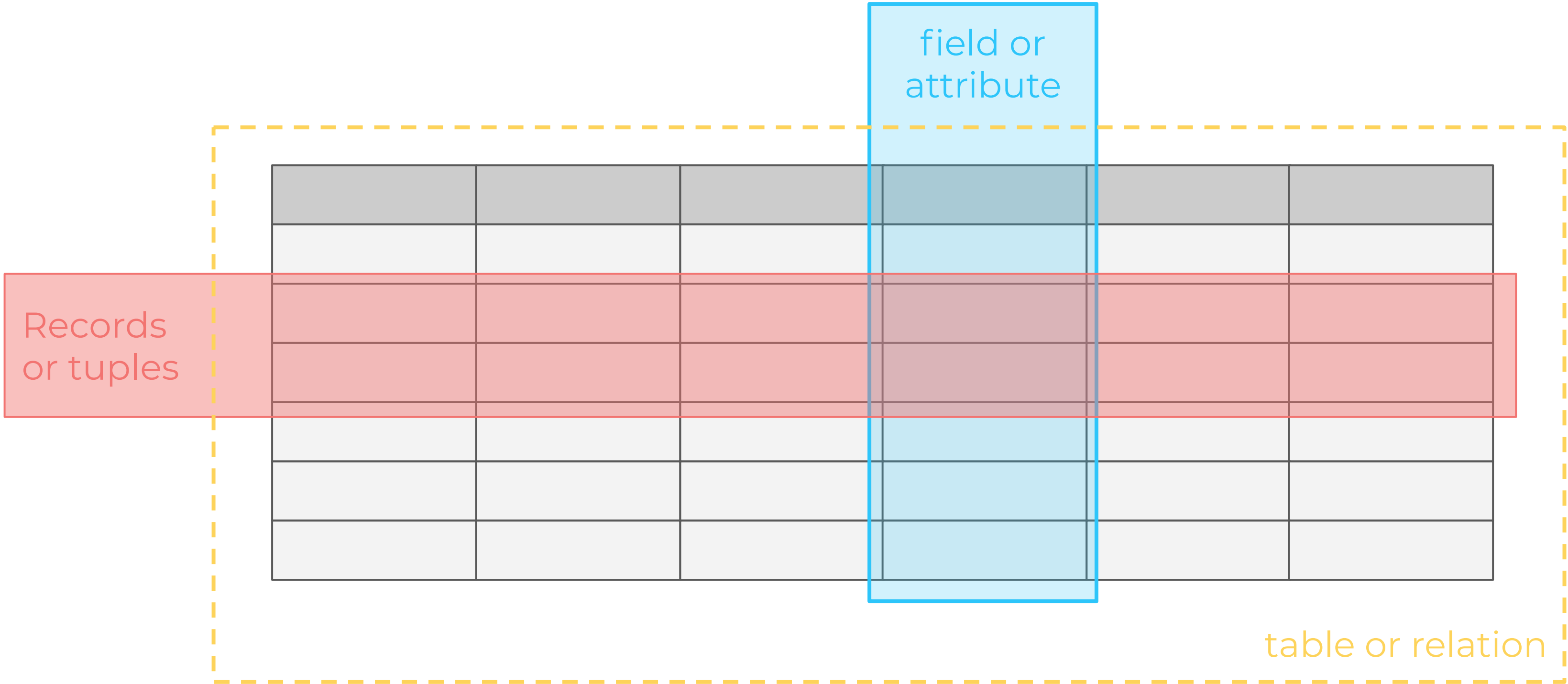
IRON HACK

INTRO TO POSTGRESQL

DATA ANALYTICS

Basic Terminology



Queries

The process of retrieving or the command to retrieve data from a database is called a **query**. SQL uses the SELECT statement to specify queries. The basic structure of a query is:

```
>> SELECT column_a, column_b, ..., column_z
>> FROM table
>> [WHERE some condition is met];
```

TIP | Whenever you see a pair of square brackets "[ ]" it means it's an optional statement

Views and Materialized Views

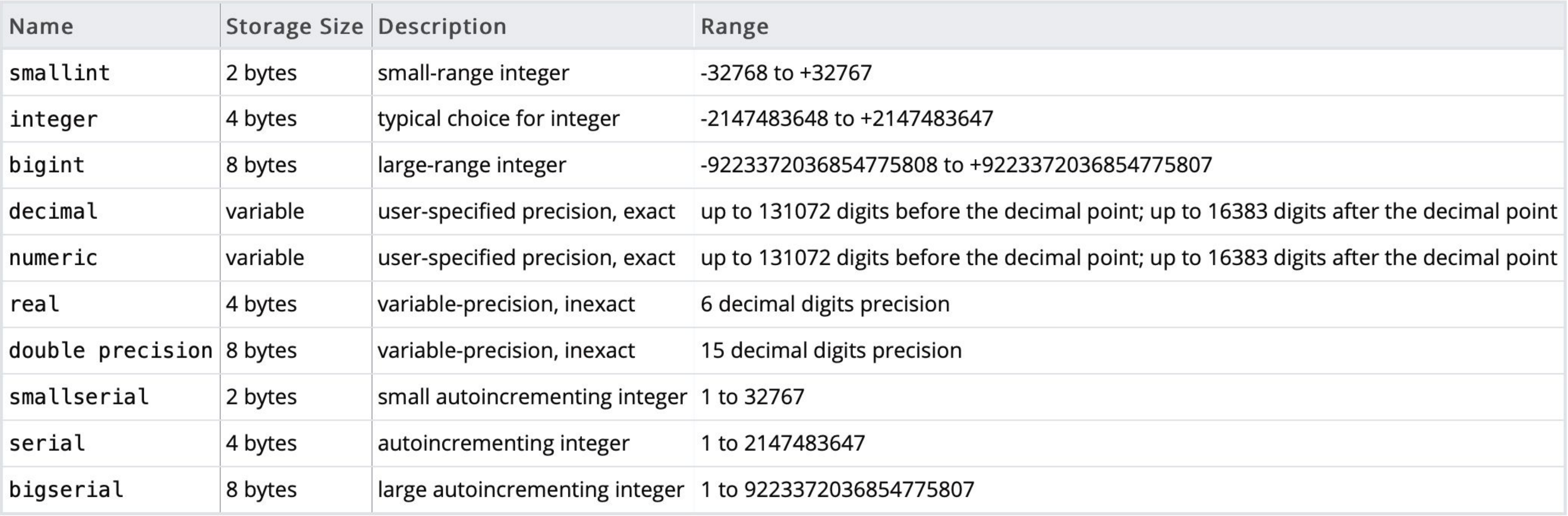
A **View** is a named query. It is a useful way to save complex or recurring queries. Their output isn't saved, the query is run every time the view is referenced.

```
>> CREATE VIEW view_name
>> AS query
```

