Digging Deeper with PostgreSQL

Breaking up processes and understanding performance

Agenda

- Case Study (10m)
- Key Concepts Review (10m)
- Advanced Functions (20m)
- Database Setup (5m)
- Explain and performance review with PGAdmin 4 (20m)
- Organizing: With, sub-select, temporary tables and views (1.5h)
    - Performance reviews included
- Indexing (30m)

Case Study Overview

- Shopping experience
- Primary keys will always be ‘id’ column in tables
- All foreign keys will refer to: <table_name>_id

Case Study: Shopping site

Shopping Components

Products
Product Types
Users
Orders
Carts

Case Study: Shopping site

Shopping Components

Products

Product Types

Users

Orders

Carts

products

- id
- product_type_id
- price
- name
- grams
- address

Case Study: Shopping site

Shopping Components

Products

Product Types

Users

Orders

Carts

products

id
product_type_id
price
name
grams
address

product_types

id
name
description

Case Study: Shopping site

Shopping Components

Products

Product Types

Users

Orders

Carts

products

id

product_type_id

price

name

grams

address

product_types

id

name

description

Case Study: Shopping site

Shopping Components

Products
Product Types
Users
Orders
Carts

products
- id
- product_type_id
- price
- name
- grams
- address

product_types
- id
- name
- description

users
- id
- name
- address
- email
- password

Case Study: Shopping site

Shopping Components

Products
Product Types
Users
Orders
Carts

products
- id
- product_type_id
- price
- name
- grams
- address

product_types
- id
- name
- description

users
- id
- name
- address
- email
- password

orders
- id
- user_id
- status
- shipping_total
- order_total

Case Study: Shopping site

Shopping Components

Products
Product Types
Users
Orders
Carts

products
- id
- product_type_id
- price
- name
- grams
- address

product_types
- id
- name
- description

users
- id
- name
- address
- email
- password

orders
- id
- user_id
- status
- shipping_total
- order_total

carts
- id
- user_id
- last_updated

Case Study: Shopping site

Shopping Components

Products
Product Types
Users
Orders
Carts

products
- id
- product_type_id
- price
- name
- grams
- address

product_types
- id
- name
- description

users
- id
- name
- address
- email
- password

orders
- id
- user_id
- status
- shipping_total
- order_total

carts
- id
- user_id
- last_updated

Case Study: Shopping site

Shopping Components

Products
Product Types
Users
Orders
Carts

products
- id
- product_type_id
- price
- name
- grams
- address

product_types
- id
- name
- description

users
- id
- name
- address
- email
- password

orders
- id
- user_id
- status
- shipping_total
- order_total

carts
- id
- user_id
- last_updated

Case Study: Shopping site

products
- id
- product_type_id
- price
- name
- grams
- address

product_types
- id
- name
- description

users
- id
- name
- address
- email
- password

orders
- id
- user_id
- status
- shipping_total
- order_total

carts
- id
- user_id
- last_updated

Case Study: Shopping site

products
- id
- product_type_id
- price
- name
- grams
- address

product_types
- id
- name
- description

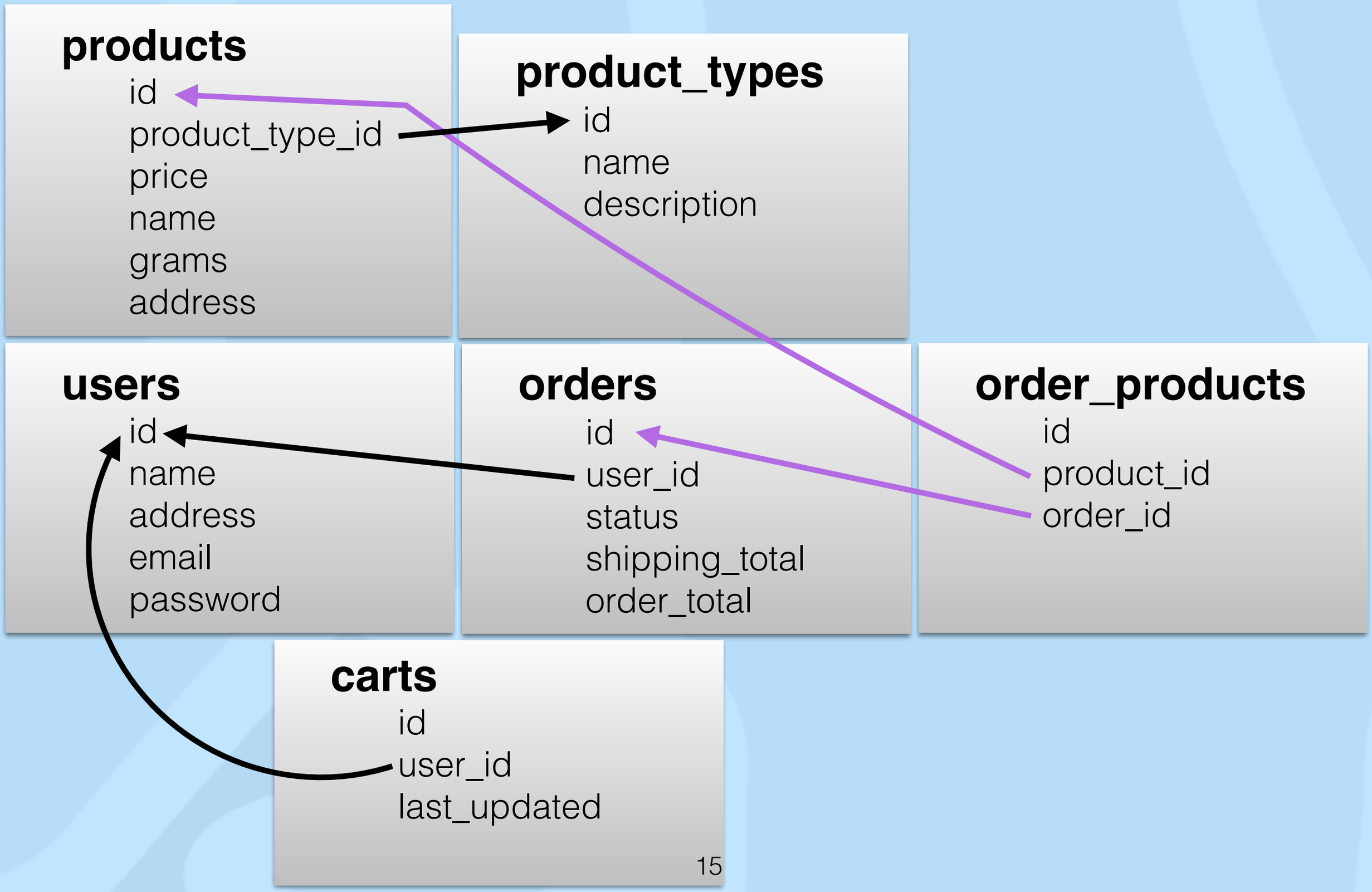
users
- id
- name
- address
- email
- password

orders
- id
- user_id
- status
- shipping_total
- order_total

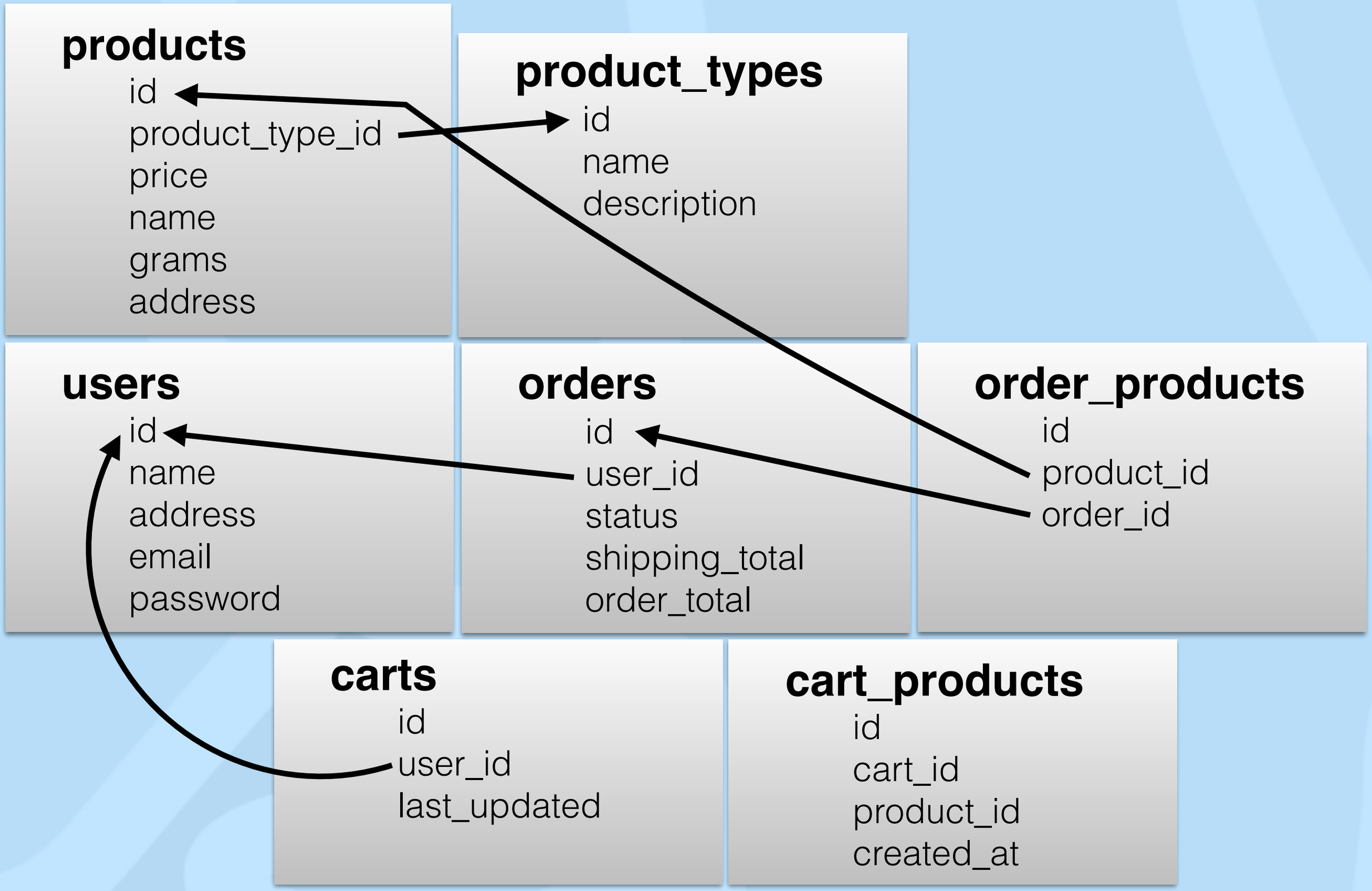
order_products
- id
- product_id
- order_id

