SQL
Structured Query Language

Database

Database

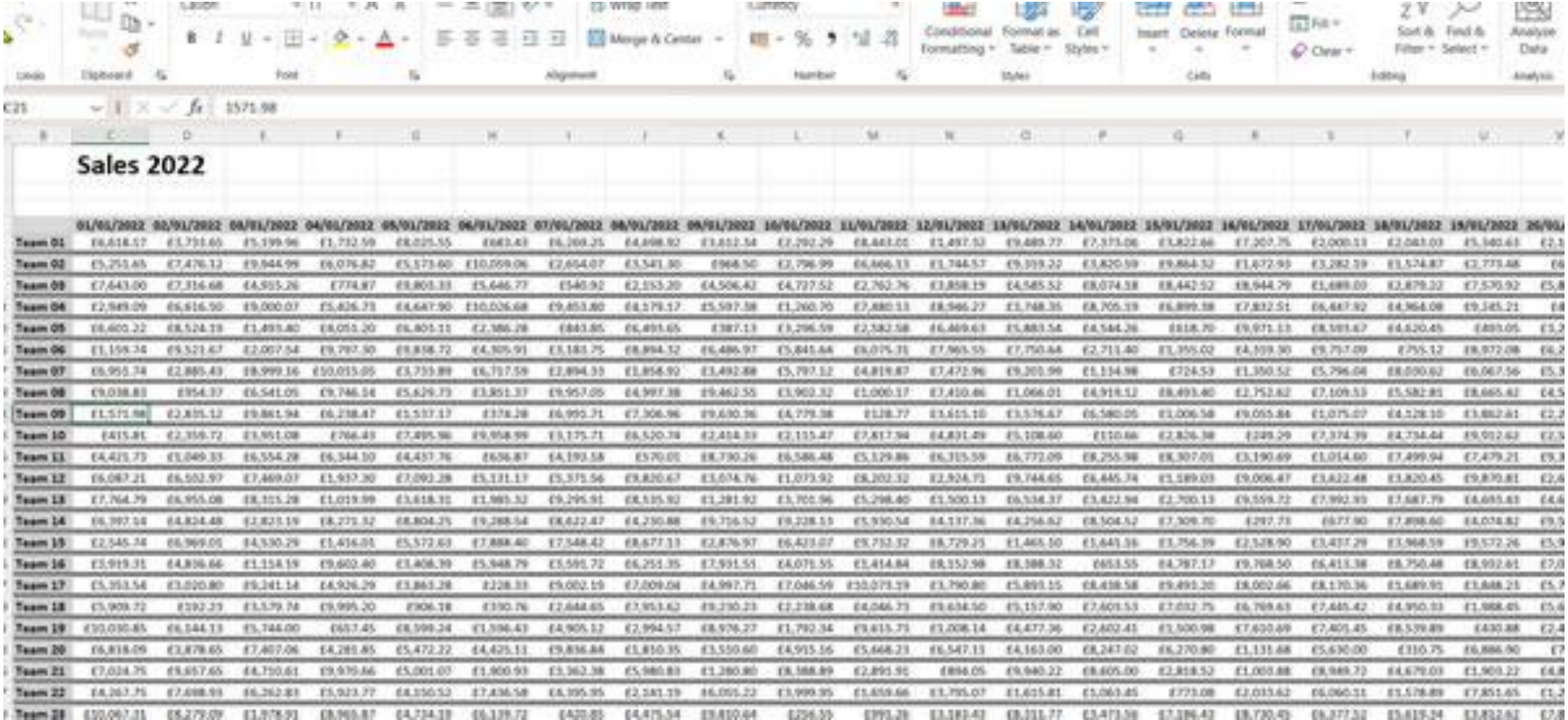
Student ID	Name	Email	Course	Teacher	Grade
S1001	Jordan Lee	j.lee@university.edu	Intro to Python	Prof. Ada Lovelace	A
S1001	Jordan Lee	j.lee@university.edu	Data Structures	Prof. Donald Knuth	B+
S1002	Sarah Chen	s.chen@university.edu	Intro to Python	Prof. Ada Lovelace	B+
S1002	Sarah Chen	s.chen@university.edu	Macroeconomics	Dr. Janet Yellen	A-
S1002	Sarah Chen	s.chen@university.edu	Business Ethics	Prof. Michael Sandel	B
S1003	Marcus Thorne	m.thorne@university.edu	Data Structures	Prof. Donald Knuth	A
S1003	Marcus Thorne	m.thorne@university.edu	Physics I	Dr. Richard Feynman	B

I am a school Administrator.