Materialized Views work a little bit different. They are tables that remember the queries used to create them. Therefore, they can be refreshed later upon demand.

```
>> CREATE MATERIALIZED VIEW table_name
>> AS query
```

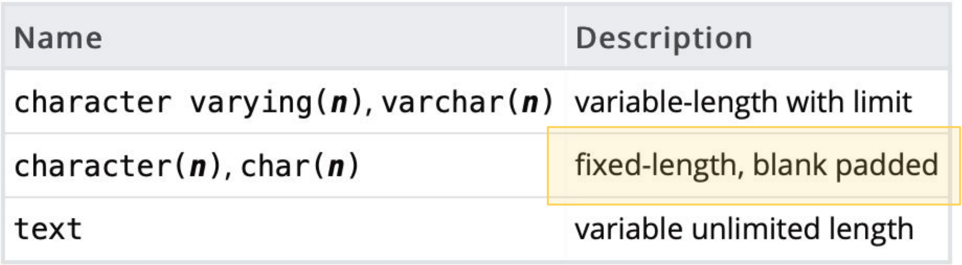
Data Types:
Numeric Data



Name	Storage Size	Description	Range
`smallint`	2 bytes	small-range integer	-32768 to +32767
`integer`	4 bytes	typical choice for integer	-2147483648 to +2147483647
`bigint`	8 bytes	large-range integer	-9223372036854775808 to +9223372036854775807
`decimal`	variable	user-specified precision, exact	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
`numeric`	variable	user-specified precision, exact	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
`real`	4 bytes	variable-precision, inexact	6 decimal digits precision
`double precision`	8 bytes	variable-precision, inexact	15 decimal digits precision
`smallserial`	2 bytes	small autoincrementing integer	1 to 32767
`serial`	4 bytes	autoincrementing integer	1 to 2147483647
`bigserial`	8 bytes	large autoincrementing integer	1 to 9223372036854775807

SOURCE: PostgreSQL Documentation

Data Types:
Character Data



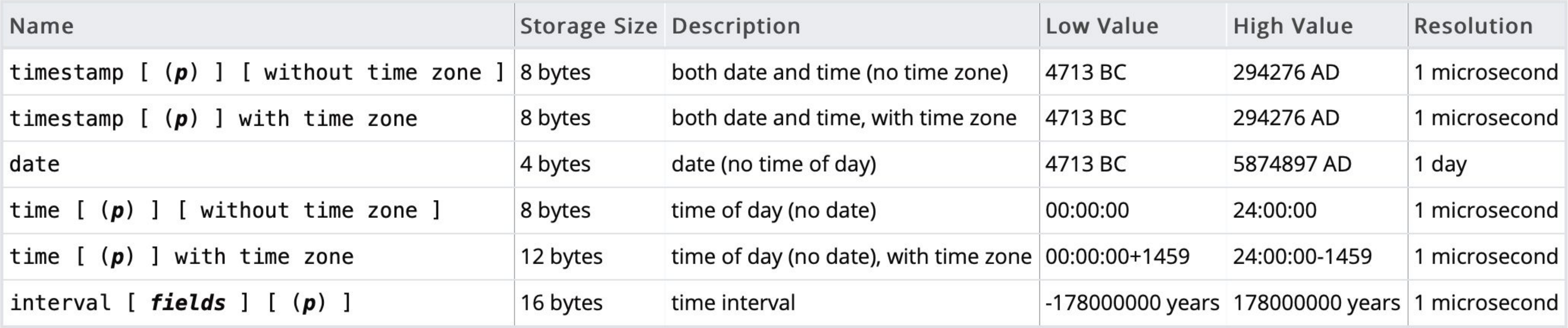
Name	Description
character varying(***n***), varchar(***n***)	variable-length with limit
character(***n***), char(***n***)	fixed-length, blank padded
text	variable unlimited length

SOURCE: PostgreSQL Documentation

IMPORTANT:

While character(***n***) has performance advantages in some other database systems, there is no such advantage in PostgreSQL; in fact character(***n***) is usually the slowest of the three because of its additional storage costs. In most situations **text** or **character varying** should be used instead.