carts
- id
- user_id
- last_updated

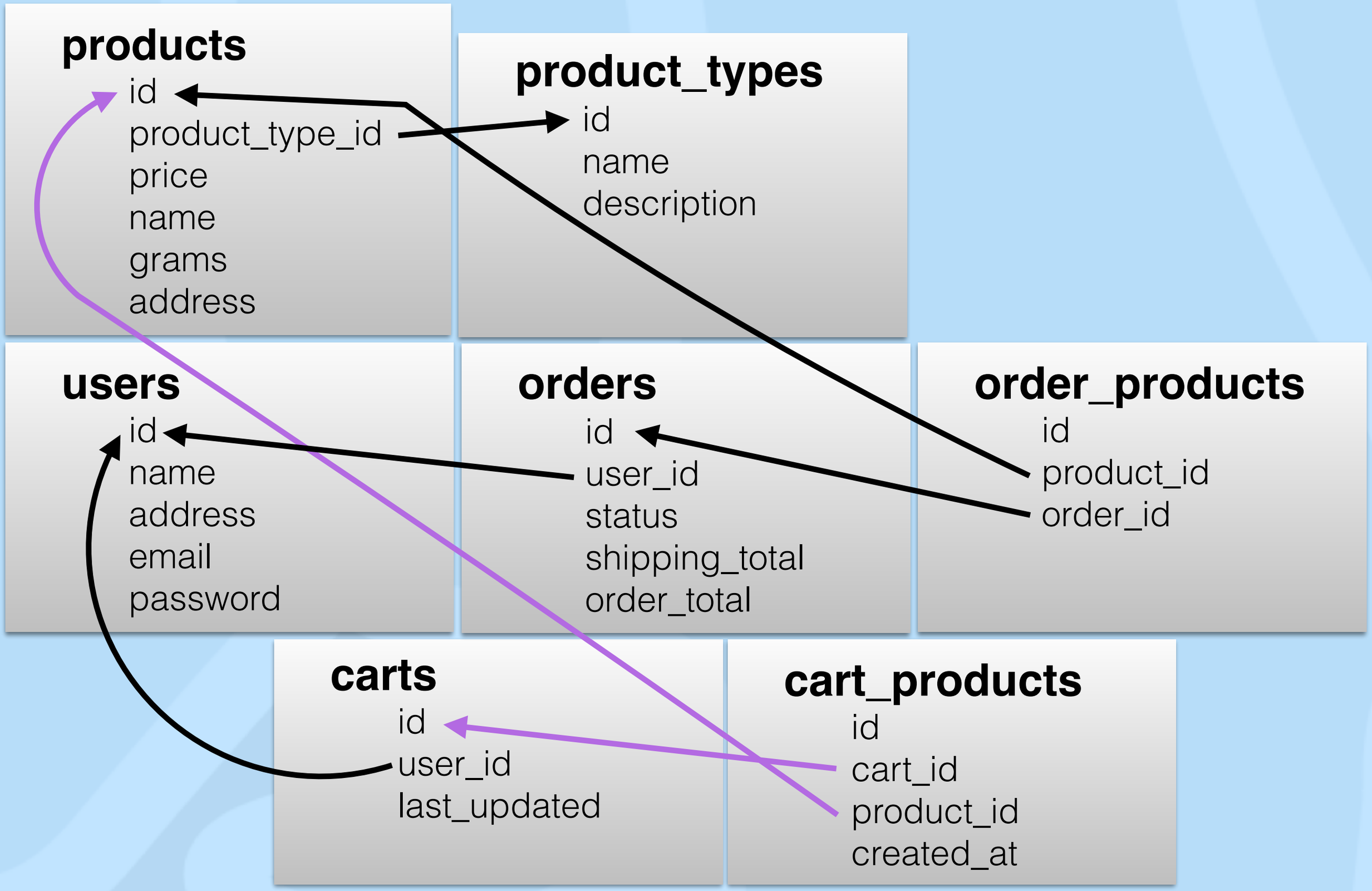
Case Study: Shopping site



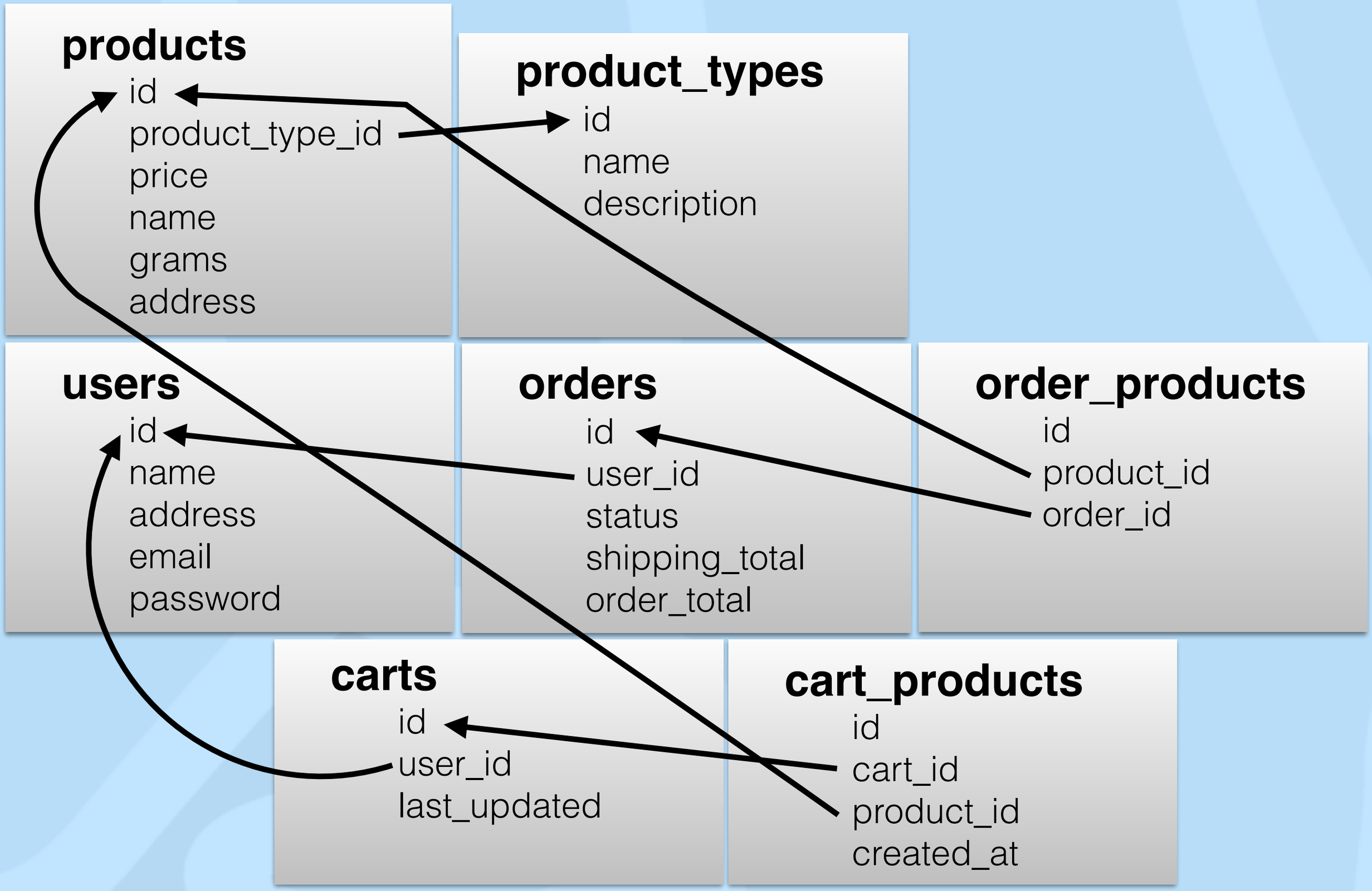
Case Study: Shopping site



Case Study: Shopping site



Case Study: Shopping site



Case Study: Shopping site

products
- id
- product_type_id
- price
- name
- grams
- address

product_types
- id
- name
- description

users
- id
- name
- address
- email
- password

orders
- id
- user_id
- status
- shipping_total
- order_total

order_products
- id
- product_id
- order_id

carts
- id
- user_id
- last_updated

cart_products
- id
- cart_id
- product_id
- created_at

Key Concept Review

- Select Functions
- Casting & data types
- Sets
- Creating tables

Key Concept Review

- select functions
  - distinct and group by
  - math and date functions
  - case statements
  - aliasing

