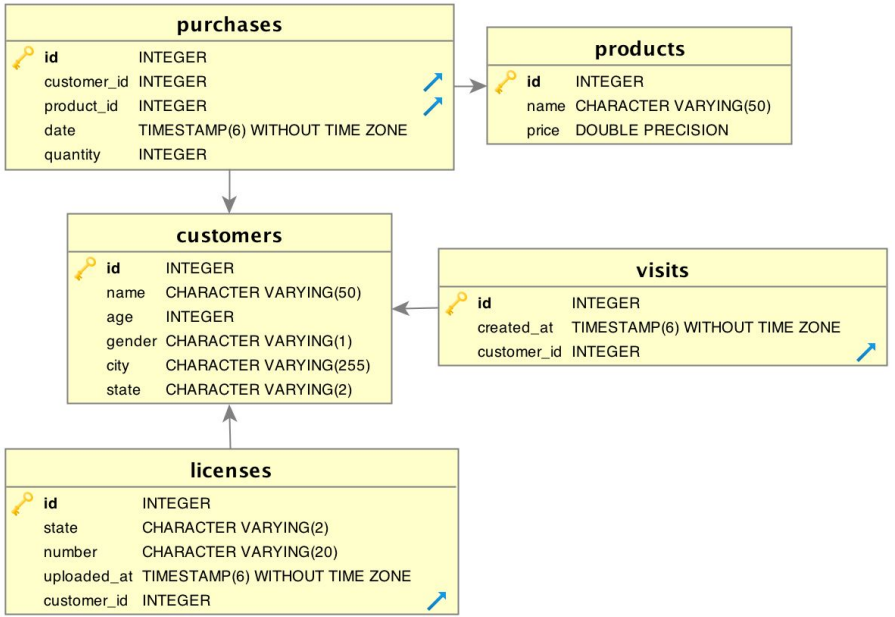
SQL

Relational Databases and the Structured Query Language



SQL

Relational Databases and the Structured Query Language

OBJECTIVES

- **Connect** to a Postgres database
- Design and write **queries** to answer questions using an RDMBS (i.e. Postgres)
- Use **JOIN** to join two tables

Relational Database Management System (RDBMS)

It is a **persistent data storage system**

- survives after the process in which it was created has ended
- is written to non-volatile storage (e.g. hard disk)

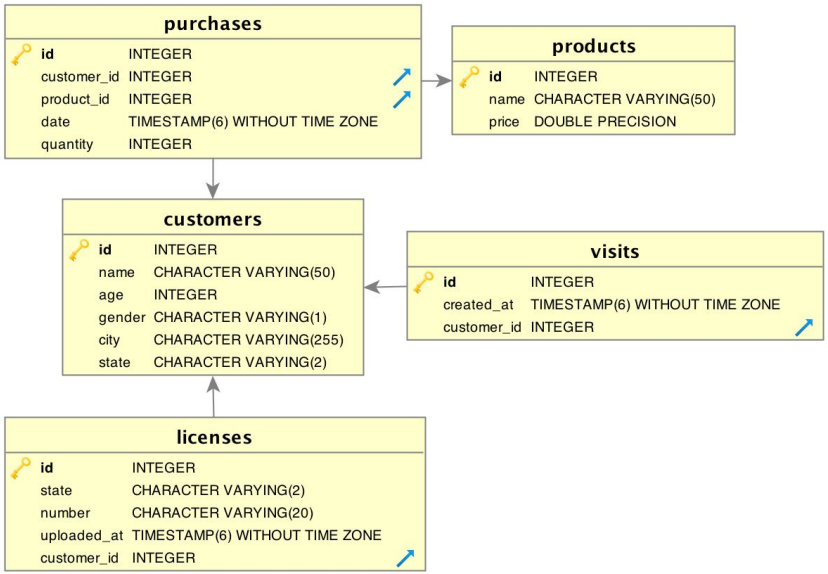
RDBMS was the de facto standard for storing data

- Examples: Oracle, MySQL, SQLServer, Postgres
- With "Big Data", this is beginning to change

Why RDBMS ?