Data Types:
Date and Time Objects



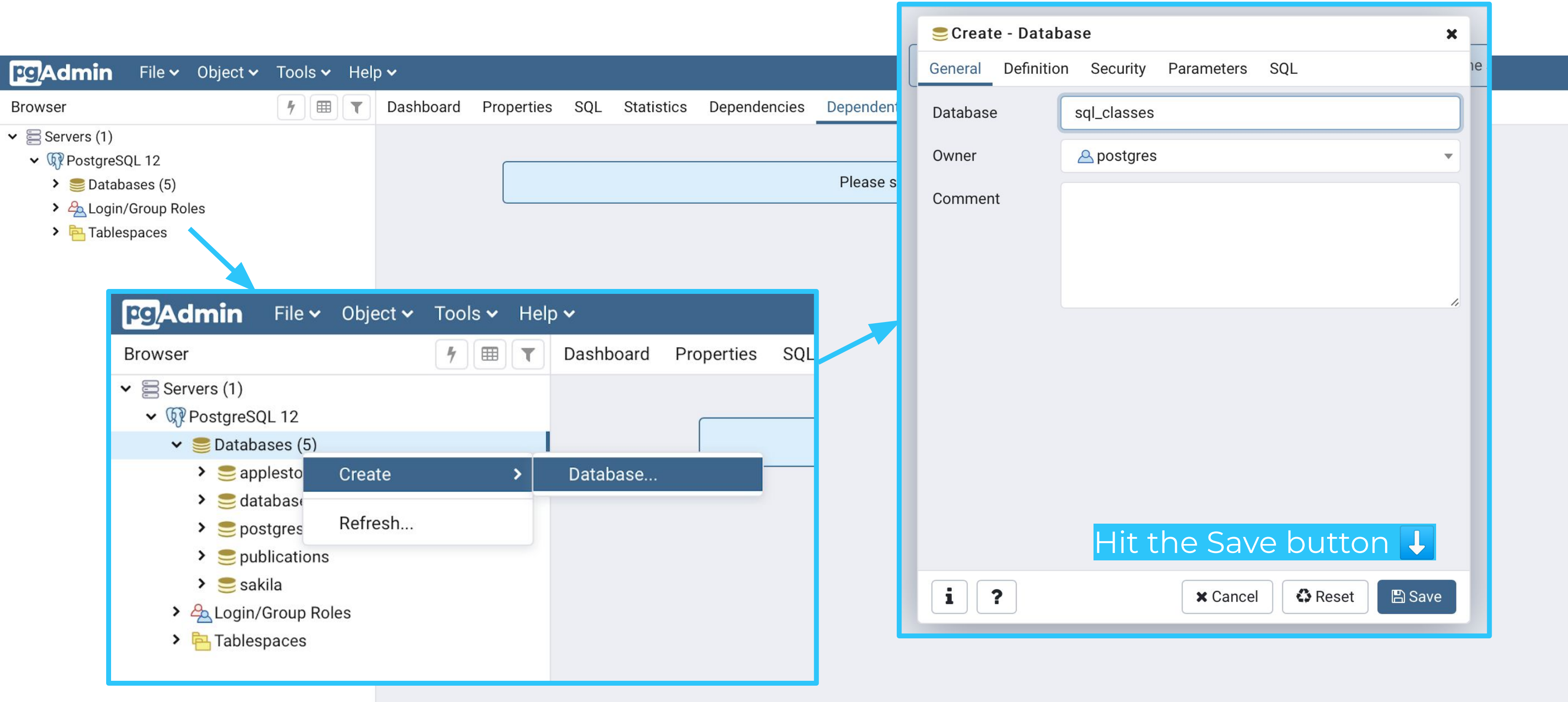
Name	Storage Size	Description	Low Value	High Value	Resolution
timestamp [(*p*)] [without time zone]	8 bytes	both date and time (no time zone)	4713 BC	294276 AD	1 microsecond
timestamp [(*p*)] with time zone	8 bytes	both date and time, with time zone	4713 BC	294276 AD	1 microsecond
date	4 bytes	date (no time of day)	4713 BC	5874897 AD	1 day
time [(*p*)] [without time zone]	8 bytes	time of day (no date)	00:00:00	24:00:00	1 microsecond
time [(*p*)] with time zone	12 bytes	time of day (no date), with time zone	00:00:00+1459	24:00:00-1459	1 microsecond
interval [***fields***] [(*p*)]	16 bytes	time interval	-178000000 years	178000000 years	1 microsecond

SOURCE: PostgreSQL Documentation

TIP | There are a lot of ways to input date type information on PostgreSQL, but it's easier if you stick to the following: YYYY-MM-DD

Let's start your journey with SQL

Creating a Database



Database: sql_classes

Owner: postgres

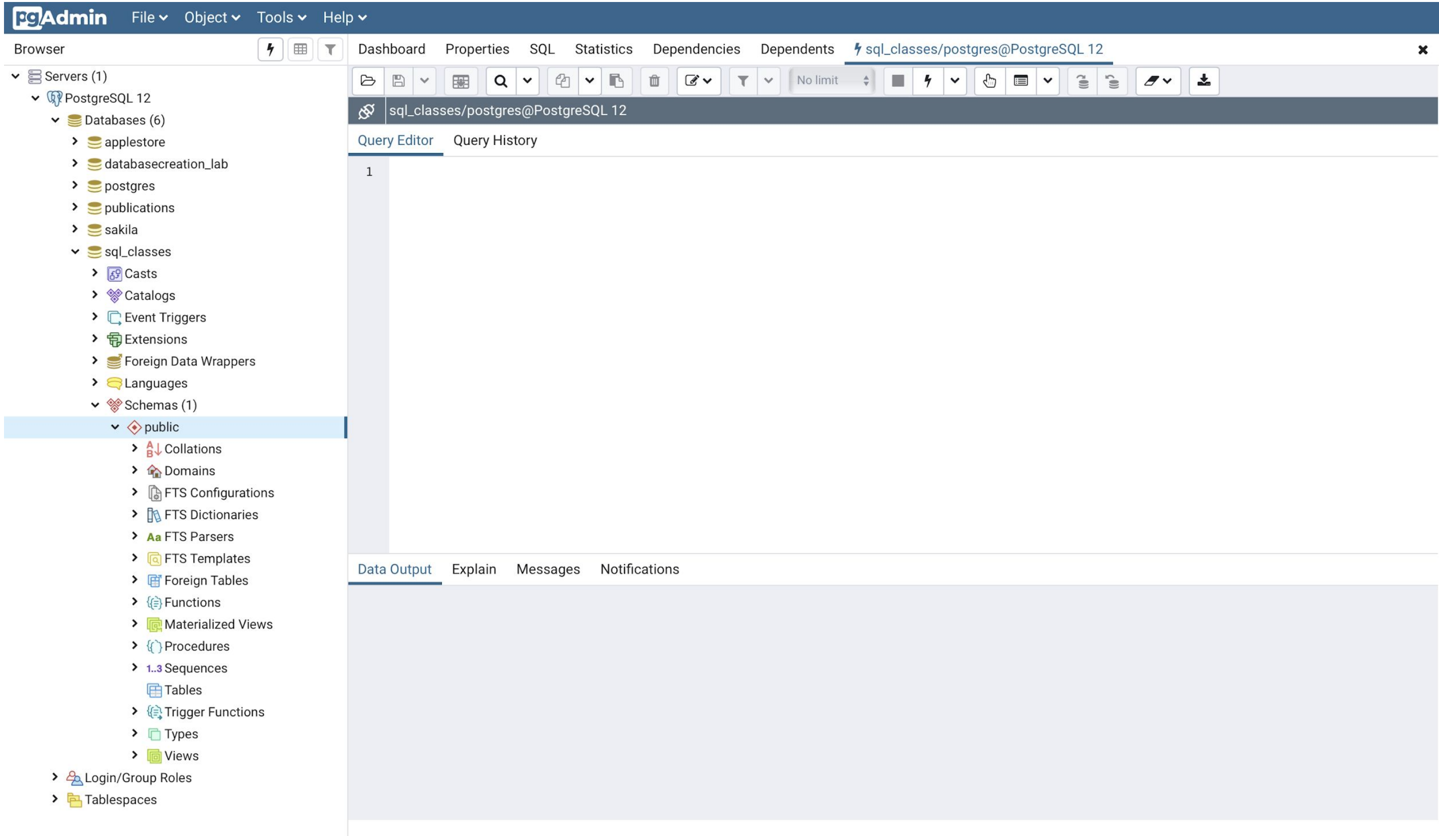
Comment:

Hit the Save button ⬇

Query Tool

Find the database you've created in the dropdown list inside the "Databases" section.

Toggle down the "Schemas" section and right-click on the "public" menu item. Select "Query Tool" and your screen should look like this one.



Now, we'll focus on our PgAdmin 4 interface

Don't forget to open the Intro2PostgreSQL.sql file to follow along

Query Commands

>> **SELECT:** Allows us to choose the specific fields (columns) you would like displayed in your query results;

>> **FROM:** Allows us to specify what table in the database the data is going to come from;

>> **WHERE:** Allows us to specify some conditions to filter the data;

>> **ORDER BY:** Allows us to sort the data in ascending or descending order by multiple fields;

>> **GROUP BY:** Allows us to group data by the values in one or more fields.

Query Example

```
SELECT app_name, price, total_ratings, overall_rating,
genre
FROM ratings
WHERE price = 0 and overall_rating >= 4
ORDER BY total_ratings DESC
LIMIT 20;
```

TIP | You can add a condition based on a column that you don't want to return.

TIP | If you want to sort your results by descending order you need to add DESC to your ORDER BY command, because the default behavior of this command is sorting things in ascending order

Query Example

```
SELECT genre, COUNT(genre) AS number_of_apps
FROM ratings
GROUP BY genre
ORDER BY number_of_apps DESC;
```

TIP | We can use the column position on the SELECT statement to call them in other commands

```
             1                          2
SELECT genre, COUNT(genre) AS number_of_apps
FROM ratings
GROUP BY 1
ORDER BY 2 DESC;
```

Joins and Relationships

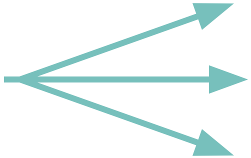
Relationships

One-to-One

When a record in a table is related to only one record in other table.

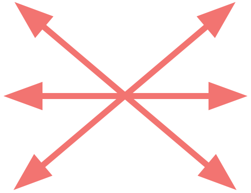
One-to-Many

When a record in a table is related to two or more records in other table, but the opposite isn't true.