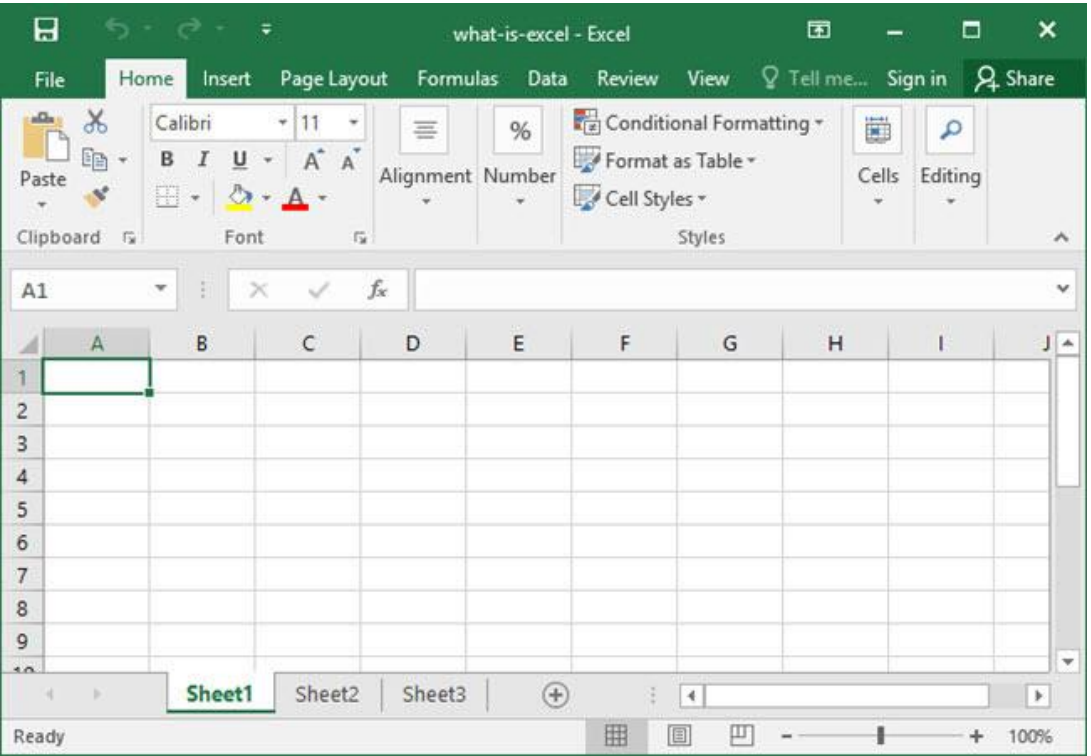
I would like to keep track of my students, courses, and teachers



Database



Database


what-is-excel - Excel
File
Home
Insert
Page Layout
Formulas
Data
Review
View
Tell me...
Sign in
Share
Paste
Calibri
11
Alignment
Number
Conditional Formatting
Format as Table
Cell Styles
Cells
Editing
Clipboard
Font
Styles
A1
Sheet1
Sheet2
Sheet3
Ready
100%



Why use a Database



1. **Data Organization and Structure**
    a. Information in databases is structured and organized into tables with defined columns and data types
2. **Efficient Data Retrieval**
    a. Databases allow Data to be searched, filtered, and set to extract specific information from massive datasets using SQL
3. **Data Integrity**
    a. Databases enforce strict rules and constraints on data meaning data in the database can be trusted as valid
4. **Scalability**
    a. Databases can scale to store between thousands to millions of records efficiently

Where are Databases Used?



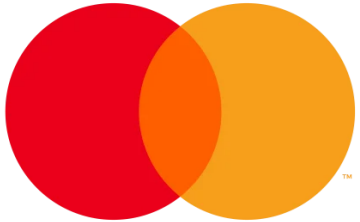












Types of Databases







1. Relational

Structured & Typed Tables / Attributes

Vertically Scale (Bigger Server)

Ex: Sqlite, mySQL, PostgreSQL

2. Non Relational

Non Structured Flexible Data

Prioritize Speed and Availability

Ex: MongoDB, Redis











3. Specialized

Designed to handle massive amounts of Data

Data storage is separated from Data Compute

Ex: Snowflake, Databricks







Database Jargon

Database Jargon

There are 3 levels to a database

- Database
  - Schema (Drawer)
    - Table (Folder containing Data)



Tables

Tables are a group of columns each having a set data type.

With Rows / Records for each Entity stored

Usually, an **Object or Entity we would like to store data about**

Tables have:

- Columns (Fields)
- Rows (Records)

student				
id	**name**	**surname**	**age**	**group_id**
1	John	Brown	19	1
2	Emma	Carly	23	2
3	Kesha	Petruk	20	1

Columns (Fields)

Columns have a name, and a data type

Columns:

- Id: Integer
- Name: string
- Surname: string
- Age: integer
- Group_id: integer

student				
id	name	surname	age	group_id
1	John	Brown	19	1
2	Emma	Carly	23	2
3	Kesha	Petruk	20	1

Rows (Records)

Rows are the records in each table

Here we have 3 records in this table:

- John (id: 1)
- Emma (id: 2)
- Kesha (id: 3)

student				
id	**name**	**surname**	**age**	**group_id**
1	John	Brown	19	1
2	Emma	Carly	23	2
3	Kesha	Petruk	20	1

Primary Key

One column in each table, that is a **UNIQUE-DEFINED IDENTIFIER for each row** (record) in the table.

There are 2 rules for a primary key:

1. It must be unique
2. It cannot be null

student				
id	**name**	**surname**	**age**	**group_id**
1	John	Brown	19	1
2	Emma	Carly	23	2
3	Kesha	Petruk	20	1