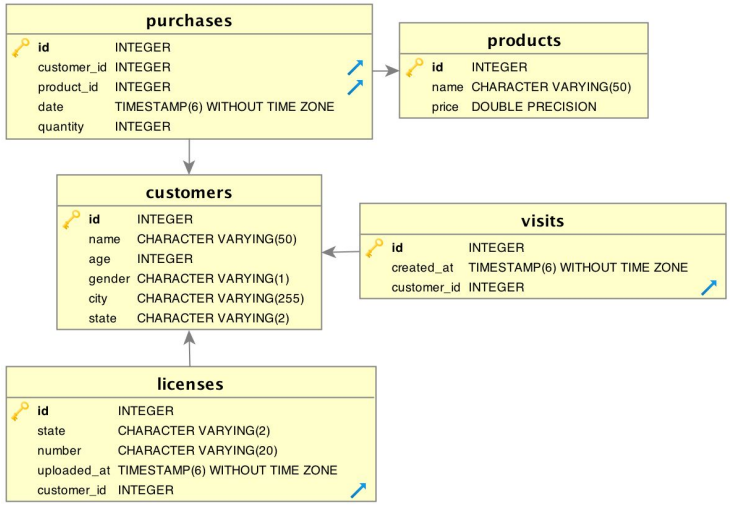
An RDBMS provides the ability to

- **Model** relations in data
- **Query** data and their relations efficiently
- **Maintain** data consistency and integrity



RDBMS Data Model

- **Schema** defines the structure of the data
- The **database** is composed of a number of user-defined **tables**
- Each **table** will have **columns** (aka fields) and **rows** (aka records)
- A column is of a given **data type**
- A row is an entry in a table with data for each column of that table
- Tables are similar to spreadsheets



Employees Table

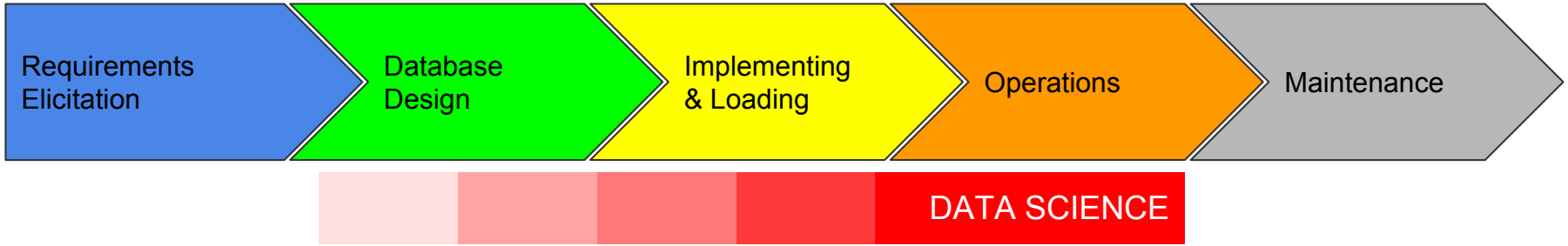
IdNum	LName	FName	JobCode	Salary	Phone
1876	CHIN	JACK	TA1	42400	212/588-5634
1114	GREENWALD	JANICE	ME3	38000	212/588-1092
1556	PENNINGTON	MICHAEL	ME1	29860	718/383-5681
1354	PARKER	MARY	FA3	65800	914/455-2337
1130	WOOD	DEBORAH	PT2	36514	212/587-0013

RDBMS and SQL

- SQL is the language used to query relational databases
- **Almost all RDBMS use SQL** and the syntax and keywords are the same for the most part, across systems
- **SQL is used to interact** with RDBMS, allowing you to create tables, alter tables, insert records, update records, delete records, and query records within and across tables.
- Even non-relational databases like **Hadoop** usually have a SQL-like interface available.