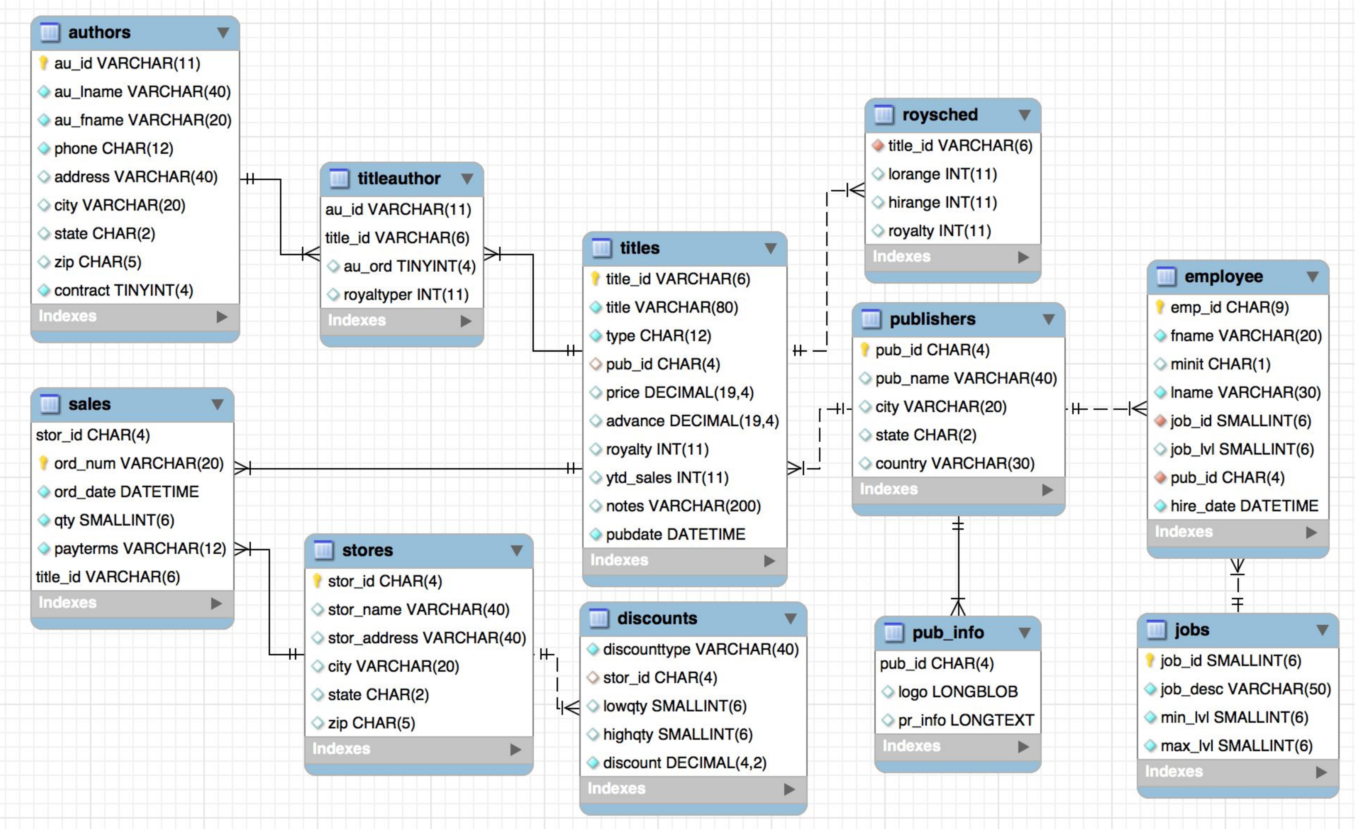


Many-to-Many

When the records of a table are related to two or more records in other table, and the records of that table are related to two or more records in the former one.



Entity Relationship Diagram



authors
au_id VARCHAR(11)
au_lname VARCHAR(40)
au_fname VARCHAR(20)
phone CHAR(12)
address VARCHAR(40)
city VARCHAR(20)
state CHAR(2)
zip CHAR(5)
contract TINYINT(4)
Indexes

titleauthor
au_id VARCHAR(11)
title_id VARCHAR(6)
au_ord TINYINT(4)
royaltyper INT(11)
Indexes

titles
title_id VARCHAR(6)
title VARCHAR(80)
type CHAR(12)
pub_id CHAR(4)
price DECIMAL(19,4)
advance DECIMAL(19,4)
royalty INT(11)
ytd_sales INT(11)
notes VARCHAR(200)
pubdate DATETIME
Indexes

roysched
title_id VARCHAR(6)
lorange INT(11)
hirange INT(11)
royalty INT(11)
Indexes

publishers
pub_id CHAR(4)
pub_name VARCHAR(40)
city VARCHAR(20)
state CHAR(2)
country VARCHAR(30)
Indexes

employee
emp_id CHAR(9)
fname VARCHAR(20)
minit CHAR(1)
lname VARCHAR(30)
job_id SMALLINT(6)
job_lvl SMALLINT(6)
pub_id CHAR(4)
hire_date DATETIME
Indexes

sales
stor_id CHAR(4)
ord_num VARCHAR(20)
ord_date DATETIME
qty SMALLINT(6)
payterms VARCHAR(12)
title_id VARCHAR(6)
Indexes

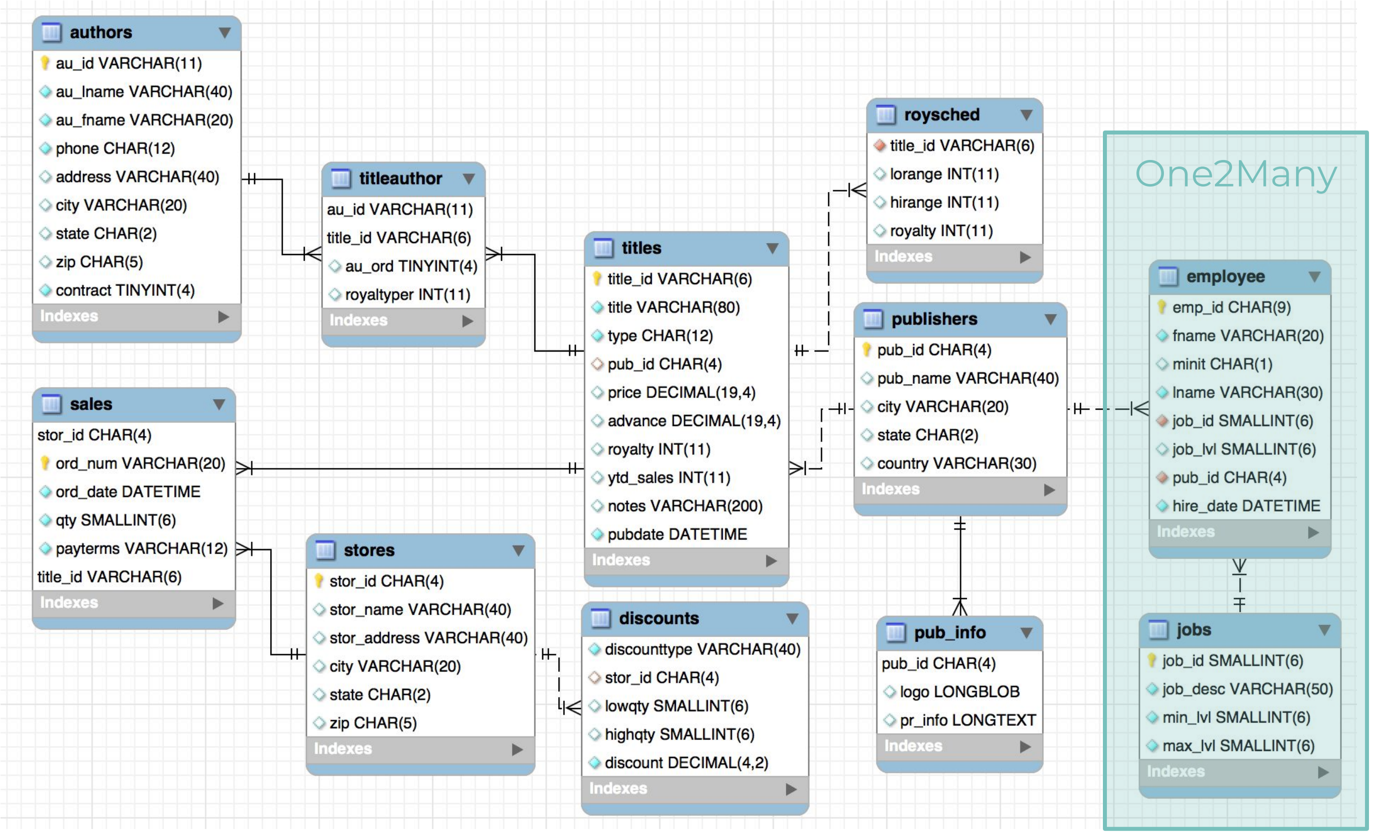
stores
stor_id CHAR(4)
stor_name VARCHAR(40)
stor_address VARCHAR(40)
city VARCHAR(20)
state CHAR(2)
zip CHAR(5)
Indexes

