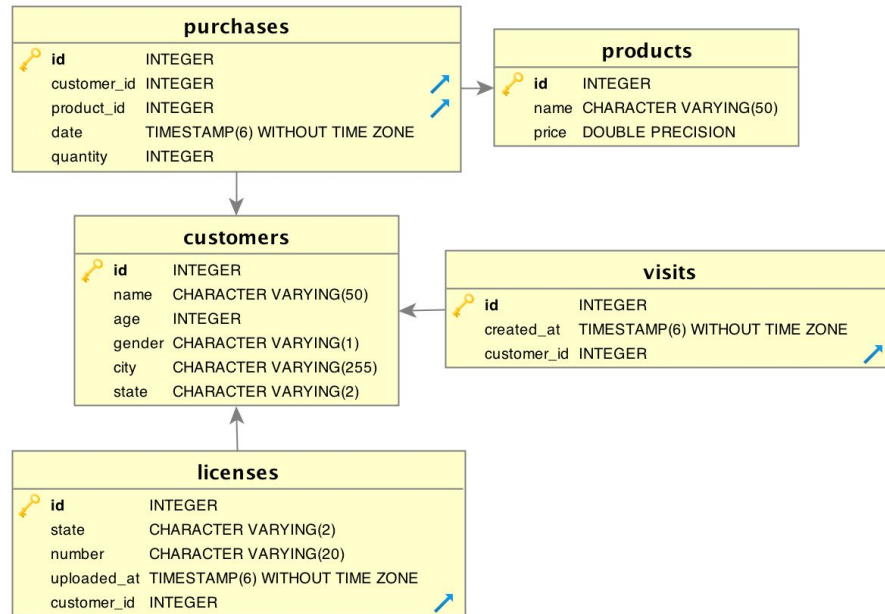# SQL

DSI, Galvanize

SQL

DSI, Galvanize

OBJECTIVES

- **Connect** to a Postgres database

- **Design** and **write** queries to answer questions using an RDBMS (i.e. Postgres)

- Connect to a SQL database via command line (i.e. Postgres).
- State the function of basic SQL commands.
- Write simple queries on a single table including SELECT, FROM, WHERE, CASE... clauses and aggregates.
- Write complex queries including JOINS and subqueries.
- Explain how indexing works in Postgres.
- Create and dump tables.
- Format a query to follow a standard style.

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

- is infrequently accessed and unlikely to be changed

RDBMS was the de facto standard for storing data

- Examples: Oracle, MySQL, SQLServer, Postgres

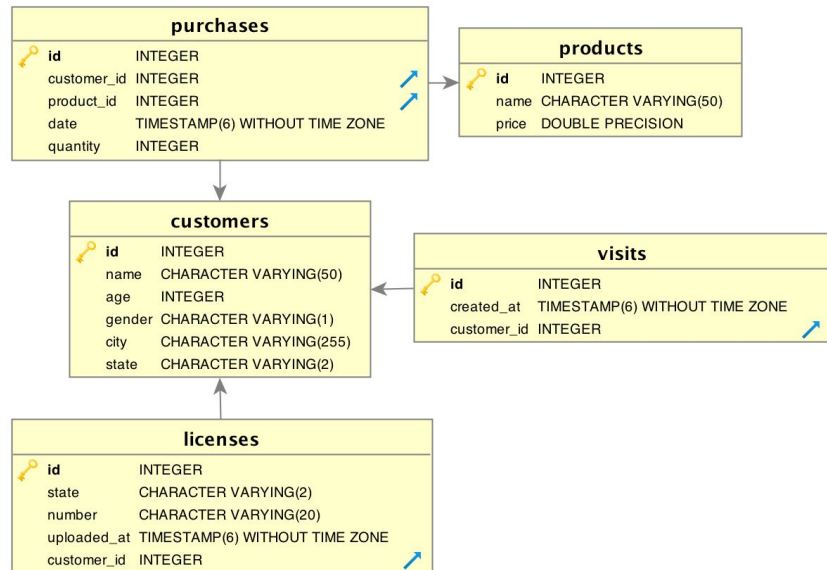- With "Big Data", this is beginning to change

Why RDBMS ?