Database Life Cycle (DBLC)

Data Science in the DBLC



DS / Operations: querying, aggregating, model building

DS / Implementing: identifying, cleaning, loading external data sources into a RDBMS

DS / Design: designing database based on needs, think about fields and relationships needed for operations

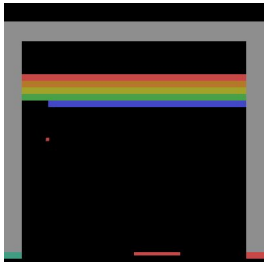
Disclaimer: just IMHO

Question !

As a data scientist, what are the advantages of

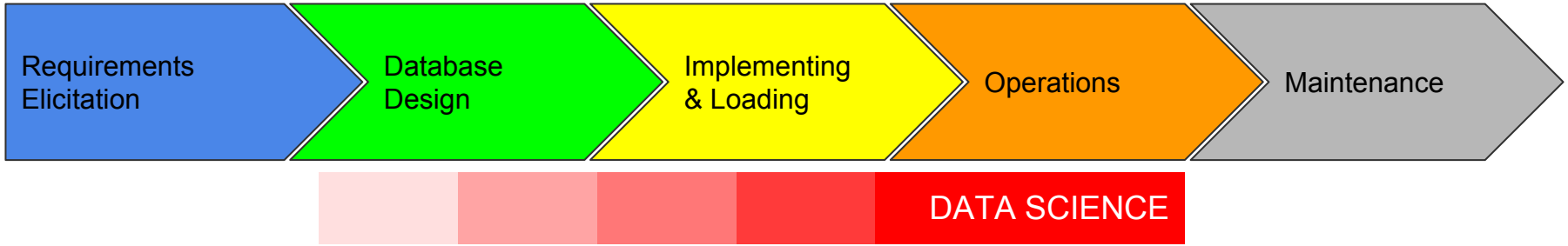
storing, querying, and maintaining data in a SQL database

over curating your own flat files (e.g. csv files) ?



...

Concepts of the day in the DBLC

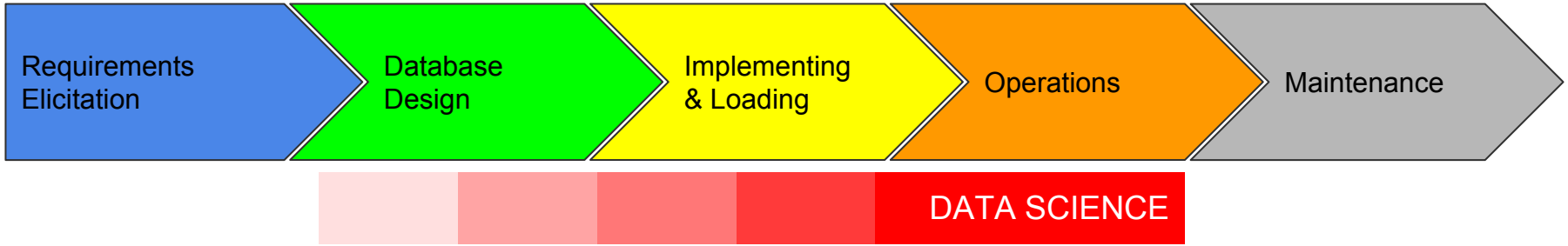


Normalization
Field and relationship definitions

Primary Keys
Foreign Keys
CREATE
DROP
INSERT

SELECT
FROM
WHERE
JOIN
HAVING
DISTINCT
GROUP BY
ORDER BY

Concepts of the day in the DBLC



Normalization
Field and relationship definitions

Primary Keys
Foreign Keys
CREATE
DROP
INSERT

SELECT
FROM
WHERE
JOIN
HAVING
DISTINCT
GROUP BY
ORDER BY

PostGres Basics

Ways to use psql in the shell/term:

`$ psql`	connects to postgres server
`$ psql -U [USERNAME]`	connects with given username
`$ psql [DBNAME]`	connects to a given database
`$ psql < script.sql`	reads file script.sql and send commands to psql

Try it live:

- *Open file sql/lecture_create.sql in atom*
- *Use it to create a "dsilecture" database on your psql server*

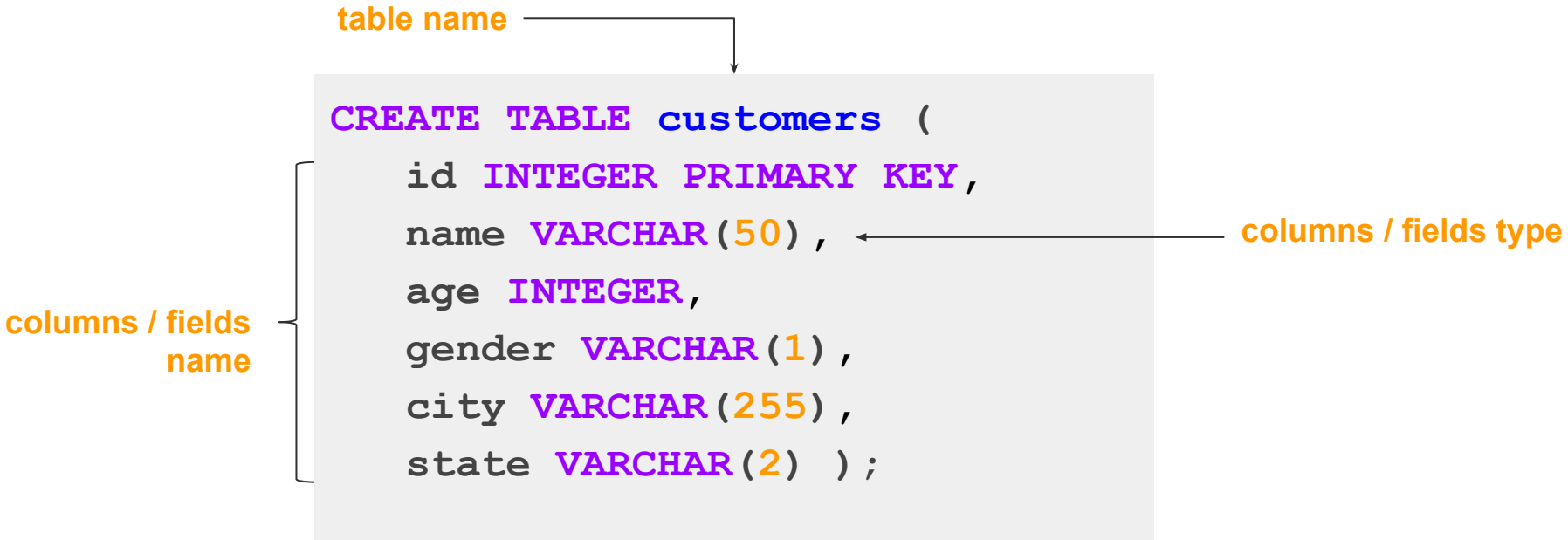
PostGres Basics

Useful psql commands at the prompt [link]:

`# \h`	SQL help
`# \?`	psql commands help
`# \l`	List all the databases
`# \d`	List all the tables of current DB
`# \d table_name`	Describes schema of specific table
`# \connect db_name`	Connects to a database

Try it live: Connect to “dsilecture” and describe schema of table “customer”

Creating a table with a schema



Inserting values in a table

table name

records and their values

```
INSERT INTO products (id, name, price) VALUES
  (1, 'soccer ball', 20.5),
  (2, 'iPod', 200),
  (3, 'headphones', 50);
```

SQL Queries for table creation / maintenance

Creating a table from query:

```
CREATE [TEMPORARY] TABLE table AS <SQL query>;
```

Inserting records in a table:

```
INSERT INTO table [(c1,c2,c3,…)] VALUES (v1,v2,v3,…);
```

Updating records:

```
UPDATE table SET c1=v1,c2=v2,… WHERE cX=vX;
```

Delete records:

```
DELETE FROM table WHERE cX=vX;
```

Change model (add, drop, modify columns):

```
ALTER TABLE table [DROP/ADD/ALTER] column [datatype];
```

Delete a table:

```
DROP TABLE table;
```

Designing a database with keys

Primary Key

A primary key is a special column of a table that uniquely identifies that entry.