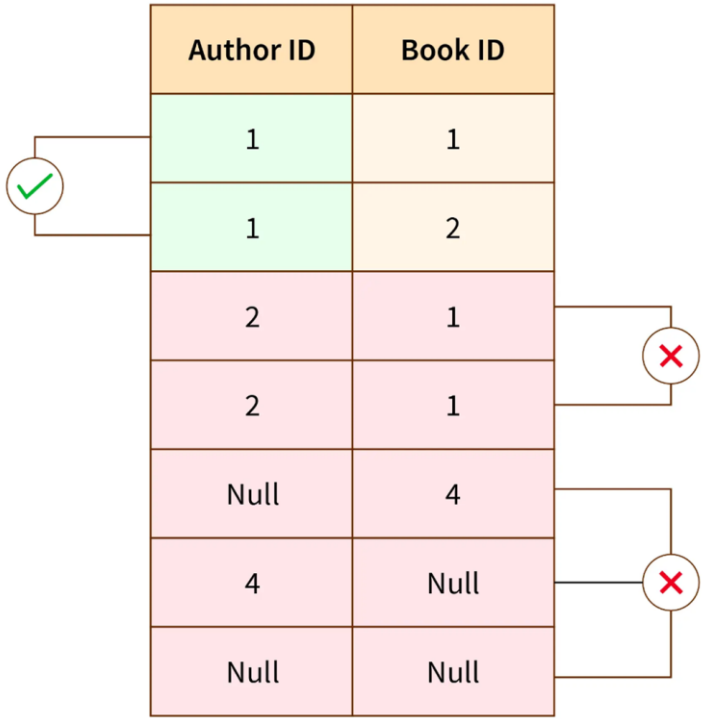
Composite Primary Key

A Primary Key made up of multiple columns

When a column is used as part

Of a composite primary key:

- That column can never be null
- That combination of those columns Must be Unique



Foreign Key

A column in a table, whose value is the primary key in another table.

The foreign key in the student table is: **group_id**

group		
id	name	grade
1	BG1	1
2	BG4	4
3	MG2	6

student				
id	name	surname	age	group_id
1	John	Brown	19	1
2	Emma	Carly	23	2
3	Kesha	Petruk	20	1

Relationships

A relationship / link between 2 tables.

Each student belongs to one group

Each student has a group_id

Groups can have Multiple Students

group		
id	name	grade
1	BG1	1
2	BG4	4
3	MG2	6

student				
id	name	surname	age	group_id
1	John	Brown	19	1
2	Emma	Carly	23	2
3	Kesha	Petruk	20	1

Database Design

Database Design

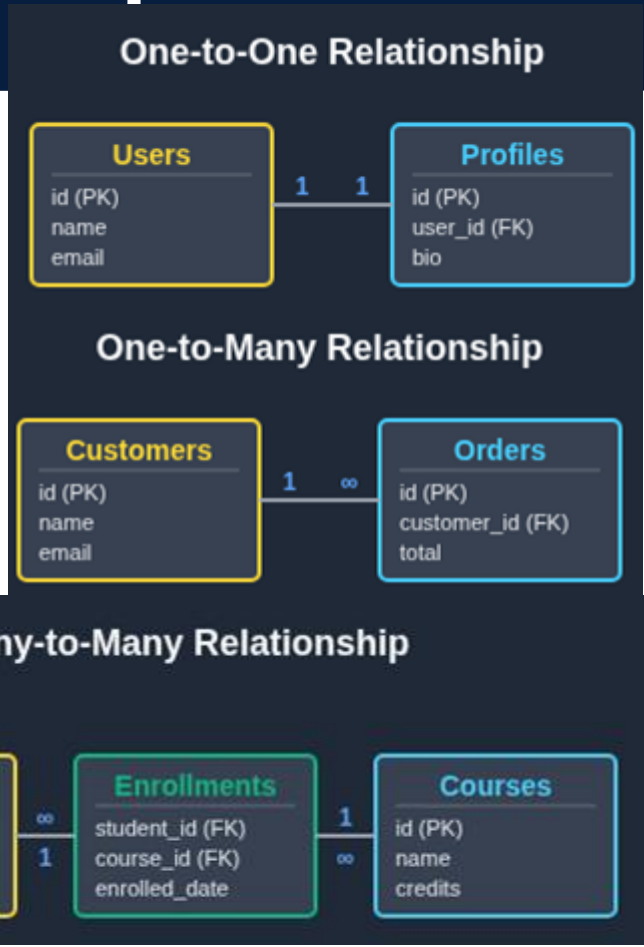
Database design is the process of **organizing data into structured formats (Tables)** so it can be **efficiently stored, retrieved, and managed**.

A well-designed database:

- Reduces redundancy
- Ensures data consistency
- Supports scalability
- Makes querying easier
- Reduces the cost of running the database

Types of Relationships

- **One-to-One (1:1)**
  - One record in A to One record in B.
  - Ex: One User has One UserProfile
- **One-to-Many (1:M)**
  - One record in A to Many records in B
  - One Customer has Many Orders
- **Many-to-Many (M:M)**
  - Many records in A to Many records
  - Students enroll in Many Courses
  - Courses have Many Students



Sample Problem

What are the Tables in this scenario?

Each student has a name, age.

Each student is scheduled for a class

Each class has an course code, room number, and instructor

Each instructor has a name, and email

Sample Problem

What are the Tables in this scenario?

Each **student** has a name, age.

Each student is **scheduled** for many Classes

Each **class** has an course code, room number, and instructor

Each **instructor** has a name, and email

Sample Problem

What are the Relationships in this scenario?

Each **student** has a name, age.

Each student is **scheduled** for many Classes

Each Class has has many scheduled students

Each **class** has an course code, room number, and instructor

Each Instructor teaches multiple classes

Each **instructor** has a name, and email

Sample Problem

What are the Relationships in this scenario?

Each **student** has a name, age.

Each student is **scheduled** for a class

Each Student is Scheduled for Many Classes (M-M)

Each Class is Scheduled to have Many Students (M-M)

Each **class** has an course code, room number, and instructor

Each class has one Instructor (1-1)

Each Instructor teaches multiple Classes (1-M)

Each **instructor** has a name, and email

Sample Problem

What are the Tables in this scenario?

Each person has a first name, last name, and one car

Each car has a name, make, and model

Cars are parked in parking Lots

Sample Problem

What are the Tables in this scenario?