discounts
discounttype VARCHAR(40)
stor_id CHAR(4)
lowqty SMALLINT(6)
highqty SMALLINT(6)
discount DECIMAL(4,2)
Indexes

pub_info
pub_id CHAR(4)
logo LONGBLOB
pr_info LONGTEXT
Indexes

jobs
job_id SMALLINT(6)
job_desc VARCHAR(50)
min_lvl SMALLINT(6)
max_lvl SMALLINT(6)
Indexes

Entity Relationship Diagram



authors
- au_id VARCHAR(11)
- au_lname VARCHAR(40)
- au_fname VARCHAR(20)
- phone CHAR(12)
- address VARCHAR(40)
- city VARCHAR(20)
- state CHAR(2)
- zip CHAR(5)
- contract TINYINT(4)
- Indexes

titleauthor
- au_id VARCHAR(11)
- title_id VARCHAR(6)
- au_ord TINYINT(4)
- royaltyper INT(11)
- Indexes

titles
- title_id VARCHAR(6)
- title VARCHAR(80)
- type CHAR(12)
- pub_id CHAR(4)
- price DECIMAL(19,4)
- advance DECIMAL(19,4)
- royalty INT(11)
- ytd_sales INT(11)
- notes VARCHAR(200)
- pubdate DATETIME
- Indexes

roysched
- title_id VARCHAR(6)
- lorange INT(11)
- hirange INT(11)
- royalty INT(11)
- Indexes

publishers
- pub_id CHAR(4)
- pub_name VARCHAR(40)
- city VARCHAR(20)
- state CHAR(2)
- country VARCHAR(30)
- Indexes

sales
- stor_id CHAR(4)
- ord_num VARCHAR(20)
- ord_date DATETIME
- qty SMALLINT(6)
- payterms VARCHAR(12)
- title_id VARCHAR(6)
- Indexes

stores
- stor_id CHAR(4)
- stor_name VARCHAR(40)
- stor_address VARCHAR(40)
- city VARCHAR(20)
- state CHAR(2)
- zip CHAR(5)
- Indexes

discounts
- discounttype VARCHAR(40)
- stor_id CHAR(4)
- lowqty SMALLINT(6)
- highqty SMALLINT(6)
- discount DECIMAL(4,2)
- Indexes

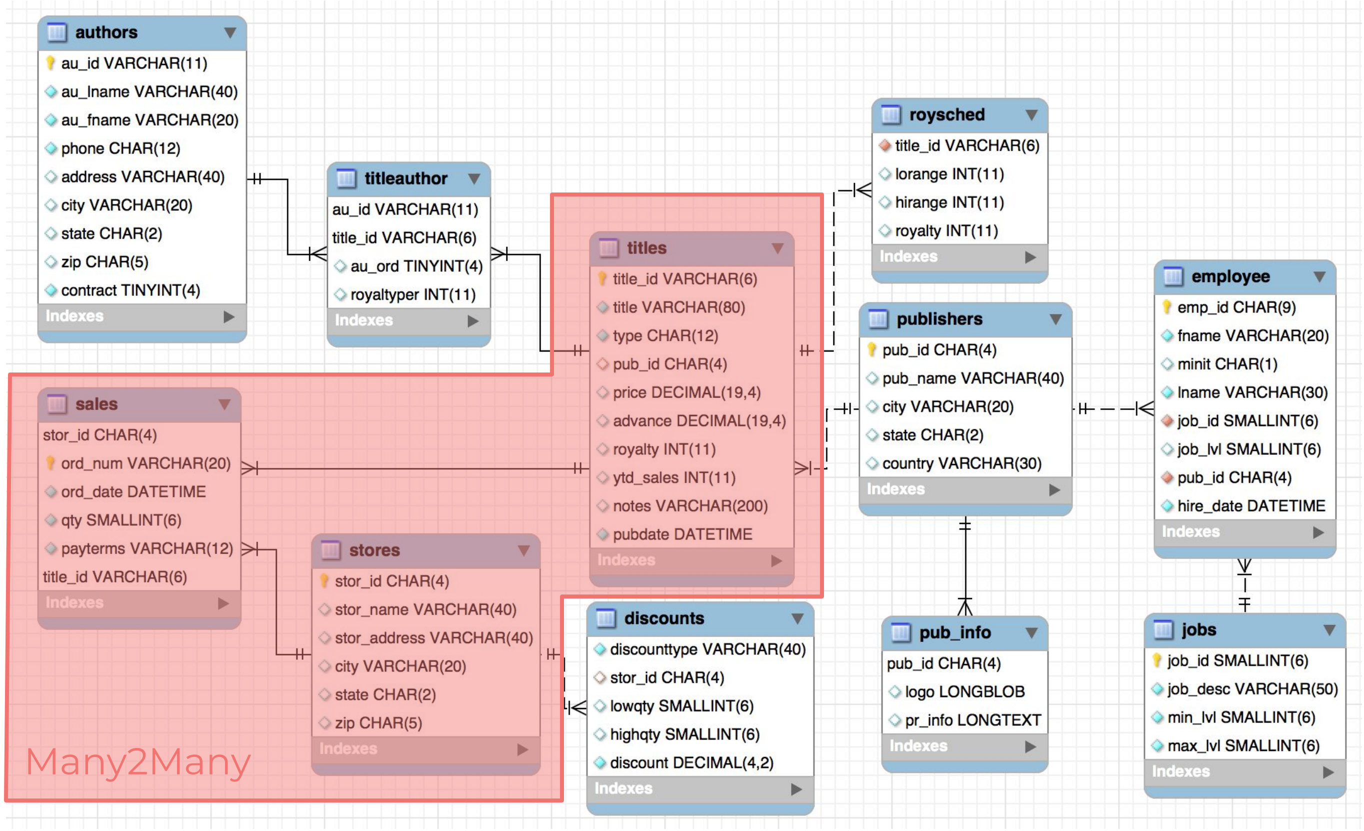
pub_info
- pub_id CHAR(4)
- logo LONGBLOB
- pr_info LONGTEXT
- Indexes

