1. Get all invoices where the `unit_price` on the `invoice_line` is greater than $0.99.

```
SELECT *
FROM INVOICE
LEFT JOIN INVOICE_LINE
ON INVOICE.INVOICE_ID = INVOICE_LINE.INVOICE_ID
WHERE TOTAL > .99
```

2. Get the `invoice_date`, customer `first_name` and `last_name`, and `total` from all invoices.

```
SELECT INVOICE_DATE, FIRST_NAME, LAST_NAME, TOTAL
FROM CUSTOMER
LEFT JOIN INVOICE
ON CUSTOMER.CUSTOMER_ID = INVOICE.CUSTOMER_ID
```

3. Get the customer `first_name` and `last_name` and the support rep's `first_name` and `last_name` from all customers.
    - Support reps are on the employee table.

```
SELECT CUSTOMER.FIRST_NAME, CUSTOMER.LAST_NAME, EMPLOYEE.FIRST_NAME
FROM CUSTOMER
LEFT JOIN EMPLOYEE
ON CUSTOMER.SUPPORT_REP_ID = EMPLOYEE.EMPLOYEE_ID
```

4. Get the album `title` and the artist `name` from all albums.

```
SELECT TITLE, NAME
FROM ALBUM
LEFT JOIN ARTIST
ON ALBUM.ARTIST_ID = ARTIST.ARTIST_ID
```

5. Get all playlist_track track_ids where the playlist `name` is Music.

```
SELECT TRACK_ID
FROM PLAYLIST_TRACK
LEFT JOIN PLAYLIST
ON PLAYLIST_TRACK.PLAYLIST_ID = PLAYLIST.PLAYLIST_ID
WHERE PLAYLIST.NAME = 'Music'
```

WITHOUT A JOIN – USING IN AND SUB-QUERY

```
SELECT TRACK_ID
FROM PLAYLIST_TRACK
WHERE PLAYLIST_ID IN (
  SELECT PLAYLIST_ID
  FROM PLAYLIST
  WHERE NAME = 'Music')
```

6. Get all track `name`s for `playlist_id` 5.

```
SELECT TRACK.NAME
FROM TRACK
LEFT JOIN PLAYLIST_TRACK
ON TRACK.TRACK_ID = PLAYLIST_TRACK.TRACK_ID
WHERE PLAYLIST_ID = 5
```

WITHOUT A JOIN – USING IN AND SUB-QUERY

```
SELECT NAME
FROM TRACK
WHERE TRACK_ID IN (
  SELECT TRACK_ID
  FROM PLAYLIST_TRACK
  WHERE PLAYLIST_ID IN (
            SELECT PLAYLIST_ID
    FROM PLAYLIST
    WHERE PLAYLIST_ID = 5))
```

7. Get all track `name`s and the playlist `name` that they're on (2 joins).

```
SELECT TRACK.NAME TRACK, PLAYLIST.NAME PLAYLIST
FROM TRACK
LEFT JOIN PLAYLIST_TRACK
  ON TRACK.TRACK_ID = PLAYLIST_TRACK.TRACK_ID
LEFT JOIN PLAYLIST
  ON PLAYLIST_TRACK.PLAYLIST_ID = PLAYLIST.PLAYLIST_ID
```

8. Get all track `name`s and album `title`s that are the genre `Alternative & Punk` (2 joins).

```
SELECT TRACK.NAME, ALBUM.TITLE
FROM TRACK
LEFT JOIN ALBUM
ON TRACK.ALBUM_ID = ALBUM.ALBUM_ID
LEFT JOIN GENRE
ON TRACK.GENRE_ID = GENRE.GENRE_ID
WHERE GENRE.NAME = 'Alternative & Punk'
```

Practice nested queries

1. Get all invoices where the `unit_price` on the `invoice_line` is greater than $0.99.

```
SELECT * FROM INVOICE
WHERE INVOICE_ID IN (
  SELECT INVOICE_ID FROM INVOICE_LINE
  WHERE TOTAL > .99)
```

2. Get all playlist tracks where the playlist name is Music.

```
SELECT *
FROM PLAYLIST
WHERE PLAYLIST.PLAYLIST_ID IN
(SELECT PLAYLIST_ID
 FROM PLAYLIST
 WHERE NAME = 'Music')
```

3. Get all track names for `playlist_id` 5.

```
SELECT TRACK.NAME
FROM TRACK
WHERE TRACK_ID IN (
  SELECT TRACK_ID
  FROM PLAYLIST_TRACK
  WHERE PLAYLIST_ID IN (
    SELECT PLAYLIST_ID
    FROM PLAYLIST
    WHERE PLAYLIST_ID = 5))
```

4. Get all tracks where the `genre` is Comedy.

```
SELECT *
FROM TRACK
WHERE GENRE_ID IN (
  SELECT GENRE_ID
  FROM GENRE
  WHERE NAME = 'Comedy')
```

5. Get all tracks where the `album` is Fireball.

```
SELECT *
FROM TRACK
```

```
WHERE ALBUM_ID IN (
  SELECT ALBUM_ID
  FROM ALBUM
  WHERE TITLE = 'Fireball')
```

6. Get all tracks for the artist Queen (2 nested subqueries).

```
SELECT *
FROM TRACK
WHERE ALBUM_ID IN (
  SELECT ALBUM_ID
  FROM ALBUM
  WHERE ARTIST_ID IN (
    SELECT ARTIST_ID
    FROM ARTIST
    WHERE NAME = 'Queen'))
```

Practice updating Rows

1. Find all customers with fax numbers and set those numbers to `null`.

```
UPDATE CUSTOMER
SET FAX = NULL
WHERE FAX IS NOT NULL
```