Key Concept Review

- distinct and group by

SELECT **DISTINCT** product_type_id FROM products ORDER BY product_type_id;

SELECT product_type_id FROM products
GROUP BY product_type_id
ORDER BY product_type_id;

Key Concept Review

- math functions: avg, count, sum; date functions
  - SELECT **SUM**(shipping_total) as shipping_total, **COUNT**(*) as total_orders, **AVG**(order_total) as avg_order,
    EXTRACT(MONTH FROM created_at) as month,
    EXTRACT(YEAR FROM created_at) as year
    FROM orders
    GROUP BY year, month
    ORDER BY year, month;

Key Concept Review

- select functions
- aliasing - setting name for columns, tables and values to be referred to more
    - ex: SELECT o.shipping_total AS total FROM orders o;

Key Concept Review

- select functions
  - aliasing - setting name for columns, tables and values to be referred to more
  - ex: SELECT o.shipping_total AS total
    FROM orders o
    JOIN order_products op on op.order_id = o.id
    JOIN cart_products cp on cp.product_id = op.product_id

Casting: Data Types

- Casting values - common 'types'
  - Text - a 'string' of characters, words/characters (i.e. from a keyboard)
  - Integer - whole number
  - Decimal - Float (i.e. floating point)
  - Date - a calendar date
  - Timestamp - a date with specific time, can also have a specific timezone

Casting

- Casting values
  - 2 methods:
    - **CAST**(<column/result> **AS** <type>)
    - <column/result>**::<type>**

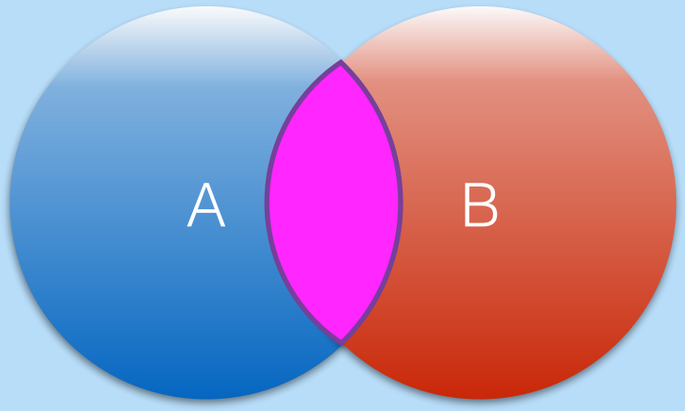
- SELECT
  CAST(o.shipping_total **AS** integer) as int_total,
  o.shipping_total**::INTEGER** as int_total_v2
  FROM orders o;

Key Concept Review

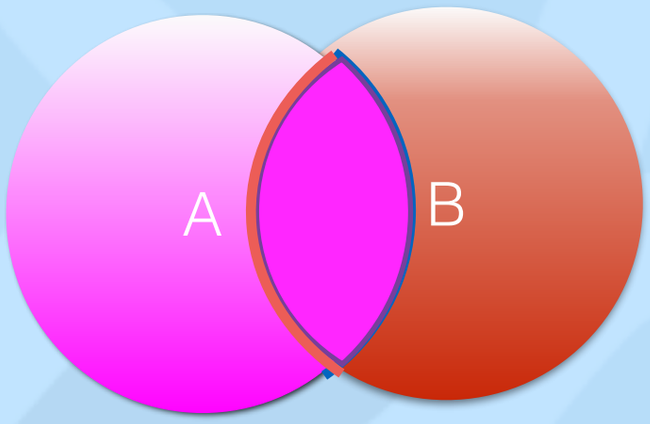
- Sets
- Outer vs Inner
- Left vs Right
- Union, Intersect Except

Key Concept Review

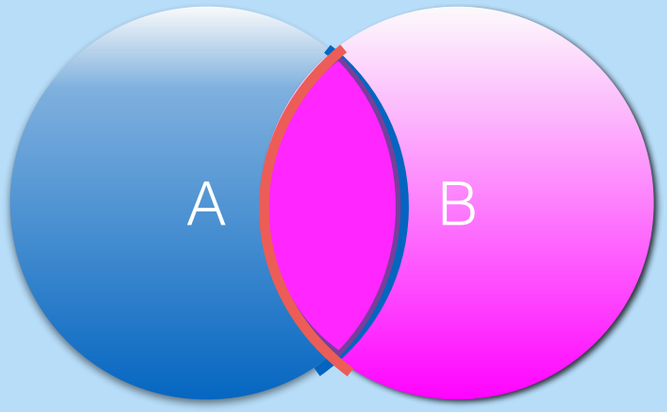
- Sets overview
  - Outer vs Inner
  - Left vs Right



A INNER JOIN B



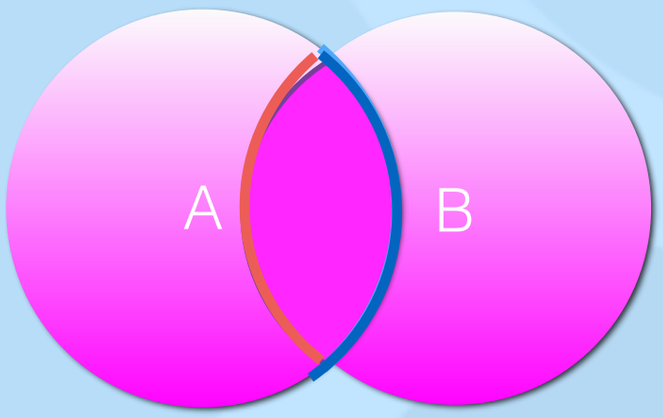
A LEFT OUTER B



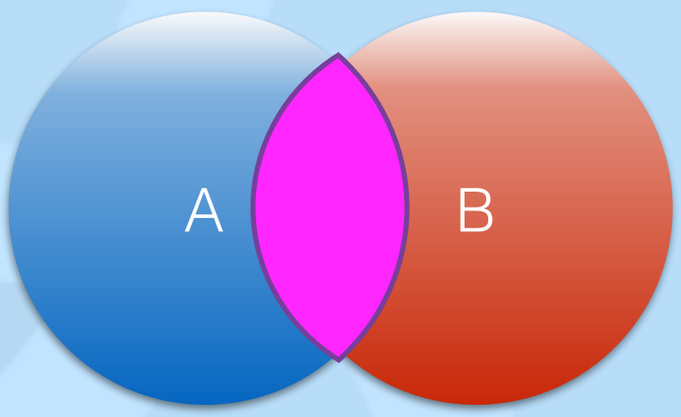
A RIGHT OUTER B

Key Concept Review

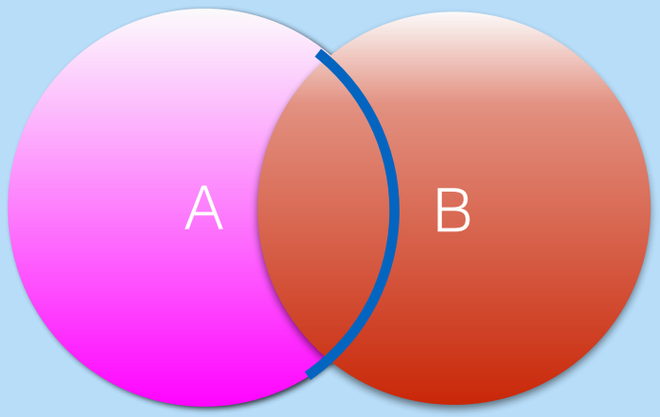
- Sets overview
  - Union, Intersect Except



A union B



A intersect B

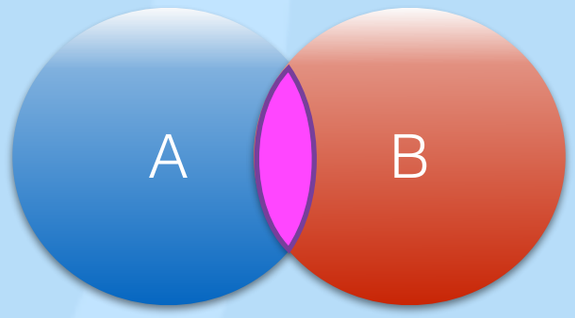


A except B

Inner Join

A = Users

id	integer
name	text
address	text
created_at	date



B = Orders

id	integer
user_id	integer
order_total	float
address	text

SELECT * FROM
users INNER JOIN orders ON users.id = orders.user_id

AA

id	name	address	created_at
1	John Nob…	123 nowhe…	2016-03-19
2	John Som…	234 a pla…	2016-05-23
3	John Doe	345 Doesv…	2016-05-22

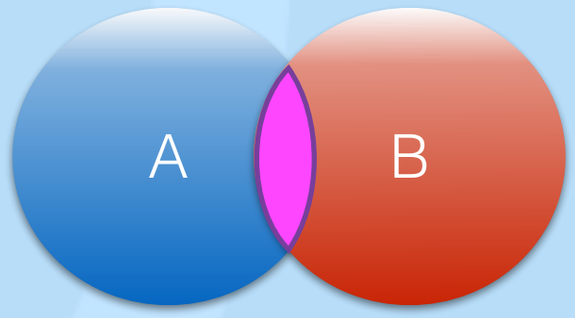
BB1

id	user_id	order_total	address
8	1	147.33	678 nowhe…
23	2	43.27	372 other…
52	3	57.9	345 Doesv…