Each **person** has a first name, last name, and one car

Each **car** has a name, make, and model

Cars are parked in **parking Lots**

Sample Problem

What are the Relationships in this Scenario

Each **person** has a first name, last name, and one car

Each **car** has a name, make, and model

Cars are parked in **parking Lots**

Sample Problem

What are the Relationships in this Scenario?

Each **person** has a first name, last name, and a car

Each Person has one Car (1-1)

Each Car has one Person (1-1)

Each **car** has a name, make, and model

Cars are parked in **parking Lots**

Each Car is parked in a Parking Lot (1-1)

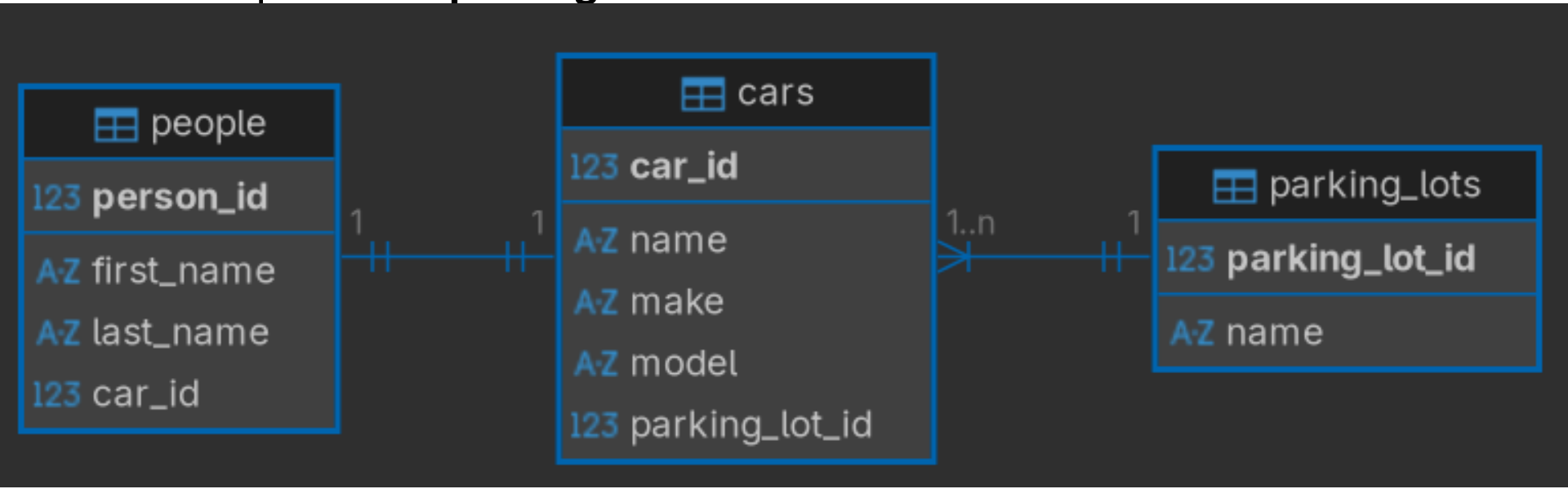
Each Parking Lot can have Many Cars (1-M)

Sample Problem

Each **person** has a first name, last name, and a car

Each **car** has a name, make, and model

Cars are parked in **parking Lots**

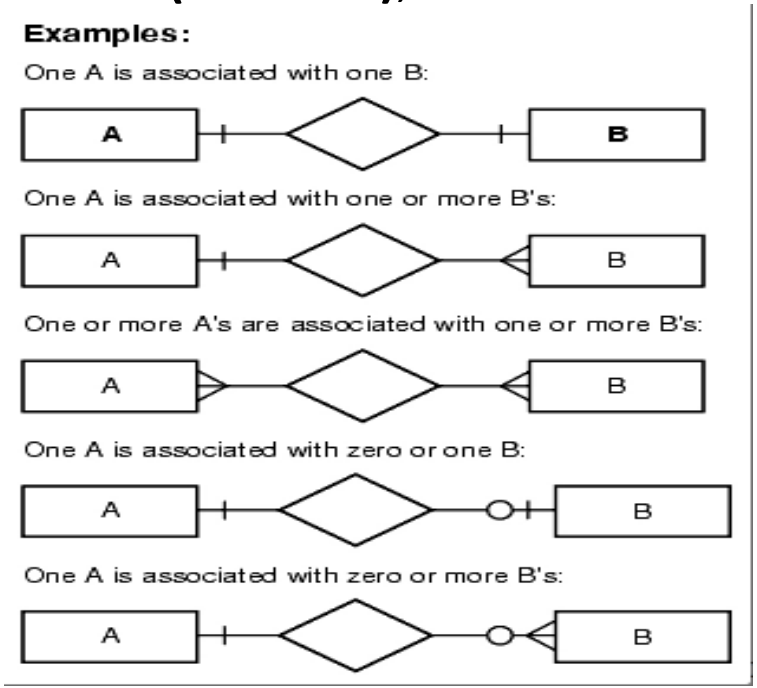


Entity Relationship Diagrams (ERD)

The Entity-Relationship Diagram (ERD) is a **diagram of a database**.