2. Find all customers with no company (null) and set their company to `"Self"`.

```
UPDATE CUSTOMER
SET COMPANY = 'Self'
WHERE COMPANY IS NULL
```

3. Find the customer `Julia Barnett` and change her last name to `Thompson`.

```
UPDATE CUSTOMER
SET LAST_NAME = 'Thompson'
WHERE FIRST_NAME = 'Julia'
  AND LAST_NAME = 'Barnett'
```

4. Find the customer with this email `luisrojas@yahoo.cl` and change his support rep to `4`.

```
UPDATE CUSTOMER
SET SUPPORT_REP_ID = 4
WHERE EMAIL = 'luisrojas@yahoo.cl'
```

5. Find all tracks that are the genre `Metal` and have no composer. Set the composer to `"The darkness around us"`.

```
UPDATE TRACK
SET COMPOSER = 'The darkness around us'
WHERE GENRE_ID IN (
  SELECT GENRE_ID
  FROM GENRE
  WHERE NAME = 'Metal')
  AND COMPOSER IS NULL
```

6. Refresh your page to remove all database changes.

Group by

1. Find a count of how many tracks there are per genre. Display the genre name with the count.

```
SELECT GENRE.NAME, COUNT(TRACK)
FROM GENRE
LEFT JOIN TRACK
ON GENRE.GENRE_ID = TRACK.GENRE_ID
```

```
GROUP BY GENRE.NAME
```

2. Find a count of how many tracks are the `"Pop"` genre and how many tracks are the `"Rock"` genre.

```
SELECT GENRE.NAME, COUNT(TRACK)
FROM GENRE
LEFT JOIN TRACK
ON GENRE.GENRE_ID = TRACK.GENRE_ID
GROUP BY GENRE.NAME
HAVING GENRE.NAME IN ('Rock', 'Pop')
```

3. Find a list of all artists and how many albums they have.

```
SELECT ARTIST.NAME, COUNT(ALBUM.ALBUM_ID)
FROM ARTIST
LEFT JOIN ALBUM
ON ARTIST.ARTIST_ID = ALBUM.ARTIST_ID
GROUP BY ARTIST.NAME
```

Use Distinct

1. From the `track` table find a unique list of all `composer`s.

```
SELECT DISTINCT COMPOSER
FROM TRACK
```

2. From the `invoice` table find a unique list of all `billing_postal_code`s.

```
SELECT DISTINCT BILLING_POSTAL_CODE
FROM INVOICE
```

3. From the `customer` table find a unique list of all `company`s.

```
SELECT DISTINCT COMPANY
FROM CUSTOMER
```

Delete Rows

1. Copy, paste, and run the SQL code from the summary.
2. Delete all `'bronze'` entries from the table.

```
DELETE FROM PRACTICE_DELETE
WHERE TYPE = 'bronze'
```

3. Delete all `'silver'` entries from the table.

```
DELETE FROM PRACTICE_DELETE
WHERE TYPE = 'silver'
```

4. Delete all entries whose value is equal to `150`.

```
DELETE FROM PRACTICE_DELETE
WHERE VALUE = 150
```

eCommerce Simulation - No Hints

Let's simulate an e-commerce site. We're going to need users, products, and orders.

- users need a name and an email.

```
create table users (
  id serial PRIMARY KEY,
  name varchar(255),
```

```
  email char(255));
```

- products need a name and a price

```
create table products (
  id serial PRIMARY KEY,
  name varchar(255),
  price float);
```

- orders need a ref to product.

```
CREATE TABLE orders (
  id serial PRIMARY KEY,
  quantity integer,
  products_id integer,
  FOREIGN KEY(products_id) REFERENCES products(id));
```

- All 3 need primary keys.
- Create 3 tables following the criteria in the summary.
- Add some data to fill up each table.
  - At least 3 users, 3 products, 3 orders.

```
insert into users (name, email)
values
('Fred', 'fred@yahoo.com'),
('Wilma', 'wilma@yahoo.com'),
('Pebbles', 'pebbles@yahoo.com');


insert into products (name, price)
values
('apples', 1),
('bananas', 2),
('oranges', 3);

insert into orders (quantity, products_id)
values
(2, 1),
(3, 2),
(4, 2);
```

- Run queries against your data.
- Get all products for the first order.
  SELECT NAME FROM PRODUCTS WHERE ID IN (SELECT PRODUCTS_ID FROM ORDERS WHERE ID = 1)
- Get all orders.
  SELECT * FROM ORDERS

- Get the total cost of an order (sum the price of all products on an order).

```
SELECT SUM(PRODUCTS.PRICE * ORDERS.QUANTITY)
FROM ORDERS
LEFT JOIN PRODUCTS
ON ORDERS.PRODUCTS_ID = PRODUCTS.ID
WHERE ORDERS.ID = 1
```

- Add a foreign key reference from orders to users.

```
ALTER TABLE orders
ADD users_id INTEGER

ALTER TABLE orders
ADD FOREIGN KEY (users_id) REFERENCES users(id);
```

- Update the orders table to link a user to each order.

```
UPDATE orders
SET users_id = 1
```

- Run queries against your data.
    - Get all orders for a user.

```
SELECT USERS.NAME, ORDERS.ID, PRODUCTS.NAME
FROM ORDERS
LEFT JOIN PRODUCTS
ON ORDERS.PRODUCTS_ID = PRODUCTS.ID
LEFT JOIN USERS
ON ORDERS.USERS_ID = USERS.ID
WHERE USERS.ID = 1
```

    - Get how many orders each user has.

```
SELECT USERS.NAME, COUNT(ORDERS.ID)
FROM ORDERS
RIGHT JOIN USERS
ON ORDERS.USERS_ID = USERS.ID
GROUP BY USERS.NAME
```