Inner Join

A = Users

id	integer
name	text
address	text
created_at	date



B = Orders

id	integer
user_id	integer
order_total	float
address	text

SELECT * FROM
users, orders
WHERE users.id = orders.user_id

A

id	name	address	created_at
1	John Nob…	123 nowhe…	2016-03-19
2	John Som…	234 a pla…	2016-05-23
3	John Doe	345 Doesv…	2016-05-22

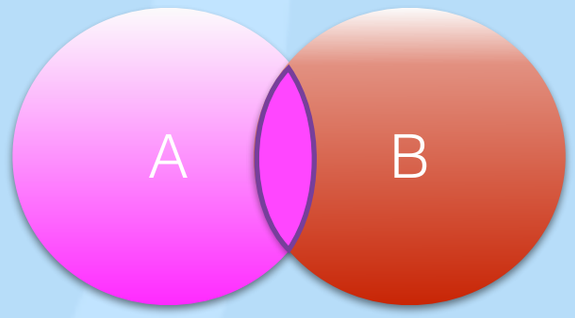
B-1

id	user_id	order_total	address
8	1	147.33	678 nowhe…
23	2	43.27	372 other…
52	3	57.9	345 Doesv…

Left Outer Join

A = Users

id	integer
name	text
address	text
created_at	date



B = Orders

id	integer
user_id	integer
order_total	float
address	text

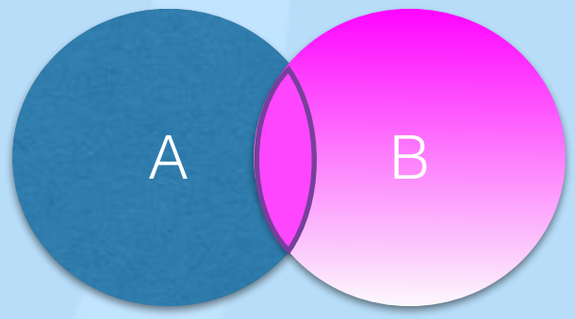
SELECT * FROM users
LEFT OUTER JOIN orders ON users.id = orders.user_id

A				B			
id	name	address	created_at	id	user_	order_total	address
1	John Nob…	123 nowhe…	2016-03-19	8	**1**	147.33	678 nowhe…
2	John Som…	234 a pla…	2016-05-23	23	**2**	43.27	372 other…
3	John Doe	345 Doesv…	2016-05-22	52	**3**	57.9	345 Doesv…
14	Jane Smith	502 downt…	2017-10-21	null	**null**	null	null
17	Janet Farm	402 lastt…	2018-01-24	null	**null**	null	null

Right Join

A = Users

id	integer
name	text
address	text
created_at	date



B = Orders

id	integer
user_id	integer
order_total	float
address	text

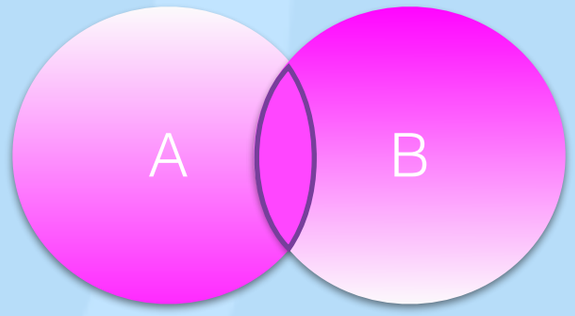
SELECT * FROM users
RIGHT OUTER JOIN orders ON users.id = orders.user_id

A				B			
id	name	address	created_at	id	user_id	order_total	address
1	John Nob…	123 nowhe…	2016-03-19	8	1	147.33	678 nowhe…
2	John Som…	234 a pla…	2016-05-23	23	2	43.27	372 other…
3	John Doe	345 Doesv…	2016-05-22	52	3	57.9	345 Doesv…

Full Join

A = Users

id	integer
name	text
address	text
created_at	date



B = Orders

id	integer
user_id	integer
order_total	float
address	text

Users FULL OUTER JOIN orders ON users.address = orders.address

A				B			
id	name	address	created_at	id	user_	order_total	address
1	John Nob…	123	2016-03-19	null	null	null	null
14	Jane Smith	502	2017-10-21	null	null	null	null
2	John Som…	234 a	2016-05-23	35	2	113.99	234 a pla…
3	John Doe	345	2016-05-22	52	3	57.9	345 Doesv…
4	John Moe	456	2016-05-22	61	4	74.99	456 somew…
null	null	null	null	109	9	19.24	36127 av…
null	null	null	null	20	1	64.23	678 nowh…

Advanced Functions

- Having
- String Functions
- Random

Having

- Having allows you to filter the results on the result of the rest of the query
- Useful for filtering results of aggregate and math results only
    - Won't work with case statement results for example

Having

- Having allows you to filter the results on the RESULT of the rest of the query

- WHERE
  Select product_id, count(*) from order_products
  WHERE count(*) > 2
  GROUP BY product_id
  ORDER BY count desc

- HAVING
  Select product_id, count(*) from order_products
  GROUP BY product_id
  HAVING count(*) > 2
  ORDER BY count(*) desc

Advanced functions con't.

- String Functions
  - ||, concatenate

Advanced functions con't.

- String Functions
  - ||, concatenate - push two text values together

Advanced functions con't.

- String Functions
- ||, concatenate - push two text values together

  Example: select 'add text: ' || name from products;

Concatenate: Quick Exercise

- String Functions
  - ||, concatenate - push two text values together

    Example: select 'add text' || name from products;

- EXERCISE: Return all users' full address with their name and address concatenated together.

Concatenate: Quick Exercise

- String Functions
  - ||, concatenate - push two text values together

    Example: select 'add text' || name from products;

- EXERCISE: Return all users' full address with their name and address concatenated together.

  SELECT name || ' ' || address from users;

Advanced functions con't.

- Random

Advanced functions con't.

- Random - returns a random value between 0 and 1
  EX: Select random()

Advanced functions con't.

- Random - returns a random value between 0 and 9
  EX: Select random() * 10

Advanced functions con't.

- Random - returns a random value between 0 and 1
  EX: Select random() * 10

- Getting whole numbers
  - round - round to the nearest whole number
  - floor - round the down to the nearest whole number less than the value
  - ceiling - round up to the nearest whole number above the value

Advanced functions con't.

ADD SPECIFIC NUMBER C
RANDOM W/OUT * 10

- Random - returns a random value between 0 and 1
  EX: Select random() * 10

- Getting whole numbers
  - round - round to the nearest whole number
  - floor - round the down to the nearest whole number less than the value
  - ceiling - round up to the nearest whole number above the value

- SELECT random_value,
  round(random_value),
  floor(random_value),
  ceiling(random_value)
  FROM (select random() * 10 AS random_value) random_query

Random: Quick Exercise

- Random - returns a random value between 0 and 1
  EX: Select random()

- Write a query to return a random whole number between 5 and 10

Random: Quick Exercise

- Random - returns a random value between 0 and 1
  EX: Select random()
- Write a query to return a random whole number between 5 and 10
  (HINT: SELECT 5 + <random whole # from 0-5>)

Random: Quick Exercise

- Random - returns a random value between 0 and 1
  EX: Select random()

- Write a query to return a random value between 5 and 10

  SELECT 5 + round(random() * 5)

PgAdmin4 Overview

MOVE ME!

- PG Admin4 intro

Q & A

Exercise: Advanced functions

1. Write a query to return each product name with a random whole number between 1 and 10 concatenated to product name
(HINT: Use products table;
|| and random() functions)

Exercise: Advanced functions

1. Write a query to return each product name with a random number between 1 and 10 concatenated to product name
   (HINT: Use products table, || and RANDOM)

```
select name || ' ' || (1 + round(random() * 9) )
from products;

select name || ' ' || (ceiling(random() * 10) )
from products;
```

Performance checks & PGAdmin 4

Performance checks & PGAdmin 4

- Explain

Performance checks & PGAdmin 4

- Explain
  - Nested
  - Hash
    - Hash full
  - sort
  - aggregate

Q & A

10 Min Break

Sub-select, With and temporary tables

- Help to organize your data
- Reduce cognitive load
- Separate process components
- Improve performance

Sub-select Commands

Sub-select Commands

- running a query within '(' and ')' and utilizing those results within a query

Sub-select Commands

- running a query within '(' and ')' and utilizing those results within a query

- EX: Round a single random value:

```
SELECT random_value,
round(random_value),
floor(random_value),
ceiling(random_value)
FROM  (select random() * 10 AS random_value )
random_query
```

Sub-select Commands

- running a query within '(' and ')' and utilizing those results within a query

- EX: Get all products with product types that are 'clothing' using a sub-select.

Sub-select Commands

- running a query within (and) and utilizing those results within a query

- EX: Get all products with product types that are 'clothing' using a sub-select.

- Step 1: create a query to get all the product type IDs that are clothing.