The diagram shows the **tables** in a database, their **fields (columns)**, and the **relationships** between those tables

Crow's Foot Notation



Crows Foot



Zero or one



Many



One



One (and only one)



Zero or many



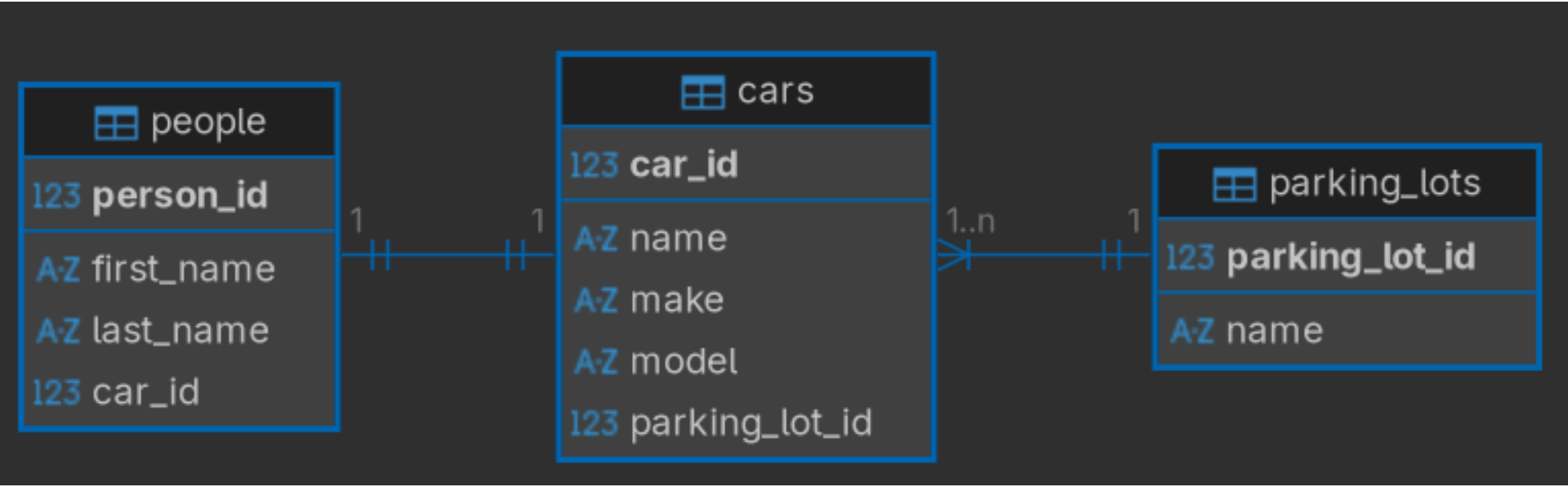
One or many

Sample Problem

Each **person** has a first name, last name, and a car

Each **car** has a name, make, and model

Cars are parked in **parking Lots**



Sample Problem

What are the Tables in this Scenario?

Each user has a username and email

Users can make Posts

Posts have text content

Each post can tag many people in it

Each post has comments

Comments have text content

Each comment, has replies

Sample Problem

What are the Tables in this Scenario?

Each **user** has a username and email

Users can make Posts

Posts have text content

Each post can tag many people in it

Each post has comments

Comments have text content

Each comment, has replies

Sample Problem

What are the Relationships in this Scenario?

Each **user** has a username and email

Users can make Posts

Posts have text content

Each post can tag many people in it

Each post has comments

Comments have text content

Each comment, has replies

Sample Problem

What are the Relationships in this Scenario?