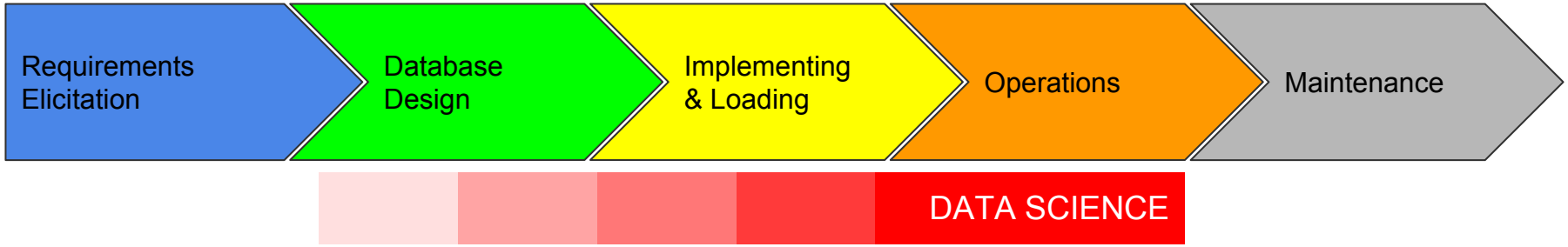
A primary key is not always an integer; it could be a string, timestamp..etc.,

Foreign Keys

Foreign Keys are columns that reference some other entry in the database.

Tip: Keyword REFERENCES enforces existence of foreign key in the other table

Concepts of the day in the DBLC



SQL Syntax

All SQL queries have three main ingredients :

`SELECT`	*What* data do you want?
`FROM`	*Where* do you want to get the data from?
`WHERE`	*Under what* conditions?

SQL is Declarative rather than Imperative. That is, you tell the machine what you want and it (database optimizer) decides how to do it

SQL Queries

Select the columns name, age from the table users.

```
SELECT name, age
FROM customers
```

SQL always returns a table, so the output of the query above is a sub-table of users with 2 columns.
Select name and age for every user in users who live in CA.

```
SELECT name, age
FROM customers
WHERE state = 'CA'
```

SQL Examples

Open file sql/lecture_examples.sql

Run them in psql

JOIN

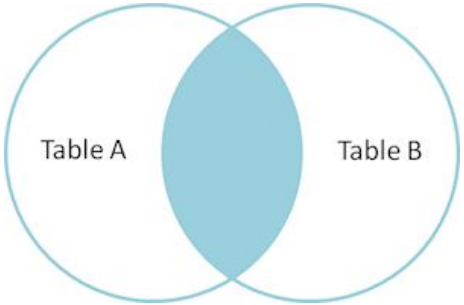
JOIN clause used to query across multiple tables using foreign keys

Every JOIN has two segments:

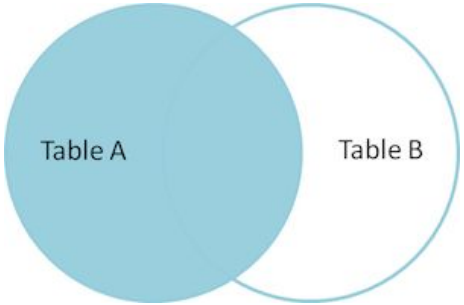
Specifying the tables to JOIN
Specifying the columns to match

JOIN types

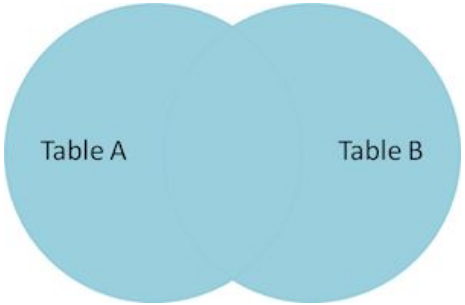
INNER JOIN
discards any entries
that do not have a match
between the tables
based on the given keys.