An RDBMS provides the ability to

- **Model** relations in data

- **Query** data and their relations efficiently

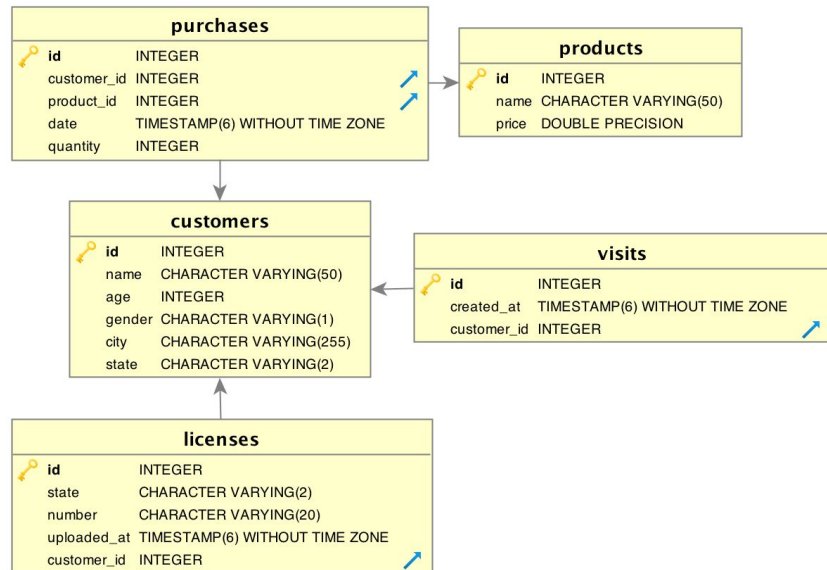- **Maintain** data consistency and integrity

It will require a Data Model

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

purchases

🔑 id	INTEGER
customer_id	INTEGER
product_id	INTEGER
date	TIMESTAMP(6) WITHOUT TIME ZONE
quantity	INTEGER

products

🔑 id	INTEGER
name	CHARACTER VARYING(50)
price	DOUBLE PRECISION

customers

🔑 id	INTEGER
name	CHARACTER VARYING(50)
age	INTEGER
gender	CHARACTER VARYING(1)
city	CHARACTER VARYING(255)
state	CHARACTER VARYING(2)

visits

🔑 id	INTEGER
created_at	TIMESTAMP(6) WITHOUT TIME ZONE
customer_id	INTEGER

licenses

🔑 id	INTEGER
state	CHARACTER VARYING(2)
number	CHARACTER VARYING(20)
uploaded_at	TIMESTAMP(6) WITHOUT TIME ZONE
customer_id	INTEGER

# RDBMS and SQL (Structured Query Language)

- SQL is the language used to query relational databases

- **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.

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) ?

...

PostGres Basics
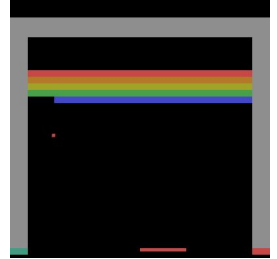
Ways to use psql in the shell/term:

`$` `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 with a text editor*
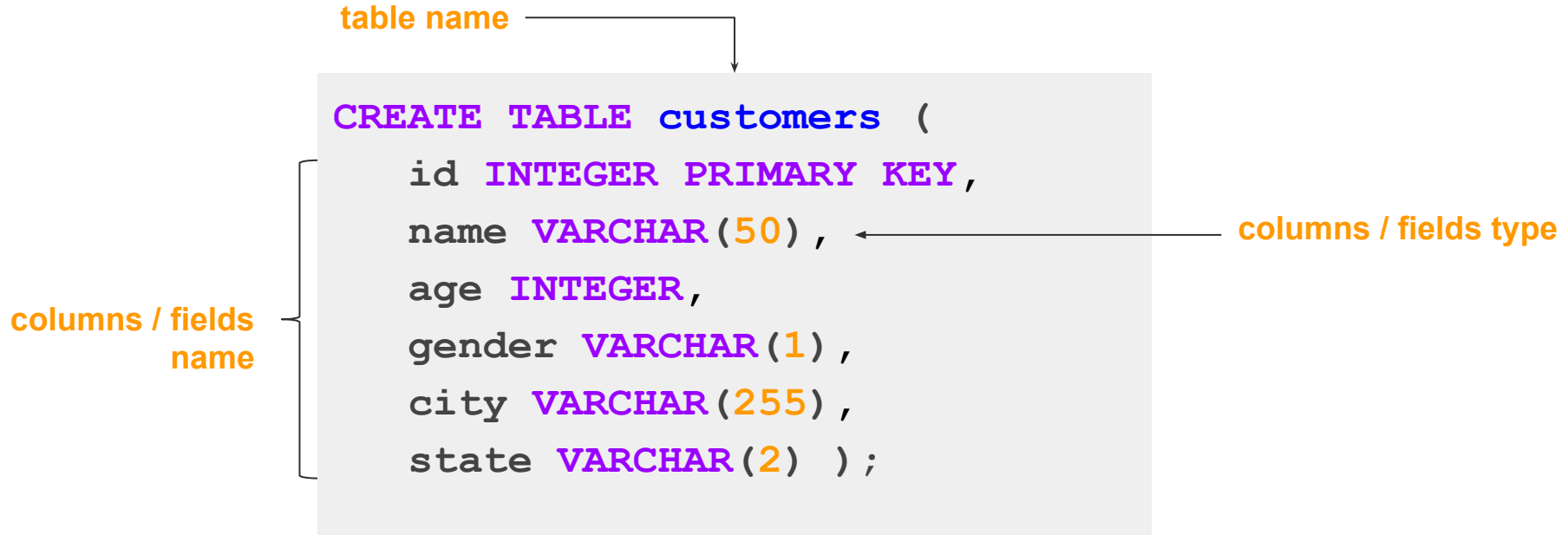- *Use it to create a "dsilecture" database on your psql server*