Each **user** has a username and email

Users can make Posts

Posts have text content

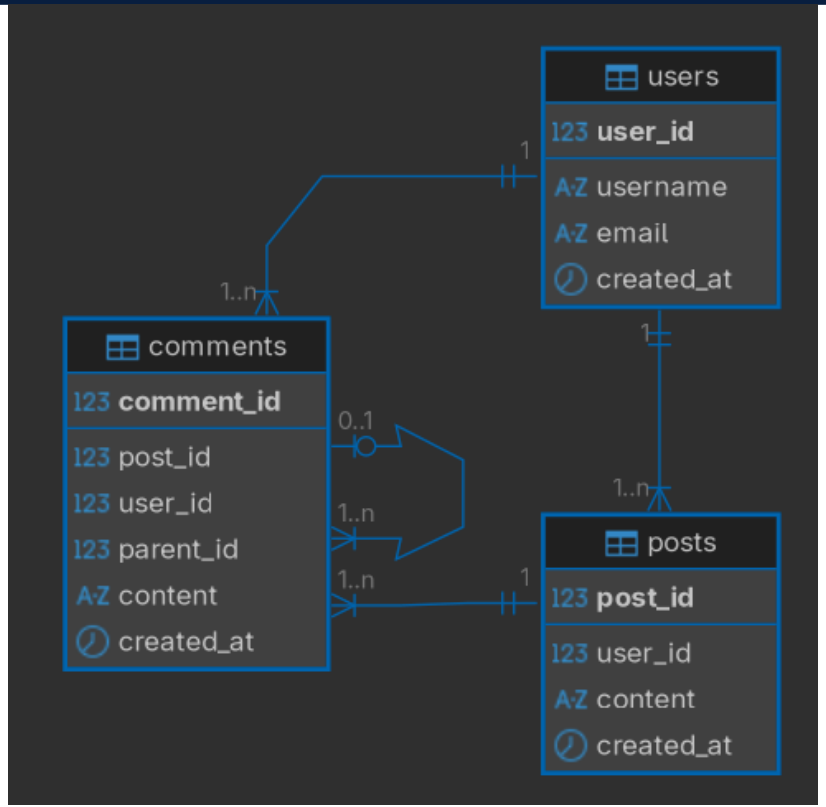
Each post can tag many people in it

Each post has comments

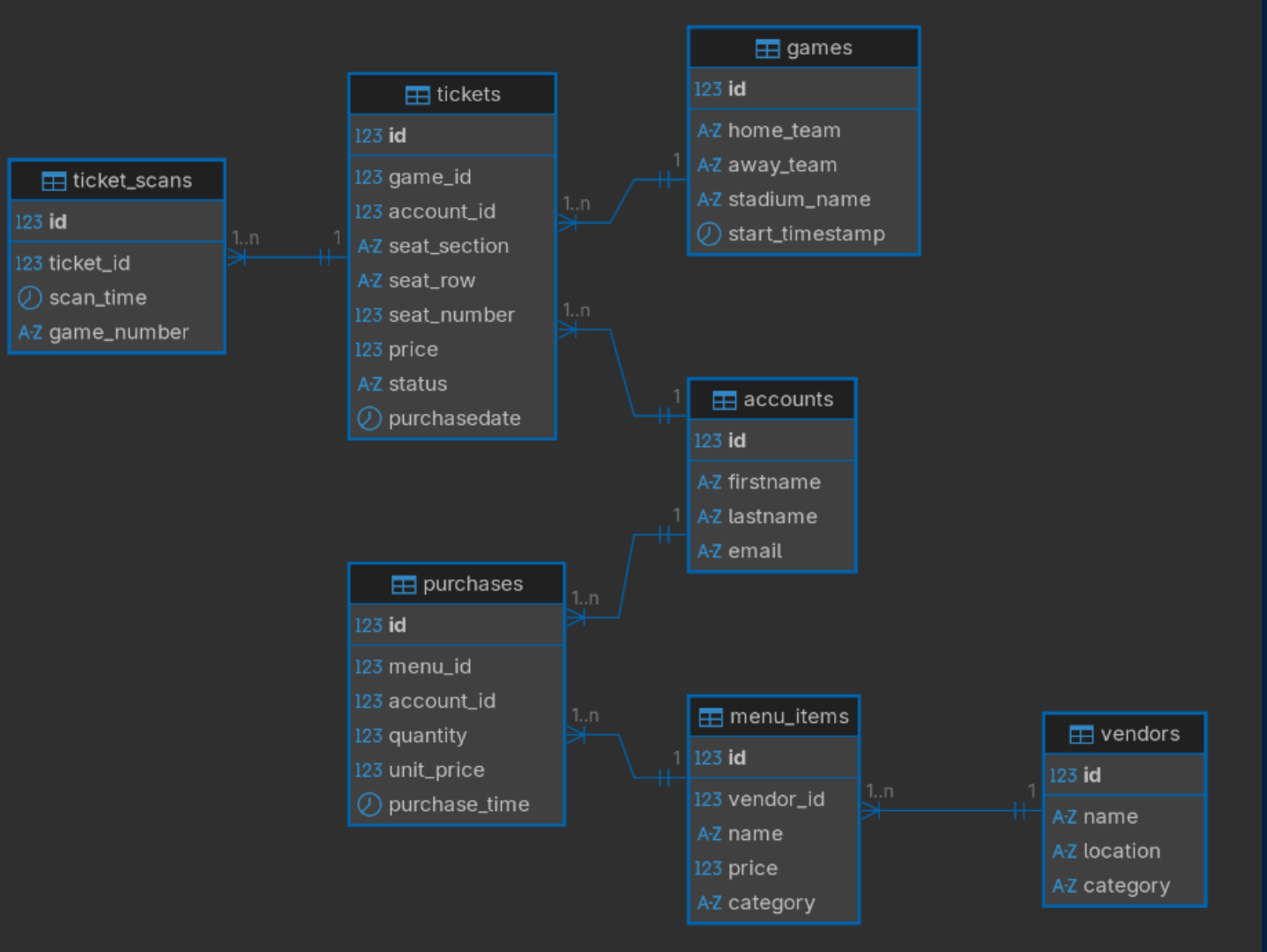
Comments have text content

Each comment, has replies

Sample Problem



In groups of 2-4 design a database containing information that can be collected from FIU


games
123 id
A-Z home_team
A-Z away_team
A-Z stadium_name
start_timestamp
tickets
123 id
123 game_id
123 account_id
A-Z seat_section
A-Z seat_row
123 seat_number
123 price
A-Z status
purchasedate
ticket_scans
123 id
123 ticket_id
scan_time
A-Z game_number
accounts
123 id
A-Z firstname
A-Z lastname
A-Z email
purchases
123 id
123 menu_id
123 account_id
123 quantity
123 unit_price
purchase_time
menu_items
123 id
123 vendor_id
A-Z name
123 price
A-Z category
vendors
123 id
A-Z name
A-Z location
A-Z category
1..n
1
1..n
1
1..n
1
1..n
1
1..n
1
1..n
1

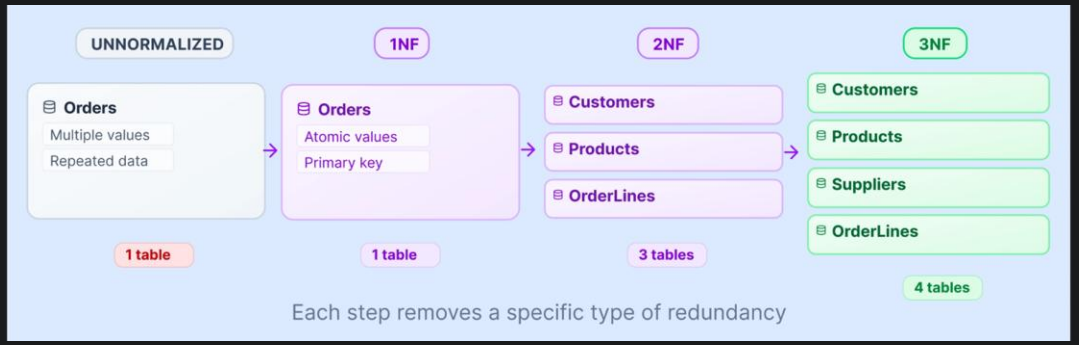
Normalization

What is Normalization?

