
INSERT INTO account (username, password, email, created_on)
VALUES
('Jose', 'password', 'jose@mail.com', CURRENT_TIMESTAMP)

```
1  SELECT * FROM account
2
```

Data Output | Messages | Explain | Notifications

	user_id [PK] integer	username character varying (50)	password character varying (50)	email character varying (250)	created_on timestamp without time zone	last_login timestamp without time zone
1	1	Jose	password	jose@mail.com	2020-10-10 20:33:30.029791	[null]

UPDATE keyword

- Allows for the changing of values of the columns in a table

General Syntax:
UPDATE table
SET column1 = value1,
Column2 = value2, …
WHERE condition;

EX:
UPDATE account
SET last_login = **CURRENT_TIMESTAMP**
WHERE last_login **IS NULL**;

Set based on another column:
UPDATE account
SET last_login = created_on

Using another table's values (UPDATE join):
UPDATE TableA
SET original_col = TableB.new_col
FROM TableB
WHERE TableA.id = TableB.id

EX:
UPDATE account_job
SET hire_date = account.created_on
FROM account
WHERE account_job.user_id = account.user_id

Return affected rows:
UPDATE account
SET last_login = created_on
RETURNING account_id, last_login

EX:
UPDATE account
SET last_login = CURRENT_TIMESTAMP
RETURNING email, created_on, last_login

	email character varying (250)	created_on timestamp without time zone	last_login timestamp without time zone
1	jose@mail.com	2020-10-10 20:33:30.029791	2020-10-10 20:53:04.953055

DELECT Clause

- Remove rows from a table
- Similar to UPDATE command, you can also add in a RETURNING call to return rows that were removed

General Syntax:
DELECT FROM table
WHERE condition

Delete rows based on their presence in other tables:
DELETE FROM TableA
USING TableB
WHERE TableA.id = TableB.id

Delete all rows from a table:
DELETE FROM Table

Return affected rows:

Ex:
DELETE FROM job
WHERE job_name = 'Cowboy'
RETURNING job_id, job_name

ALTER Table

- Allows for changes to an existing table structure, such as:
    - Adding, dropping, or renaming columns
    - Changing a column's data type
    - Set DEFAULT values for a column
    - Add CHECK constraints
    - Rename table

- https://www.postgresql.org/docs/9.6/sql-altertable.html

General Syntax:
ALTER TABLE table_name action

Ex:
ADD COLUMN new_col **TYPE**
DROP COLUMN col_name
SET DEFAULT value
DROP DEFAULT
SET NOT NULL
DROP NOT NULL
ADD CONSTRAINT constraint_name

Rename table name:
ALTER TABLE information
RENAME TO new_info

Rename column name:
ALTER TABLE new_info
RENAME COLUMN person **TO** people

Drop constraint:
ALTER TABLE new_info
ALTER COLUMN people **DROP NOT NULL**

DROP keyword

- Allows for the complete removal of a column in a table
- In PostgreSQL, this will also automatically remove all of its indexes and constraints involving the column
- However, it will not remove columns used in views, triggers, or stored procedures without the additional CASCADE clause

General Syntax:
ALTER TABLE table_name
DROP COLUMN col_name

Remove all dependencies:
ALTER TABLE table_name
DROP COLUMN col_name CASCADE

Check for existence to avoid error:
ALTER TABLE table_name
DROP COLUMN IF EXISTS col_name

Drop multiple columns:
ALTER TABLE table_name
DROP COLUMN col_one,
DROP COLUMN col_two

CHECK constraint

- Allows us to create more customized constraints that adhere to a certain condition
    - Such as making sure all inserted integer values fall below a certain threshold

General Syntax:
CREATE TABLE example (
Ex_id **SERIAL PRIMARY KEY**,
Age **SAMLLINT CHECK** (age > 21),
Parent_age **SAMLLINT CHECK** (parent_age > age)
);

EX:

```
CREATE TABLE employees(
        emp_id SERIAL PRIMARY KEY,
        first_name VARCHAR(50) NOT NULL,
        last_name VARCHAR(50) NOT NULL,
        birthdate DATE CHECK (birthdate > '1900-01-01'),
        hire_date DATE CHECK (hire_date > birthdate),
        salary INTEGER CHECK (SALARY > 0)
)
```

Notice:
If enter '-100' for 'salary', it will show:

Error Message new row for relation "employees" violates check constraint "employees_salary_check"

And serial will count.

The emp_id is 2 and 4, it's because when we insert, for Jose, we insert wrong birthdate, for Sammy, we insert wrong salary.

	emp_id [PK] integer	first_name character varying (50)	last_name character varying (50)	birthdate date	hire_date date	salary integer
1	2	Jose	Portilla	1988-11-03	2010-01-01	100
2	4	Sammy	Smith	1990-11-03	2010-01-01	100

Assessment Test 3

Create a new database called "School" this database should have two tables: **teachers** and **students.**

The **students** table should have columns for student_id, first_name,last_name, homeroom_number, phone,email, and graduation year.

The **teachers** table should have columns for teacher_id, first_name, last_name, homeroom_number, department, email, and phone.

The constraints are mostly up to you, but your table constraints do have to consider the following:

1. We must have a phone number to contact students in case of an emergency.
2. We must have ids as the primary key of the tables
3. Phone numbers and emails must be unique to the individual.

Once you've made the tables, insert a student named Mark Watney (student_id=1) who has a phone number of 777-555-1234 and doesn't have an email. He graduates in 2035 and has 5 as a homeroom number.
Then insert a teacher names Jonas Salk (teacher_id = 1) who as a homeroom number of 5 and is from the Biology department. His contact info is: jsalk@school.org and a phone number of 777-555-4321.

```
CREATE TABLE students (
        student_id SERIAL PRIMARY KEY,
        first_name VARCHAR (50) NOT NULL,
        last_name VARCHAR (50) NOT NULL,
        homeroom_number INTEGER,
        phone_number VARCHAR (50) UNIQUE NOT NULL,
        email VARCHAR (50) UNIQUE,
        graduation_year INTEGER
)

CREATE TABLE teachers (
        teacher_id SERIAL PRIMARY KEY,
        first_name VARCHAR (50) NOT NULL,
        last_name VARCHAR (50) NOT NULL,
        homeroom_number INTEGER,
        department VARCHAR (50),
        phone_number VARCHAR (50) UNIQUE,
        email VARCHAR (20) UNIQUE
)
```

```
INSERT INTO students (
        first_name,
        last_name,
        homeroom_number,
        phone_number,
        graduation_year
)
VALUES (
        'Mark',
        'Watney',
        5,
        '777-555-1234',
        2035
)


INSERT INTO teachers (
        first_name,
        last_name,
        homeroom_number,
        department,
        phone_number,
        email
)
VALUES (
        'Jonas',
        'Salk',
        5,
        'Biology department',
        '777-555-4321',
        'jsalk@school.org'
)
```