LEFT OUTER JOIN
keeps all entries
in the left table
regardless of
whether a match is found
in the right table



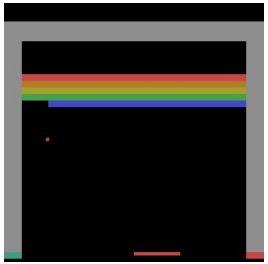
FULL OUTER JOIN
will keep the rows
of both tables
no matter what



copied from *http://blog.codinghorror.com/a-visual-explanation-of-sql-joins/*

Question ! (JOIN the fun)

How many rows would result in
(inner join, left join, right join, full outer join) on `department_id` ?



employee_id	department_id	name	salary
2	1	Jon	40000
7	1	Linda	50000
12	2	Ashley	15000
1	0	Mike	80000

department_id	location
1	NY
2	SF
3	Austin

Subqueries

In general, you can replace any table name with a SELECT statement.

```
SELECT ..... FROM (SELECT ....)
```

If a query returns a **single value**, you can treat it as such.

```
WHERE var1 = (SELECT ...)
```

If a query returns a **single column**, you can treat it sort of like a list/vector

```
WHERE var1 IN (SELECT ...)
```

Order of Evaluation of a SQL SELECT Statement

1. FROM + JOIN: first the product of all tables is formed

2. WHERE: the where clause filters rows that do not meet the search condition

3. GROUP BY + (COUNT, SUM, etc): the rows are grouped using the columns in the group by clause and the aggregation functions are applied on the grouping