Normalization is the process of organizing data into tables to reduce duplication and improve consistency.

A well normalized database:

- eliminates repeated data
- makes updates easier
- prevents inconsistencies
- improves long-term maintainability



Trade Offs

Pros	Cons
No redundant data	More tables to manage
Easier updates	Queries require more JOINs
Data stays consistent	Can be slower on large reads
Easier to scale	More complex to design upfront

Sometimes databases are intentionally denormalized for performance; this is a common in analytics and caching layers.

Unnormalized Table

record_id	student_id	student_name	student_email	student_age	course_code	course_name	course_room	inst_id	inst_name	inst_email	grade
1	1	Jordan Lee	jlee@student.edu	20	CS101	Intro to Python	A-204	1	Sarah Mitchell	smitchell@university.edu	A
2	1	Jordan Lee	jlee@student.edu	20	MATH201	Calculus I	B-110	2	James Thornton	jthornton@university.edu	B+
3	2	Maya Robinson	mrobinson@student.edu	19	CS101	Intro to Python	A-204	1	Sarah Mitchell	smitchell@university.edu	A-
4	2	Maya Robinson	mrobinson@student.edu	19	ENG105	Composition I	C-302	3	Linda Park	lpark@university.edu	B
5	3	Lucas Nguyen	lnguyen@student.edu	21	CS101	Intro to Python	A-204	1	Sarah Mitchell	smitchell@university.edu	B+
6	4	Ava Patel	apatel@student.edu	22	BIO150	General Biology	E-125	4	Elena Garcia	egarcia@university.edu	A-
7	5	Ethan Carter	ecarter@student.edu	20	ENG105	Composition I	C-302	3	Linda Park	lpark@university.edu	A

Student Data

Course Info

Instructor Data

In our unnormalized students table, we see several issues concerning data repetition.

First Normal Form: *1NF*

RULES:

- Atomic values (indivisible values)
- Each row is unique (no duplicates)
- Each column has a unique name.
- The order in which data is stored does not matter.

VIOLATIONS:

- `student_name`
- `instructor_name`

record_id	student_id	student_name	student_email	student_age	course_code	course_name	course_room	inst_id	inst_name	inst_email	grade
1	1	Jordan Lee	jlee@student.edu	20	CS101	Intro to Python	A-204	1	Sarah Mitchell	smitchell@university.edu	A
2	1	Jordan Lee	jlee@student.edu	20	MATH201	Calculus I	B-110	2	James Thornton	jthornton@university.edu	B+
3	2	Maya Robinson	mrobinson@student.edu	19	CS101	Intro to Python	A-204	1	Sarah Mitchell	smitchell@university.edu	A-
4	2	Maya Robinson	mrobinson@student.edu	19	ENG105	Composition I	C-302	3	Linda Park	lpark@university.edu	B
5	3	Lucas Nguyen	lnguyen@student.edu	21	CS101	Intro to Python	A-204	1	Sarah Mitchell	smitchell@university.edu	B+

Updated Design - *1NF*

record_id	student_id	student_first_name	student_last_name	student_email	student_age	course_code	course_name	course_room	inst_id	inst_first_name	inst_last_name	inst_email	grade
1	1	Jordan	Lee	jlee@student.edu	20	CS101	Intro to Python	A-204	1	Sarah	Mitchell	smitchell@university.edu	A
2	1	Jordan	Lee	jlee@student.edu	20	MATH201	Calculus I	B-110	2	James	Thornton	jthornton@university.edu	B+
3	2	Maya	Robinson	mrobinson@student.edu	19	CS101	Intro to Python	A-204	1	Sarah	Mitchell	smitchell@university.edu	A-
4	2	Maya	Robinson	mrobinson@student.edu	19	ENG105	Composition I	C-302	3	Linda	Park	lpark@university.edu	B
5	3	Lucas	Nguyen	lnguyen@student.edu	21	CS101	Intro to Python	A-204	1	Sarah	Mitchell	smitchell@university.edu	B+
6	4	Ava	Patel	apatel@student.edu	22	BIO150	General Biology	E-125	4	Elena	Garcia	egarcia@university.edu	A-
7	5	Ethan	Carter	ecarter@student.edu	20	ENG105	Composition I	C-302	3	Linda	Park	lpark@university.edu	A

*By splitting student and instructor **name** columns into **first_name** and **last_name,** we have entered the 1st Normal Form*

Second Normal Form: *2NF*

RULES:

- Must meet 1NF
- No partial dependencies on composite keys or on a non-key column
  - Meaning, every non-key column depends on the **whole PK**

VIOLATIONS:

1. Student information only depends on **`student_id`**
2. Course information only depends on **`course_code`**
3. Instructor information only depends on **`instructor_id`**

record_id	student_id	student_name	student_surname	student_email	student_age	course_code	course_name	course_room	inst_id	inst_name	inst_surname	inst_email	grade
1	1	Jordan	Lee	jlee@student.edu	20	CS101	Intro to Python	A-204	1	Sarah	Mitchell	smitchell@university.edu	A
2	1	Jordan	Lee	jlee@student.edu	20	MATH201	Calculus I	B-110	2	James	Thornton	jthornton@university.edu	B+
3	2	Maya	Robinson	mrobinson@student.edu	19	CS101	Intro to Python	A-204	1	Sarah	Mitchell	smitchell@university.edu	A-