Sub-select Commands

- **Sub-select** with **IN**

  SELECT * FROM products
  WHERE product_type_id IN (select id from product_types where name ilike '%clothing');

Sub-select Commands

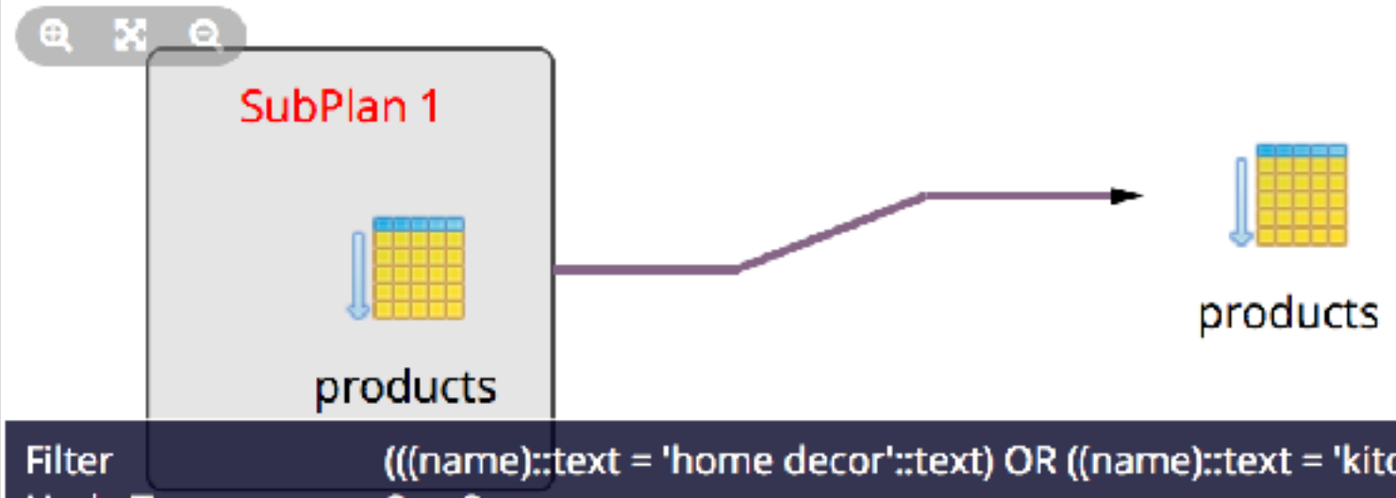
- **Where** with And, **Or**, **In**
  return products that are either heavy or products for the home

```
SELECT * FROM products
WHERE grams > 10
OR product_type_id IN (select id from products
where name = 'home decor' OR name = 'kitchen
and dining');
```

Performance checks & PGAdmin 4

```
1  SELECT * FROM products
2  WHERE grams > 10
3  OR product_type_id IN
4  (select id from products where name = 'home decor' OR name = 'kitchen and dining')
```

Data Output | Explain | Messages | Query History



Filter	(((name)::text = 'home decor'::text) OR ((name)::text = 'kitchen and dining'::text))
Node Type	Seq Scan
Relation Name	products
Alias	products_1
Parallel Aware	false
Parent Relationship	SubPlan
Subplan Name	SubPlan 1

Sub-select Commands

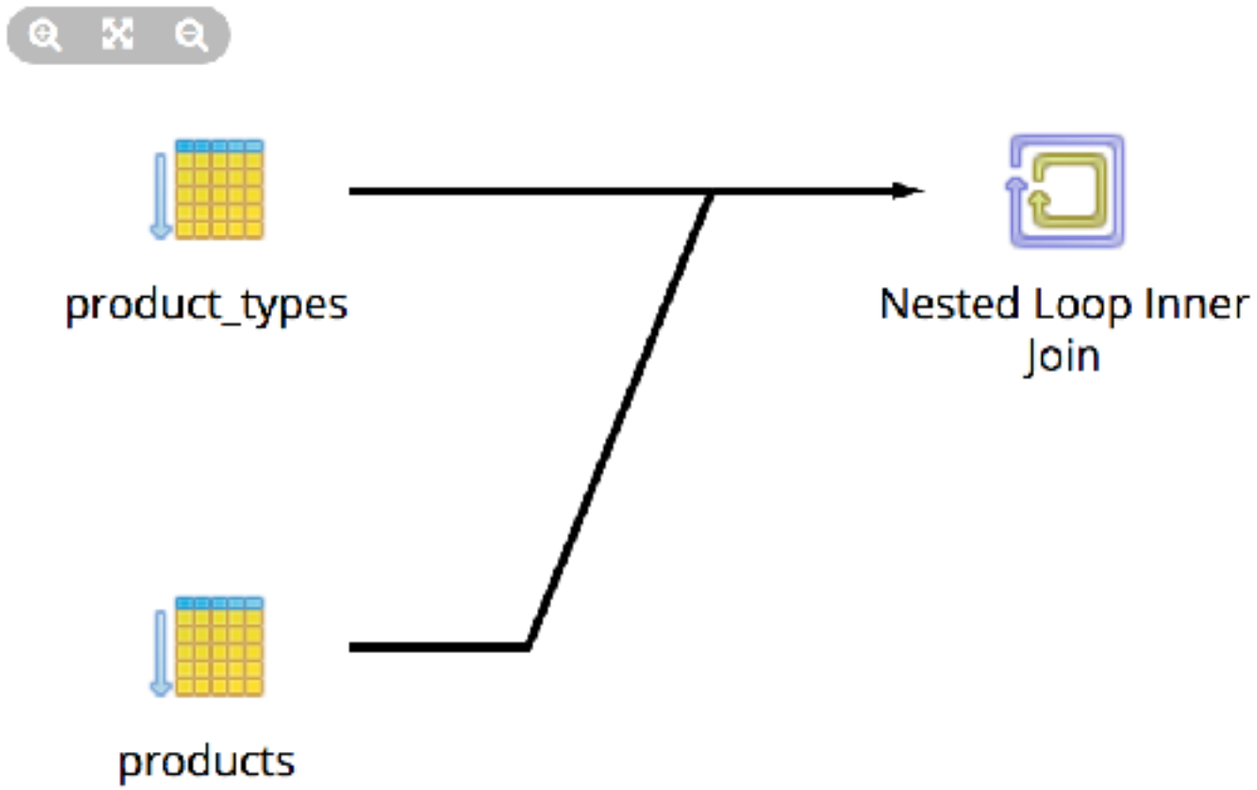
- **Sub-select** with **JOINs**

  SELECT * FROM products p
  JOIN (select * from product_types where name ilike '%clothing') p_types ON p_types.id = p.product_type_id;

Performance checks & PGAdmin 4

```
1  SELECT * FROM products p
2  JOIN (select * from product_types
3        where name ilike '%clothing') p_types ON p_types.id = p.product_type_id;
```

Data Output | Explain | Messages | Query History



WHY DOES SUB-SELECT NOT HAVE AN EXERCISE???

With

- **WITH (a.k.a. common table expressions (CTE))**

  Temporary view of the data only available in the scope of the query it runs in

- Named so can be interacted with in queries as tables
- fundamentally equivalent to sub-select statements (also CTEs)
- FORMAT:
  - WITH use_query_as_table as (<query from database>)

    <Query utilizing temp_view>;

With

- **WITH**

  Temporary view of the data only available in the scope of the query it runs in

- Named so can be interacted with in queries as tables
- FORMAT:
  - WITH use_query_as_table as (<query from database>)

    <Query utilizing temp_view>;

With

- **WITH**

  Temporary view of the data only available in the scope of the query it runs in

- Named so can be interacted with in queries as tables

- FORMAT:

  - WITH use_query_as_table as (<query from database>)
    <Query utilizing temp_view>;

  - **Example:**
    WITH clothing_product_types as (select id from product_types where name ilike '%clothing')

    SELECT * FROM products
    WHERE product_type_id IN (select id from clothing_product_types);

With: Quick Exercise

- Use WITH to create a set of user_ids that have updated their cart this year;
  select all user information for user who have updated their cart this year
  (HINT: with will select from carts, the select statement will select from users)

  format:
  WITH use_query_as_table as (<query from database>)
  <Query utilizing temp_view>;

With: Quick Exercise

- Use WITH to create a set of user_ids that have updated their cart this year; select all user information for user who have updated their cart this year
(HINT: with will select from carts
last_updated > '2018-01-01',
the select statement will select from users)