PostGres Basics

Useful psql commands at the prompt [link]:

`# \h`	SQL help
`# \?`	psql commands help
`# \l`	List all the tables in the database
`# \d`	Describe the table schema
`# \d db_name`	Describe tables for a specific db
`# \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

table name

```
CREATE TABLE customers (
    id INTEGER PRIMARY KEY,
    name VARCHAR(50),
    age INTEGER,
    gender VARCHAR(1),
    city VARCHAR(255),
    state VARCHAR(2) );
```

columns / fields type

columns / fields name

Inserting values in a table

table name

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

records
and their values

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;
```
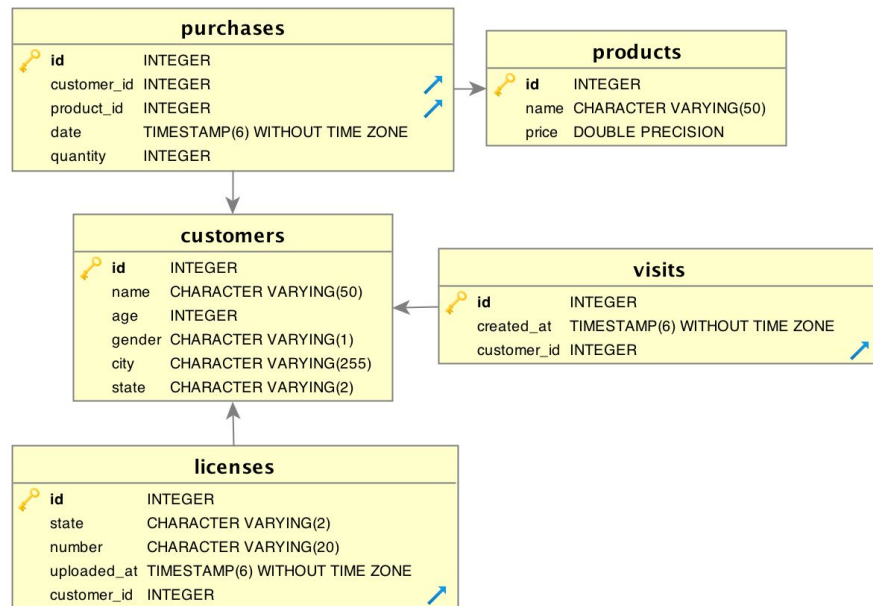
Designing a database with keys

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 combination of columns, hash, timestamp..etc.,

Foreign Keys

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

SQL Syntax

All SQL queries have three main ingredient :

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

Advanced: You can use `EXPLAIN` to look at the how

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

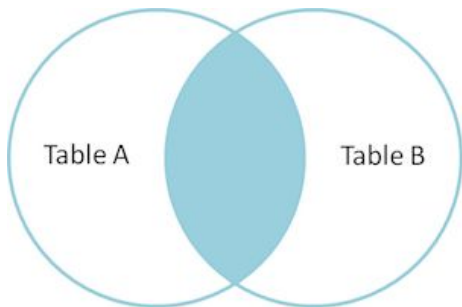JOIN clause used to query across multiple tables using foreign keys

Every JOIN has two segments:

Specifying the tables to JOIN
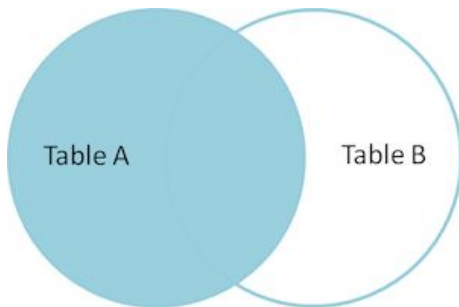Specifying the columns to match

JOIN types

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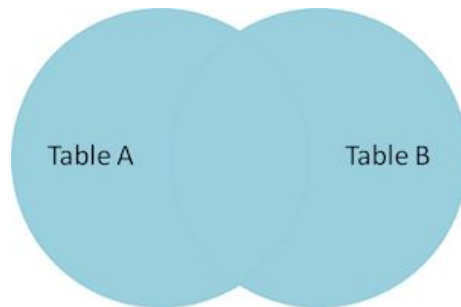based on the given keys.

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

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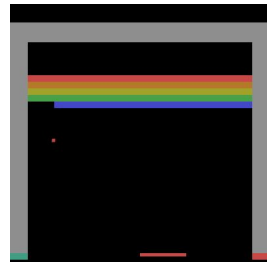will keep the rows
of both tables
no matter what

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

How many rows would result after an
(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

Given the following query, number what order the commands are executed:

```sql
SELECT a.userid, count(0) as recent_visits
FROM users a
LEFT JOIN visits b
ON a.userid = b.userid
WHERE b.dt > '2012-01-01'
GROUP BY a.userid
HAVING count(0) < 10
ORDER BY recent_visits;
```

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

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

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 ...)
```

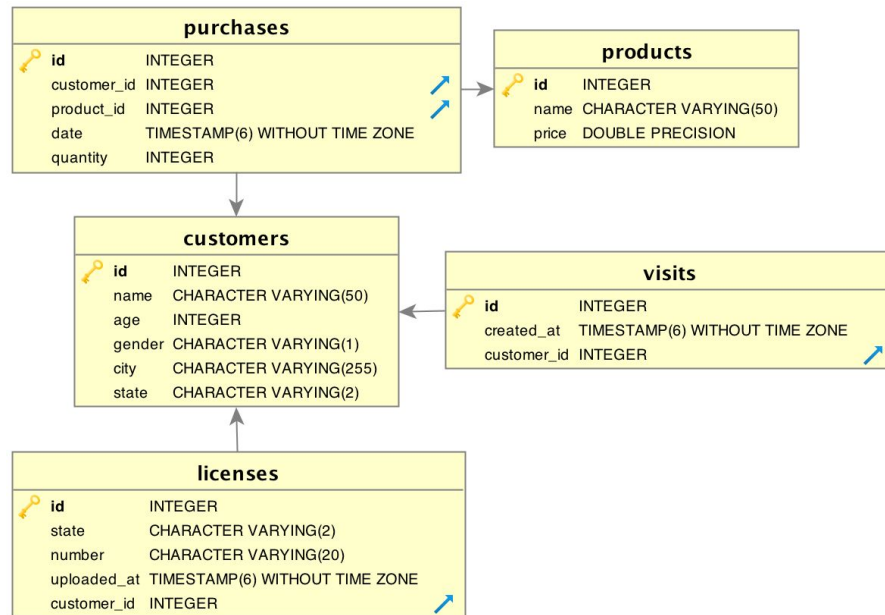If a query returns a **single column**, you can treat it sort of like a list/vector

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

- 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)

24

Advanced SQL

- "self join": JOIN a table with itself:
  http://www.w3resource.com/sql/joins/perform-a-self-join.php
- CROSS JOIN: join each row in table A with every row in table B:
  http://www.w3resource.com/sql/joins/cross-join.php
- window functions:
  https://www.postgresql.org/docs/9.1/static/tutorial-window.html
- COALESCE: often used to turn NULL values into non-null values:
  https://www.postgresql.org/docs/9.5/static/functions-conditional.html
- EXCEPT: return rows from one SELECT statement that are NOT returned by a second SELECT statement:
  https://www.tutorialspoint.com/sql/sql-except-clause.htm