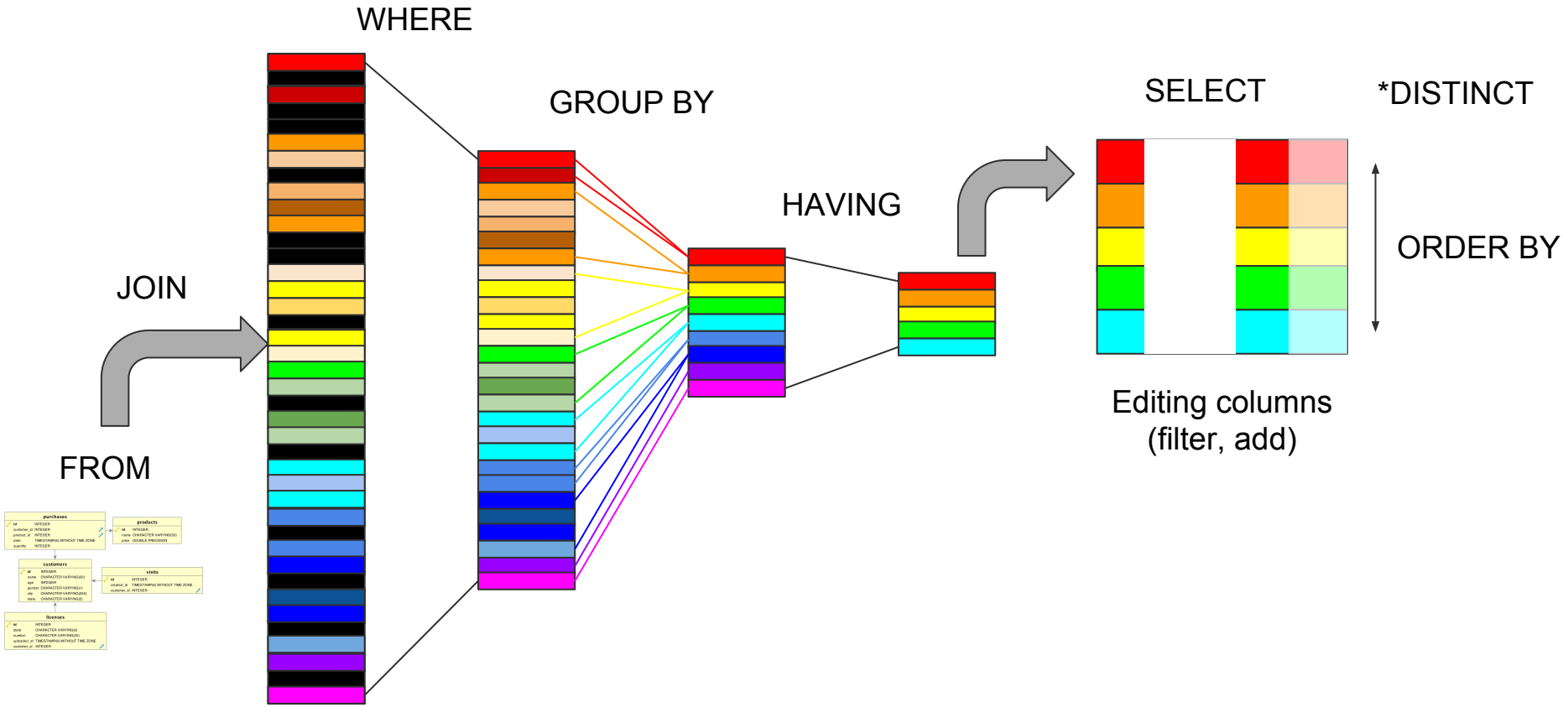
4. HAVING: like the WHERE clause, but can be applied after aggregation (3.)

5. SELECT: the targeted list of columns are evaluated and returned

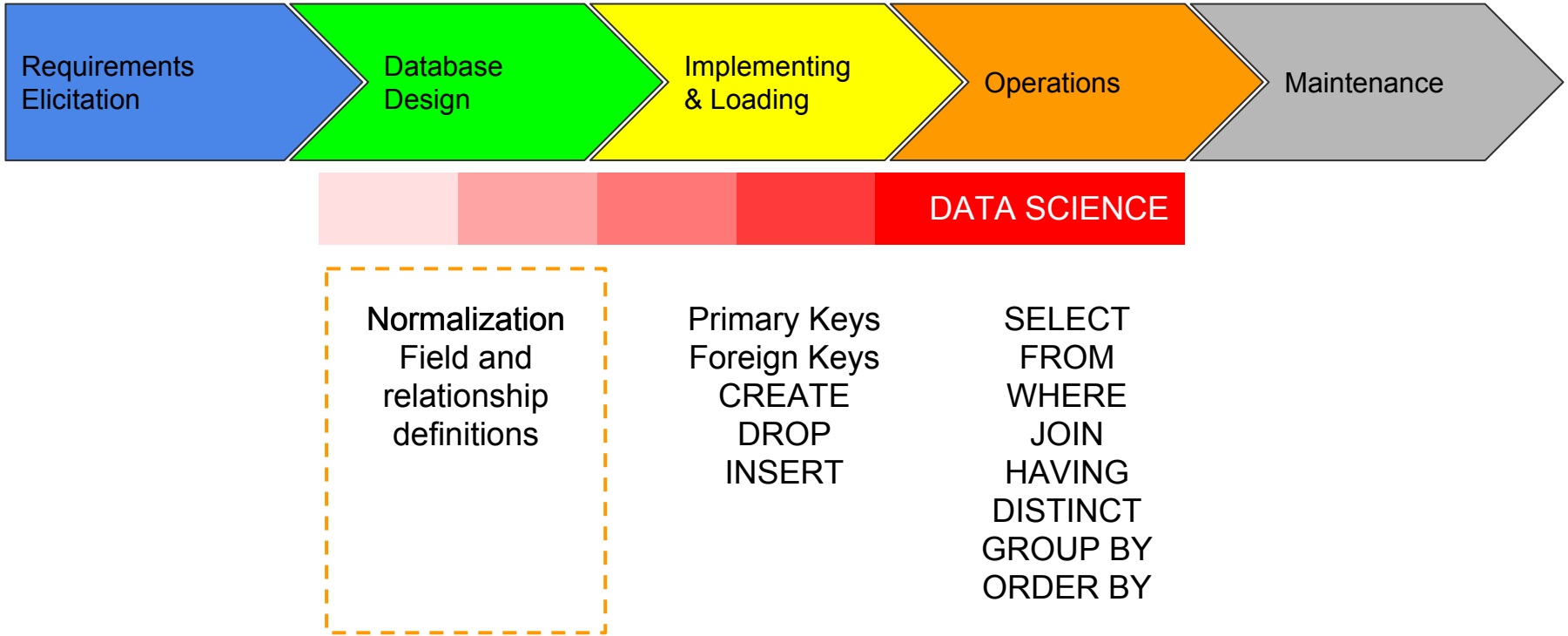
6. DISTINCT: duplicate rows are eliminated