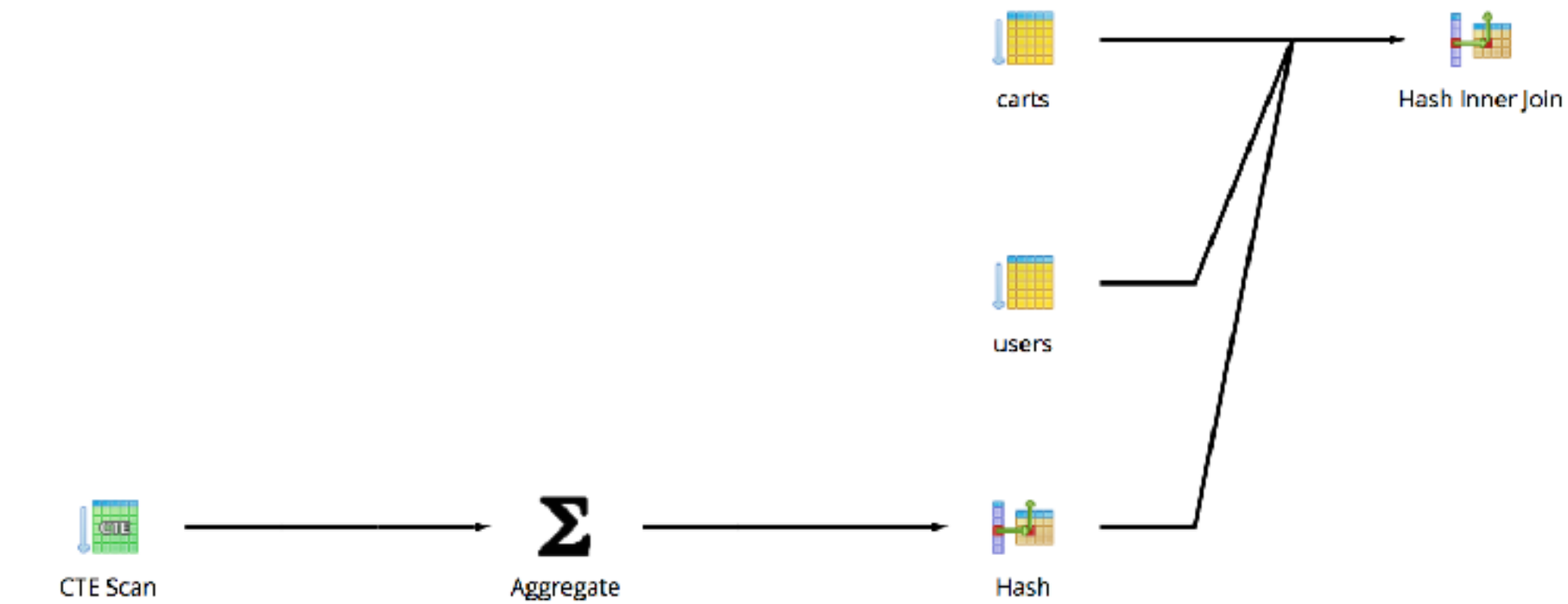
```
WITH user_updated_carts AS (SELECT user_Id from
carts where last_updated > '2018-01-01')

SELECT * FROM users
WHERE id IN (select user_id from
                    user_updated_carts)
```

Performance checks & PGAdmin 4

```
1  WITH user_updated_carts AS (SELECT user_Id from carts
2  where last_updated > '2018-01-01')
3
4  SELECT * FROM users
5  WHERE id IN (select user_id from
6                      user_updated_carts)
```

Data Output | Explain | Messages | Query History



With performance

- **With** and **IN**

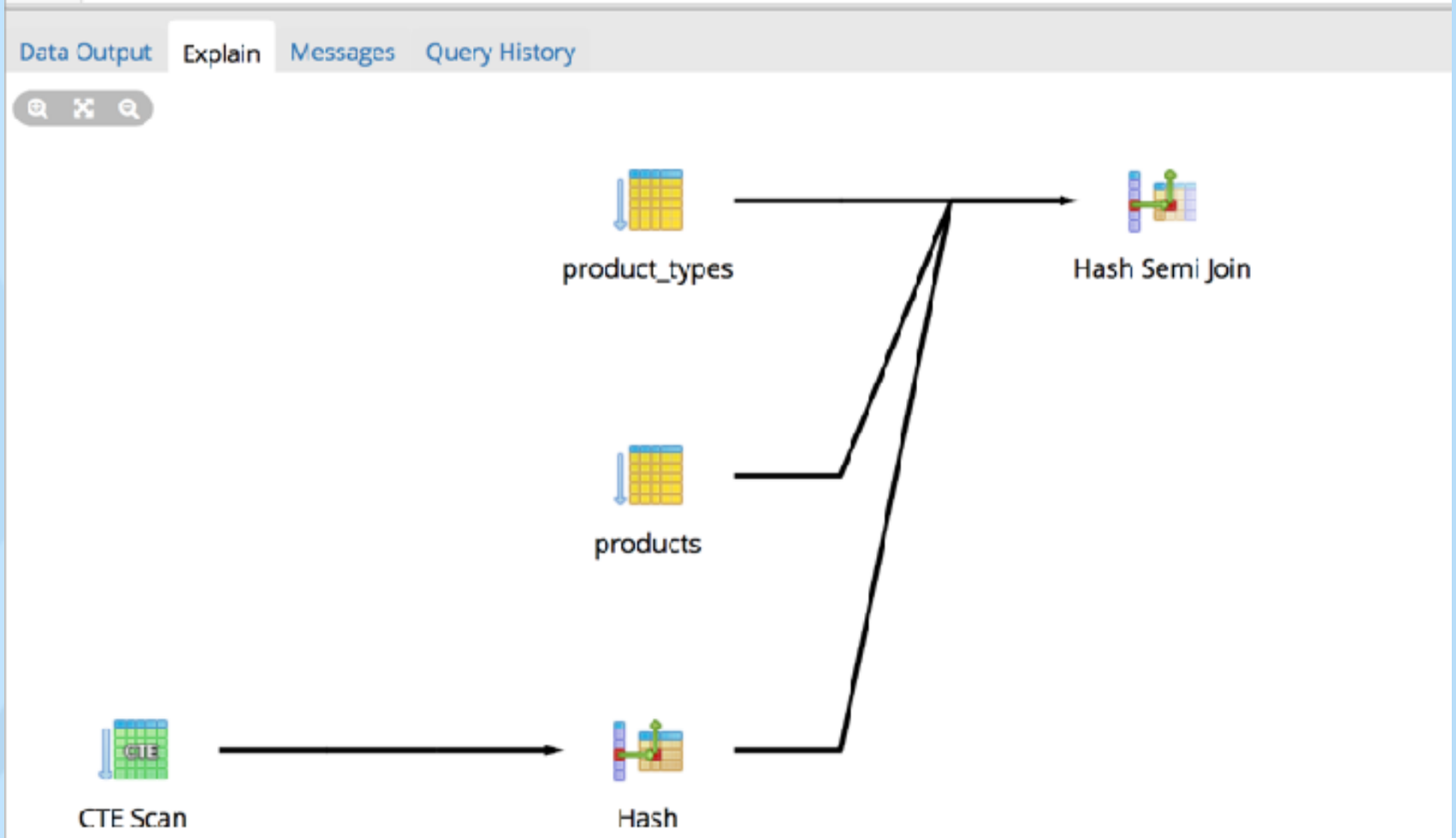
  WITH clothing_product_types as (select id from product_types where name ilike '%clothing')

  SELECT * FROM products
  WHERE product_type_id IN (select id from clothing_product_types);

Performance checks & PGAdmin 4

```
1  WITH clothing_product_types as (select id from product_types where name ilike '%clothing')
2
3  SELECT * FROM products
4  WHERE product_type_id IN (select id from clothing_product_types);
```

Data Output | Explain | Messages | Query History



With

- **WITH** and **JOIN**

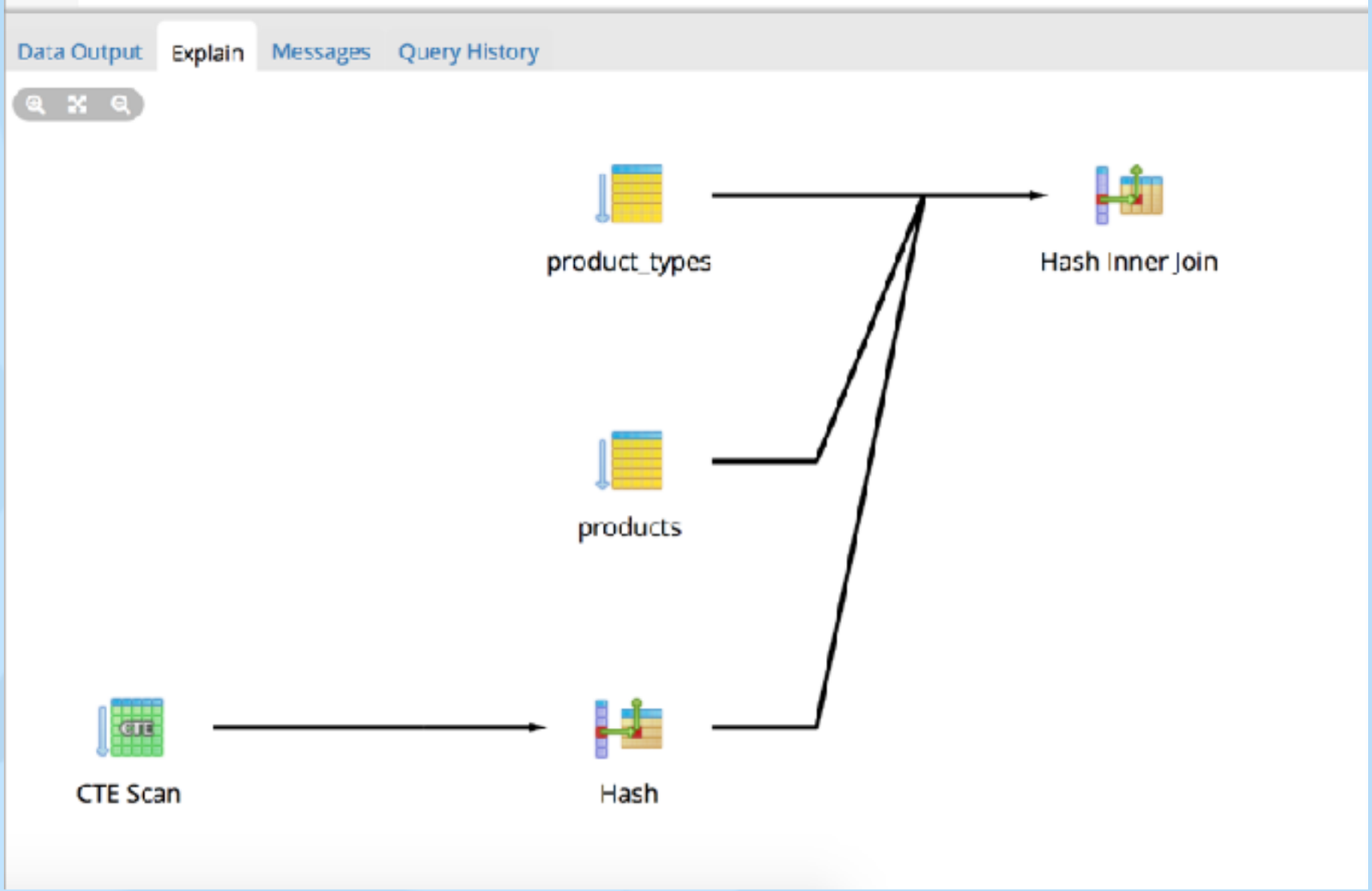
```
WITH clothing_product_types as (select * from
product_types where name ilike '%clothing')

SELECT * FROM products
JOIN clothing_product_types p_types ON
p_types.id = p.product_type_id;
```