One2Many

employee
- emp_id CHAR(9)
- fname VARCHAR(20)
- minit CHAR(1)
- lname VARCHAR(30)
- job_id SMALLINT(6)
- job_lvl SMALLINT(6)
- pub_id CHAR(4)
- hire_date DATETIME
- Indexes

jobs
- job_id SMALLINT(6)
- job_desc VARCHAR(50)
- min_lvl SMALLINT(6)
- max_lvl SMALLINT(6)
- Indexes

Entity Relationship Diagram



authors
au_id VARCHAR(11)
au_lname VARCHAR(40)
au_fname VARCHAR(20)
phone CHAR(12)
address VARCHAR(40)
city VARCHAR(20)
state CHAR(2)
zip CHAR(5)
contract TINYINT(4)
Indexes

titleauthor
au_id VARCHAR(11)
title_id VARCHAR(6)
au_ord TINYINT(4)
royaltyper INT(11)
Indexes

roysched
title_id VARCHAR(6)
lorange INT(11)
hirange INT(11)
royalty INT(11)
Indexes

titles
title_id VARCHAR(6)
title VARCHAR(80)
type CHAR(12)
pub_id CHAR(4)
price DECIMAL(19,4)
advance DECIMAL(19,4)
royalty INT(11)
ytd_sales INT(11)
notes VARCHAR(200)
pubdate DATETIME
Indexes

publishers
pub_id CHAR(4)
pub_name VARCHAR(40)
city VARCHAR(20)
state CHAR(2)
country VARCHAR(30)
Indexes

employee
emp_id CHAR(9)
fname VARCHAR(20)
minit CHAR(1)
lname VARCHAR(30)
job_id SMALLINT(6)
job_lvl SMALLINT(6)
pub_id CHAR(4)
hire_date DATETIME
Indexes

sales
stor_id CHAR(4)
ord_num VARCHAR(20)
ord_date DATETIME
qty SMALLINT(6)
payterms VARCHAR(12)
title_id VARCHAR(6)
Indexes

stores
stor_id CHAR(4)
stor_name VARCHAR(40)
stor_address VARCHAR(40)
city VARCHAR(20)
state CHAR(2)
zip CHAR(5)
Indexes

discounts
discounttype VARCHAR(40)
stor_id CHAR(4)
lowqty SMALLINT(6)
highqty SMALLINT(6)
discount DECIMAL(4,2)
Indexes

pub_info
pub_id CHAR(4)
logo LONGBLOB
pr_info LONGTEXT
Indexes

jobs
job_id SMALLINT(6)
job_desc VARCHAR(50)
min_lvl SMALLINT(6)
max_lvl SMALLINT(6)
Indexes

Entities Keys

PRIMARY KEY

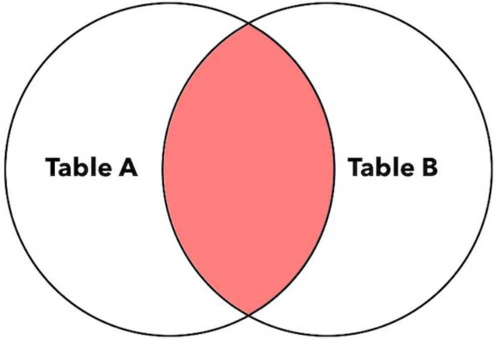
A primary key constraint indicates that a column, or group of columns, can be used as a unique identifier for rows in the table. This requires that the values be both unique and not null.

FOREIGN KEY

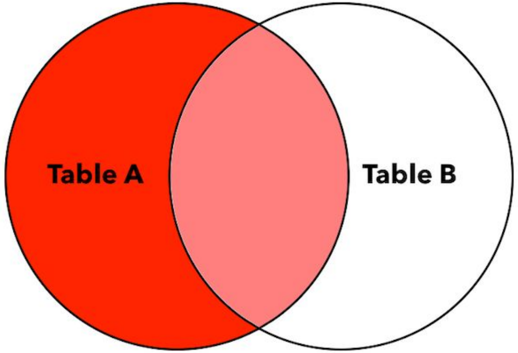
The foreign key is very simple: it is the primary key of another table. Therefore, it helps us to connect two tables.