7. ORDER BY: the resulting rows are sorted

Order of Evaluation of a SQL SELECT Statement

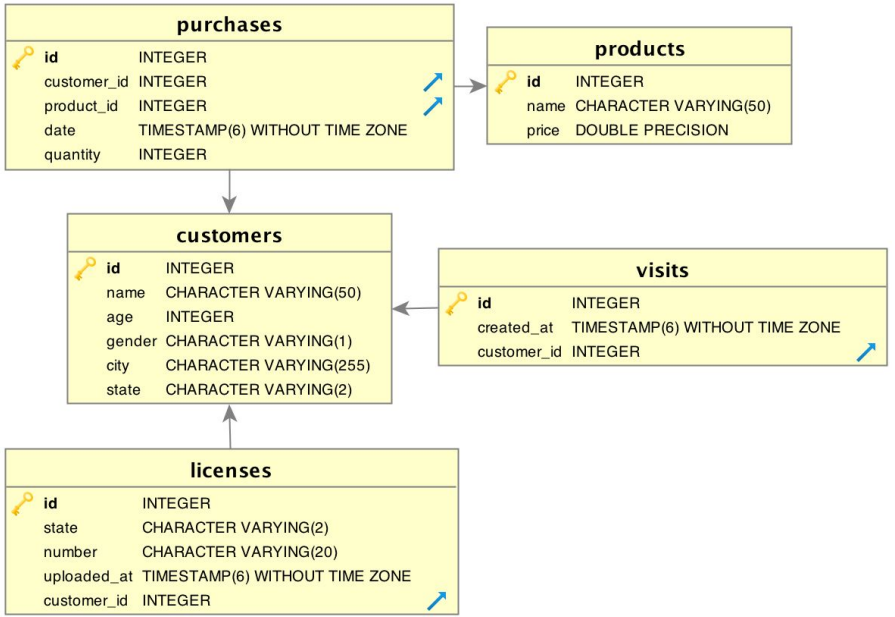


Concepts of the day in the DBLC



Database Normalization

- Minimizes Redundancy, for example:
  - Details about a user(address, age) are only stored once (in a users table)
  - Any other table (eg. purchases) where this data might be relevant, only references the user_id
  - Choose Normalized or Denormalized Schemas based on the use case:
    - Heavy reporting (Data Warehouse)
    - Transactional Systems (Ordering System)



OBJECTIVES

- **Connect** to a Postgres database
- Design and write **queries** to answer questions using an RDMBS (i.e. Postgres)
- Use **JOIN** to join two tables