Performance checks & PGAdmin 4

```
1  WITH clothing_product_types as (select * from product_types where name ilike '%clothing')
2
3  SELECT * FROM products
4  JOIN clothing_product_types p_types ON p_types.id = products.product_type_id;
```

Data Output | Explain | Messages | Query History



Q & A

Temporary Tables

Temporary Tables

- Temporary Tables
  - provide similar simplification to complex queries,
  - Performance optimization control
  - Format: CREATE TEMPORARY TABLE <table_name> as (<query>); SELECT * FROM <table_name>;

Temporary Tables

- Based on:
  WITH clothing_product_types as (select * from product_types where name ilike '%clothing')

  SELECT * FROM products
  JOIN clothing_product_types p_types ON p_types.id = p.product_type_id;

  FORMAT - CREATE TEMPORARY TABLE <table_name> as (<query>); SELECT * FROM <table_name>;

Temporary Tables

- Based on:
  WITH clothing_product_types as (select * from product_types where name ilike '%clothing')

  SELECT * FROM products
  JOIN clothing_product_types p_types ON p_types.id = p.product_type_id;

FORMAT - CREATE TEMPORARY TABLE <table_name> as (<query>); SELECT * FROM <table_name>;

CREATE TEMPORARY TABLE clothing_product_types as (select * from product_types where name ilike '%clothing');

SELECT * FROM products
JOIN clothing_product_types p_types ON p_types.id = product_type_id;

Temporary Tables: Quick Exercise

- Based on the with statement from earlier. Use a TEMPORARY TABLE to hold all users/user_ids that have updated their cart this year. Select all user information from the users table for those users:
  Based ON: WITH user_updated_carts AS (SELECT user_Id from carts where last_updated > '2018-01-01')
  SELECT * FROM users where user_id in (user_updated_carts);

  FORMAT - CREATE TEMPORARY TABLE <table_name> as (<query>); SELECT * FROM <table_name>;

Temporary Tables: Quick Exercise

- Based on the with statement from earlier. Use a TEMPORARY TABLE to hold all users/user_ids that have updated their cart this year. Select all user information from the users table for those users:

Based ON: WITH user_updated_carts AS (SELECT user_Id from carts where last_updated > '2018-01-01')

CREATE TEMPORARY TABLE user_updated_carts as (select * from carts where last_updated >= '2018-01-01');

```
SELECT * FROM users
WHERE id IN (select user_id from
                    user_updated_carts);
```

Performance checks & PGAdmin 4

```
1  CREATE TEMPORARY TABLE clothing_product_types as (select * from product_types
2                                                    where name ilike '%clothing')
3  ;
4  EXPLAIN
5  SELECT * FROM products
6  JOIN clothing_product_types p_types ON p_types.id = products.product_type_id;
```

Data Output | Explain | Messages | Query History

	QUERY PLAN text
1	Hash Join (cost=2.24..19.92 rows=88 width=257)
2	Hash Cond: (p_types.id = products.product_type_id)
3	-> Seq Scan on clothing_product_types p_types (cost=0.00..13.20 rows=320 width=222)
4	-> Hash (cost=1.55..1.55 rows=55 width=35)
5	-> Seq Scan on products (cost=0.00..1.55 rows=55 width=35)

Q & A

With, Sub-select, and Temporary Tables Exercises

- Write a query to randomly assign each user a 'bucket' that is a value between 1 and 3. Return the total number of orders for users in each bucket.
  Use each of the following to store this result as user_buckets
  (SELECT id as user_id, 1 + round(random() * 2) AS bucket FROM users)

1) Sub-select - Using '(' and ')'
2) WITH
3) TEMPORARY TABLE

With, Sub-select, and Temporary Tables Exercises

- Write a query to randomly assign each user a 'bucket' that is a value between 1 and 3. Return the total number of orders for users in each bucket.
  Use each of the following to store this result as user_buckets:
  (SELECT id as user_id, 1 + round(random() * 2) AS bucket FROM users)

HINT: Join the results and group by user_ID:
JOIN orders ON orders.user_id = user_buckets.user_id

1) Sub-select - Using '(' and ')'
2) WITH
3) TEMPORARY TABLE

With, Sub-select, and Temporary Tables Exercises

- Write a query to randomly assign each user a 'bucket' that is a value between 1 and 3. Return the total number of orders for users in each bucket.
  Use each of the following to store this result as user_buckets:
  (SELECT id as user_id, 1 + round(random() * 2) AS bucket FROM users)

HINT: Join the results and group by user_ID:
SELECT bucket, count(*) as total_orders…
…
JOIN orders ON orders.user_id = user_buckets.user_id
GROUP BY bucket

1) Sub-select - Using '(' and ')'
2) WITH
3) TEMPORARY TABLE

With, Sub-select, and Temporary Tables Exercises

- Write a query to randomly assign each user a 'bucket' that is a value between 1 and 3. Return the total number of orders for users in each bucket.
  Use each of the following to store this result as user_buckets
  (SELECT id, 1 + round(random() * 2) AS bucket FROM users)

HINT: Join the results and group by user_ID:
JOIN orders ON orders.user_id = user_buckets.user_id

1) Sub-select - Using '(' and ')'
SELECT bucket, count(orders.id) FROM orders
*JOIN (**Select id, 1 + round(random() * 2) as bucket from users**) user_buckets on user_buckets.id = orders.user_id*
GROUP BY BUCKET

With, Sub-select, and Temporary Tables Exercises

- Write a query to randomly assign each user a 'bucket' that is a value between 1 and 3. Return the total number of orders for users in each bucket.
  Use each of the following to store this result as user_buckets
  (SELECT id, 1 + round(random() * 2) AS bucket FROM users)

HINT: Join the results and group by user_ID:
JOIN orders ON orders.user_id = user_buckets.user_id

2) WITH
WITH user_buckets **AS** (***Select id, 1 + round(random() * 2) as bucket from users****)*

SELECT bucket, count(orders.id) FROM orders
JOIN user_buckets on user_buckets.id = orders.user_id
GROUP BY BUCKET

With, Sub-select, and Temporary Tables Exercises

- Write a query to randomly assign each user a 'bucket' that is a value between 1 and 3. Return the total number of orders for users in each bucket.
  Use each of the following to store this result as user_buckets
  (SELECT id, 1 + round(random() * 2) AS bucket FROM users)

HINT: Join the results and group by user_ID:
JOIN orders ON orders.user_id = user_buckets.user_id

3) Temporary Table
CREATE TEMPORARY TABLE user_buckets **AS** (***Select id, 1 + round(random() * 2) as bucket from users***);

SELECT bucket, count(orders.id) FROM orders
JOIN user_buckets on user_buckets.id = orders.user_id
GROUP BY BUCKET

Q & A

10 Min Break

Views

Views

- Persisted outside of a single session or query

Views

- Persisted outside of a single session or query
- Reflect updates to tables used by views - query ran each time view is referenced

Views

- Persisted outside of a single session or query
- Reflect updates to tables used by views - query ran each time view is referenced
- Marterialized Views - Store the results on disk

Views

- Persisted outside of a single session or query
- Reflect updates to tables used by views - query ran each time view is referenced
- Marterialized Views - Store the results on disk
- Can also be temporary

Views

- Common Uses
  - Summarize/consolidate data
  - Re-structure data to be easier to consume/use
  - Utilize separate permissions to limit the data some users have access to

Creating Views

- Create

Creating Views

- Create

CREATE VIEW <view_name> AS <query>

Creating Views

- Create
  CREATE VIEW <view_name>
  (<col1_name>, <col2_name>,…)
  AS <query>

Creating Views

- CREATE VIEW <view_name> AS <query>

Ex: CREATE VIEW clothing
AS SELECT id as
product_type_id,
name
FROM product_types
WHERE name ilike '%clothing'

- CREATE VIEW <view_name> (<col1_name>, <col2_name>,…) AS <query>

Ex: CREATE VIEW clothing
(product_type_id,name)
AS SELECT id, name
FROM product_types
WHERE name ilike '%clothing'

Creating Views

- Create
- Create or Replace
    - Replace any existing views with the same name if they exist

Views

- CREATE VIEW <view_name>
  (<col1_name>, <col2_name>,…) -- optional
  AS <query>

- Create or Replace View
  CREATE **OR REPLACE** VIEW <view_name>
  AS <query>

Creating Views

- CREATE **OR REPLACE** VIEW <view_name> *(<col1_name>, <col2_name>,…) -- optional*
  AS <query>