Joining Tables

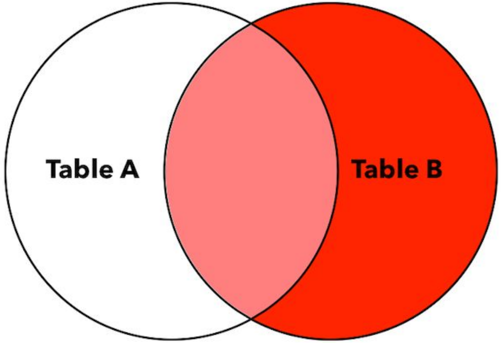
INNER JOIN



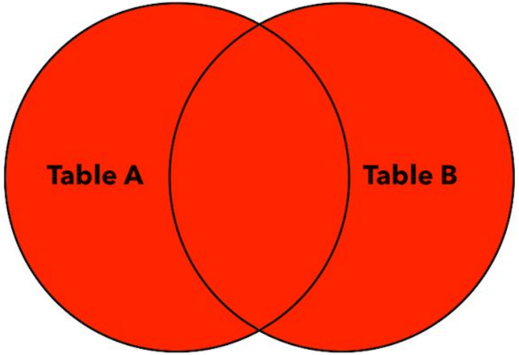
LEFT JOIN



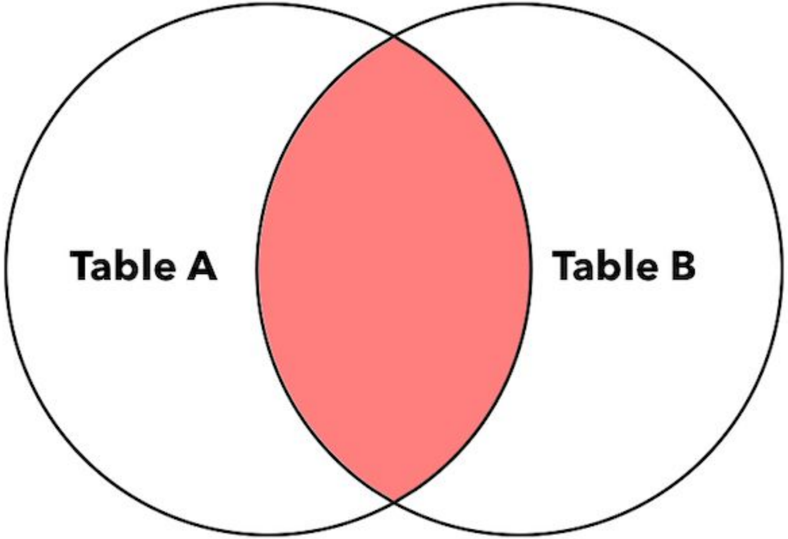
RIGHT JOIN



FULL OUTER JOIN

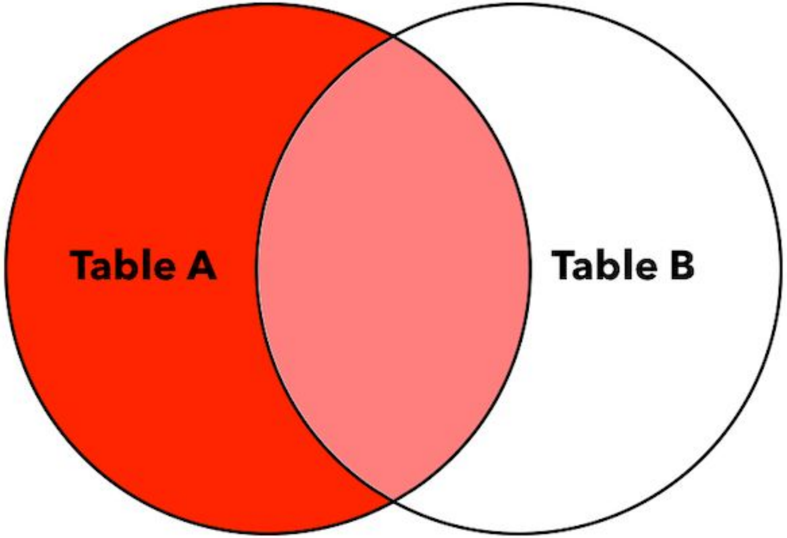


Inner Join



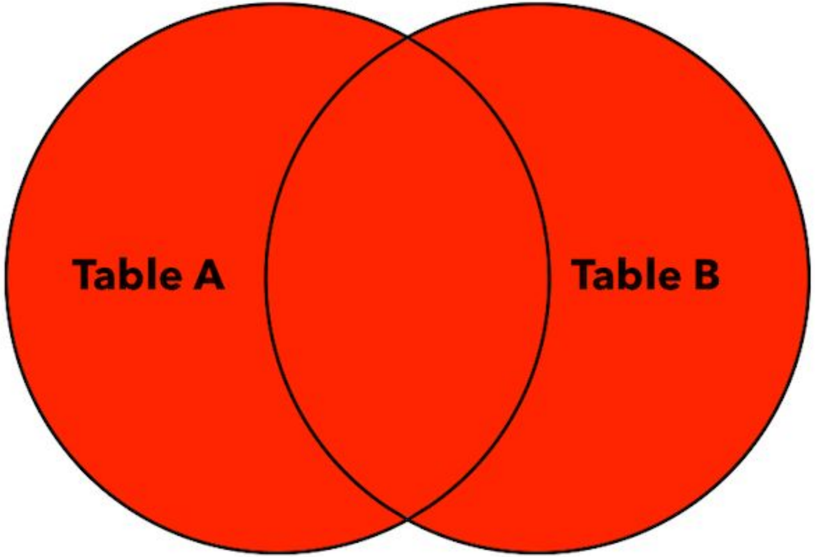
```
SELECT *
FROM table_a
JOIN table_b
ON table_a.key = table_b.key
```

Left Join



```
SELECT *
FROM table_a
LEFT JOIN table_b
ON table_a.key = table_b.key
```

Outer Join



```
SELECT *
FROM table_a
FULL [OUTER] JOIN table_b
ON table_a.key = table_b.key
```