Data Definition

CSC 256, SQL Programming

SQL Overview

- DDL data definition language: create, alter, drop
- DML data manipulation language: insert, update, delete, select
- DCL data control language: grant, revoke
- TCL transaction control language: begin, commit, rollback

Table Operations

- Table operations are part of the data definition language (DDL)
- Operations:
  - Creating tables
  - Altering tables
  - Deleting tables
  - Adding constraints to tables

CREATE

- CREATE syntax:

```
create table table_name (
  col1_name col_type(field_length) col1 constraints,
  ...
  table constraints
);
```

- Common table naming conventions
  - snake case
  - singular instead of plural (some relations could get strange names: `user_has_role` versus `users_have_roles`)

CREATE Example

- Model a user

```
create table user (
  first_name varchar(100),
  last_name varchar(100),
  email varchar(100),
  dob date
);
```

- Note: this table does not conform to relational model because it does not have constraints.

CREATE Example (continued)

- Change the create to be idempotent (can run the command multiple times with the same result.)

```
create table if not exists user (
  first_name varchar(100),
  last_name varchar(100),
  email varchar(100),
  dob date
);
```

- Note: it is a point of contention on whether a SQL script should be idempotent or not.

INSERT a Row

- Basic INSERT syntax (more on this in the next lecture)

```
insert into table_name
values (col1_value, col2_value, ...);
```

- Note: with this syntax the values for a record must be in the order defined in the CREATE statement for the table.

ALTER: Add a Column

- Syntax to add a new column to an existing table

```
alter table table_name
  add col_name col_type;
```

- Note: a new column added with ALTER will have NULL values for existing records

ALTER: Remove a Column

- Syntax to remove a column from an existing table

```
alter table table_name
  drop col_name;
```

- Note: be careful – all data from the removed column is permanently deleted

DROP TABLE

- Syntax to remove a table

  ```
  drop table table_name;
  ```

- Note: be careful – all data from the removed table is permanently deleted

CREATE SCHEMA (PostgeSQL)

- A PostgreSQL schema is a namespace for tables and other database objects.
- Example: schema creation

  ```
  create schema example;
  ```

- The default schema for a newly created database is called `public`; database objects in the `public` schema do not need to be prefixed with the schema name.

Schema Examples

- Example: create a table in a schema

```
create table if not exists example.users (
  first_name varchar(100),
  last_name varchar(100),
  email varchar(100),
  dob date
);
```

- Exampe: drop a schema

```
drop schema example cascade;
```

  - Note: CASCADE recursively deletes everything

Default Values

- During table creation, we can specify default values for columns if a value is not supplied during an insert.
- Example:

```
create table if not exists user (
  first_name varchar(100),
  last_name varchar(100),
  email varchar(100),
  dob date,
  created timestamp default now()
);
```

Primary Keys

- A primary key is a column or set of columns (composite) that uniquely identify a record.
- Types of primary keys:
  - Natural key: column values that are "naturally" unique for the data model
  - Surrogate key: artificial key to guarantee uniqueness
- We can specify a primary key during table creation; this guarantees that inserted values are unique and not NULL (an error occurs if these constraints are violated.)

Primary Key Example

- Specify `email` as a natural primary key

```
create table if not exists users (
  email varchar(100) primary key,
  first_name varchar(100),
  last_name varchar(100),
  dob date
);
```

Primary Key Syntax

- Inline (column constraint)

```
CREATE TABLE <table name> (
  <column name> <column type> [CONSTRAINT <constraint n
  ...
);
```

- After column definitions (table constraint)

```
CREATE TABLE <table name> (
  ...
  [CONSTRAINT <constraint name>] PRIMARY KEY (<column n
);
```

Example: Surrogate Key

- A classic beginner project is a TODO list application. Let us model the TODO list:

```
create table todo (
  item text,
  user_email varchar(100)
);
```

- Here we do not have a reasonable natural key; use a surrogate key

```
create table todo (
  todo_id integer,
  item text,
  user_email varchar(100)
);
```

Surrogate Key Value Generation

- A drawback of surrogate keys is finding a valid value for a new record.
- Most RDBMS systems have a method to automatically generate surrogate key values
- PostgreSQL has the `serial` type for this purpose:

```
create table todo (
  todo_id serial,
  item text,
  user_email varchar(100)
);
```

  - Note: when we insert a new record, we do not specify the `item_id` value

Surrogate Key Value Generation (continued)

- PostgrSQL has a newer, more standards compliant syntax

```
create table todo (
  todo_id bigint generated by default as identity prima
  item text,
  user_email varchar(100)
);
```

Specifying Composite Keys

- A composite key is a primary key composed of multiple column values
- Suppose we want to tag TODO items, but not more than once with the same tag

```
create table todo_tags (
  todo_id bigint,
  tag varchar(100),
  primary key (todo_id, tag)
);
```

  - Note: here we can not use the column constraint syntax

Foreign Keys

- Based on the previous examples, we are able to create TODO items that have have an email that does not match an existing email in the user table.
- Foreign keys are a way to protect referential integrity, that is, ensure that references to data in other tables are valid.
- Example (inline syntax):

```
create table todo (
  todo_id serial,
  item text,
  user_email varchar(100) references user (email)
);
```

Foreign Key Example

- We also want to ensure that tags reference a valid TODO item (table syntax):

```
create table todo_tags (
  todo_id bigint,
  tag varchar(100),
  primary key (todo_id, tag)
  foreign key (todo_id) references todo (todo_id)
);
```

CASCADE

- CASCADE allows us to maintain referential integrity (so, related to foreign keys)
- We may need to delete any rows referencing the deleted row, or update the values of the referencing column(s) to the new values of the referenced columns.
- Example (inline syntax):

```
create table todo (
  todo_id serial,
  item text,
  user_email varchar(100) references user (email)
    on update cascade
    on delete cascade
);
```

CASCADE (continued)

- Example (table syntax):

```
create table todo_tags (
  todo_id bigint,
  tag varchar(100),
  primary key (todo_id, tag)
  foreign key (todo_id) references todo (todo_id)
    on update cascade
    on delete cascade
);
```

- Explanation: in the previous two examples, if a user is updated/deleted from the `user` table, then the references to corresponding email are updated/deleted in the `todo` and `todo_tags` tables automatically.

Additional Constraints

- `CHECK`: ensure that the value of a column satisfies a Boolean condition
- `NOT NUL`: ensure the value of a column is not null
- `UNIQUE`: ensure the value of a column (or set of columns) is unique

CHECK Example

- Let us add a simple check to ensure that an email contains the `@` character.

```
create table if not exists users (
  email varchar(100) primary key,
  first_name varchar(100),
  last_name varchar(100),
  dob date,
  check (position('@' in email) > 0)
);
```

Additional Constraints Example

- Example: ensure that a TODO item is unique and cannot be NULL

```
create table todo (
  todo_id serial,
  item text not null unique,
  user_email varchar(100) references user (email)
    on update cascade
    on delete cascade
);
```