Ex: CREATE OR REPLACE VIEW clothing
AS SELECT id as product_type_id,
name
FROM product_types
WHERE name ilike '%clothing'

Creating Views

- Create
- Create or Replace
  - Replace any existing views with the same name if they exist
- Temporary Views
  - View only exists during the session

Creating Views

- CREATE OR REPLACE **TEMPORARY** VIEW <view_name>
  (<col1_name>, <col2_name>,…) -- optional
  AS <query>

Ex: CREATE OR REPLACE TEMPORARY VIEW temp_clothing
AS SELECT id as product_type_id,
name
FROM product_types
WHERE name ilike '%clothing'

Creating Views

- CREATE OR REPLACE **TEMPORARY** VIEW temp_clothing
AS SELECT id as product_type_id, name
FROM product_types
WHERE name ilike '%clothing'

Ex:
SELECT * FROM temp_clothing

Creating Views

- CREATE **TEMPORARY** VIEW temp_tech
AS SELECT id as product_type_id, name
FROM product_types
WHERE name ilike '**computer%**'

Ex:
SELECT * FROM temp_tech

Creating Views: Quick Exercise

- Use a VIEW to hold all users/user_ids that have updated their cart this year. Select all user information from the users table for those users:

Based ON: WITH user_updated_carts AS (SELECT user_Id from carts where last_updated > '2018-01-01')

CREATE VIEW <view_name>
(<col1_name>, <col2_name>,…) -- optional
AS <query> ;

SELECT * FROM <view_name>;

Creating Views: Quick Exercise

- Use a VIEW to hold all users/user_ids that have updated their cart this year. Select all user information from the users table for those users:

Based ON: WITH user_updated_carts AS (SELECT user_Id from carts where last_updated > '2018-01-01')

CREATE VIEW user_updated_carts
AS select * from carts where last_updated >= '2018-01-01';

SELECT * from users where id in (SELECT user_id from user_updated_carts)

Q & A

Views Lab

Views Lab

- Group Exercise: Create View

Views Lab

- Group Exercise: Let's create a view to store order_Id and all product information

Views Lab

- Group Exercise: Let's create a view to store order_id, created at date and product_ID with all the product's information

Views Lab

- Group Exercise: Let's create a view to store order_id, order created at date, user who made the order with all the product information included.

```
CREATE VIEW order_product_details AS
SELECT order_id, product_id,
name as product_name,
price, grams, product_type_id,
orders.created_at AS order_date,
user_id
FROM products
JOIN order_products ON order_products.product_id = products.id
JOIN orders on orders.id = order_id;
```

Views Lab

- Group Exercise: Let's create a view to store order_id, order created at date, user who made the order with all the product information included.

SELECT count(*) FROM order_product_details;

SELECT * FROM order_product_details ORDER BY order_id, product_id;

Views Lab

- Now let's add some data to one of the tables the view uses to see what happens!

Views Lab

- Group Exercise: Insert Into Source Tables used in View

  INSERT INTO orders (user_id, status, shipping_total, order_total, address, created_at)
  VALUES
  (13, 'order confirmed', (1 + round(random() *1000)), 10.00, '123 nowhere 12345', now()),
  (14, 'order confirmed', (1 + round(random() *1000)), 20.00, '234 somewhere 23456', now()),
  (15, 'order confirmed', (1 + round(random() *1000)), 30.00, '345 anywhere 34567', now())

Views Lab

- Group Exercise: Insert Into second Source Table so our query returns results!

```
SELECT * FROM orders order by created_at desc;
select count(*) from products
INSERT INTO order_products (order_id, product_id)
VALUES
(160,1),
(160,2),
(160,3),
(161,4),
(161,5),
(161,6),
(162,7),
(162,8),
(162,9),
(162, (1 + round(random() *50)))
```

Views Lab

- Group Exercise: Select from view again and your results are up to date!

Views Lab

- Group Exercise: Select from view again and your results are up to date!

SELECT * FROM order_product_details
ORDER BY order_id desc, product_id;

Views Lab

- Group Exercise: Try to drop a table used in the view

DROP TABLE orders;

Views Lab

- Group Exercise: Try to drop a table used in the view…
  Cant!

  DROP TABLE carts;

  ERROR: cannot drop table carts because other objects depend on it
  DETAIL: constraint cart_products_cart_id_fkey on table cart_products depends on table carts
  view user_updated_carts depends on table carts
  HINT: Use DROP ... CASCADE to drop the dependent objects too.
  SQL state: 2BP01

Views Lab: Cascade

- apply to any and all related or dependent objects (tables, views, etc)

Views Lab: Cascade

- apply to any and all related or dependent objects (tables, views, etc)

- Use case: Have views and need to drop all artifacts -> requires view to be re-created

Views Lab: Cascade

- Group Exercise: First let's create a table and view to work with

Views Lab: Cascade

- Group Exercise: First let's create a table and view to work with

CREATE TABLE carts_cascade AS SELECT *
FROM carts;

Views Lab: Cascade

- Group Exercise: First let's create a table and view to work with

```
CREATE VIEW updated_carts
AS select * from carts_cascade WHERE
last_updated > '2018-01-01';

SELECT * FROM updated_carts;
```

Views Lab: Cascade

- Try to drop the parent table.. should see same error

  DROP TABLE carts_cascade;

Views Lab: Cascade

- Try to drop the parent table.. should see same error

  DROP TABLE carts_cascade;

  ERROR: cannot drop table carts_cascade because other objects depend on it
  DETAIL: view updated_carts depends on table carts_cascade
  HINT: Use DROP ... CASCADE to drop the dependent objects too.
  SQL state: 2BP01

Views Lab: Cascade

- Try to drop the parent table.. should see same error

  DROP TABLE carts_cascade **CASCADE**;

  ERROR: cannot drop table carts_cascade because other objects depend on it
  DETAIL: view updated_carts depends on table carts_cascade
  HINT: Use DROP ... CASCADE to drop the dependent objects too.
  SQL state: 2BP01

Views Lab: Cascade

- Group Exercise:
  DROP TABLE carts_cascade CASCADE;

Views Lab: Cascade

- Group Exercise:

DROP TABLE carts_cascade CASCADE;

NOTICE: drop cascades to view updated_carts
DROP TABLE
Query returned successfully in 73 msec.

Views Lab: Cascade

- Group Exercise: Re-setup the table and view

Views Lab: Cascade

- Group Exercise: Re-setup the table and view
  CREATE TABLE carts_cascade AS SELECT * FROM carts;

  CREATE VIEW updated_carts
  AS select * from carts_cascade WHERE last_updated > '2018-01-01';

  SELECT * FROM updated_carts;

Views Lab

- Work-around: Re-load tables using TRUNCATE instead

Views Lab: Truncate

- Truncate delete all rows in a table
(equivalent to: DELETE FROM <table>)

TRUNCATE <table>;

Views Lab: Truncate

- Truncate delete all rows in a table
(equivalent to: DELETE FROM <table>)

GROUP EXERCISE: TRUNCATE carts_cascade;

Views Lab: Truncate

- Truncate delete all rows in a table
(equivalent to: DELETE FROM <table>)

GROUP EXERCISE: TRUNCATE carts_cascade;

SELECT * FROM updated_carts;

Views Lab: Truncate

- USE CASES
  - Replace table
  - Views
  - Permissions

Q & A

10 Min Break

Indexes

Indexes

- What is an index (how do they work at high level)

Indexes

- What is an index (how do they work at high level)
  - create quick-lookup tables Postgres uses (overhead)
  - Essentially a pointer to data in a table

Indexes

- what is an index (how do they work at high level)

- Over-indexing

Indexes

- Speed up: Select, Joins and Where

- Slows Down: Update and Insert

Indexes

- Primary keys - Automatically constrained to be unique with index

Performance checks & PGAdmin 4

- Query from Views Lab:

```
SELECT order_id, product_id,
name as product_name,
price, grams, product_type_id,
orders.created_at AS order_date,
user_id
FROM products
JOIN order_products ON
order_products.product_id = products.id
JOIN orders on orders.id = order_id;
```

Performance checks & PGAdmin 4

- Query from Views Lab:
  First let's add some random data to the tables:

  INSERT INTO order_products (order_id, product_id)

- SELECT floor(random_to((select max(id) from orders))),floor(random_to((select max(id) from products)))
  FROM generate_series(1,500000);

  INSERT INTO cart_products (cart_id, product_id, created_at)
  SELECT floor(random_to((select max(id) from carts))),floor(random_to((select max(id) from carts))), now()
  FROM generate_series(1,500000);

Indexes

- Add index

Indexes

- Add index
  CREATE INDEX <index_name> ON <table>
  (<table_column>)

Indexes

- Add index

- Vacuum (analyze) is required for the database to setup the pointers/look-up data.

Indexes

- Where to apply indexes

Indexes

- Where to apply indexes
  - nested joins
  - full table scans (on large tables)

Indexes

- Where to apply indexes: what columns?
  - columns being joined on
  - column values used in where clauses

Indexes

- Index for Join clause:

```
CREATE INDEX id_products_index ON products
(id);
CREATE INDEX product_id_cart_prod_idx ON
cart_products (product_id);
```

Indexes

- Index for Where clause:

  CREATE INDEX name_products_index ON products (name)

Review + Q&A

Review + Q&A

- TODO: SETUP LARGE DB, STUDENTS AREN"T DOING , JUST GET IT READY AND GIVE IT TO THEM!