Splitting Things Up - *2NF*

record_id	student_id	student_name	student_surname	student_email	student_age	course_code	course_name	room_number	inst_id	inst_name	inst_surname	inst_email	grade
1	1	Jordan	Lee	jlee@student.edu	20	CS101	Intro to Python	A-204	1	Sarah	Mitchell	smitchell@university.edu	A

students

first_name	text
last_name	text
email	text
age	integer
student_id	integer

instructors

first_name	text
last_name	text
email	text
instructor_id	integer

courses

course_name	text
course_code	text

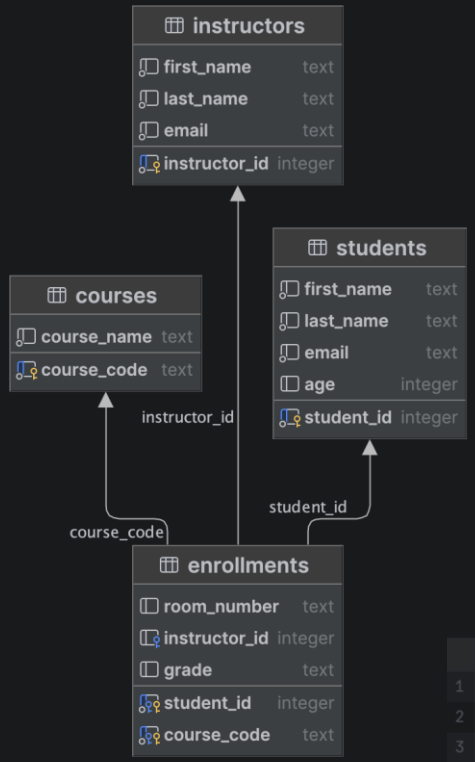
After extracting the repeated entity data, the remaining columns (**`course_code, room_number, instructor_id, and grade`**) stay together in an **enrollments** table, using (**`student_id, course_code`**) as a composite key.

This table is what ties everything together. Without it, we lose the actual enrollment records from the original data.

enrollments

room_number	text
instructor_id	integer
grade	text
student_id	integer
course_code	text

Updated Design - *2NF*



Courses

	instructor_id	first_name	last_name	email
1	1	Sarah	Mitchell	smitchell@university.edu
2	2	James	Thornton	jthornton@university.edu
3	3	Linda	Park	lpark@university.edu
4	4	Elena	Garcia	egarcia@university.edu

Enrollments

	student_id	course_code	room_number	instructor_id	grade
1	1	CS101	A-204	1	A
2	1	MATH201	B-110	2	B+
3	2	CS101	A-204	1	A-
4	2	ENG105	C-302	3	B

Instructors

	instructor_id	first_name	last_name	email
1	1	Sarah	Mitchell	smitchell@university.edu
2	2	James	Thornton	jthornton@university.edu
3	3	Linda	Park	lpark@university.edu
4	4	Elena	Garcia	egarcia@university.edu

Courses

	course_code	course_name
1	CS101	Intro to Python
2	MATH201	Calculus I
3	ENG105	Composition I
4	BIO150	General Biology

Third Normal Form: *3NF*

RULES:

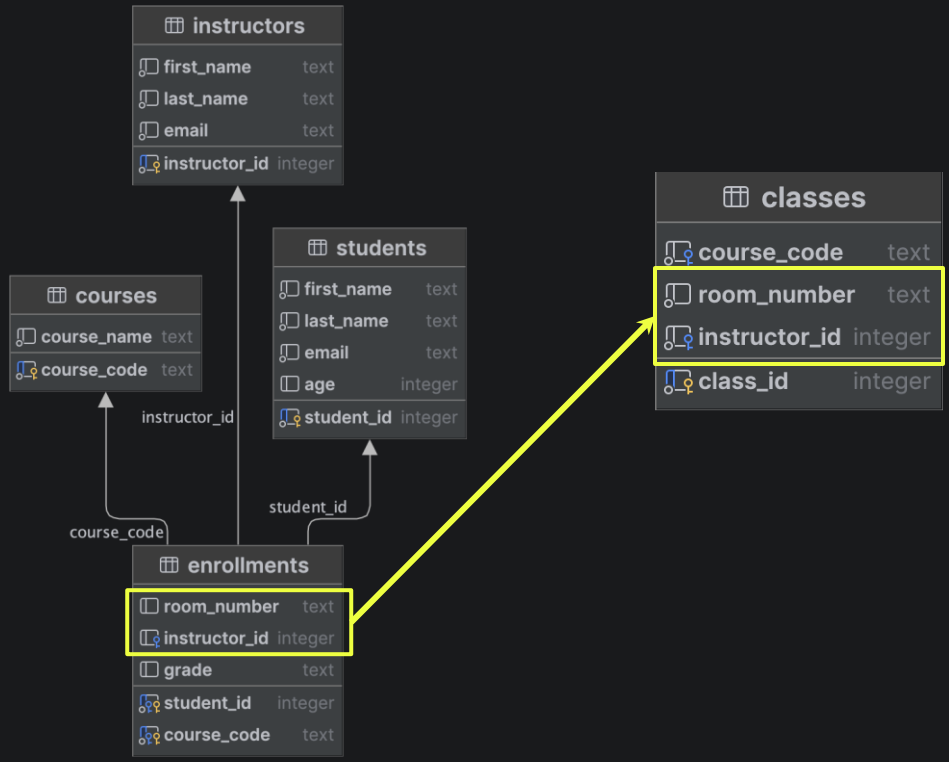
- Must be already in 2NF
- No *transitive dependencies*
  - Meaning, non-key columns shouldn't depend on other non-key columns

VIOLATIONS:

`room_number` and `instructor_id` are tied to a specific class section, *not* just the course.

ACTION:

Create **classes** table to separate course sections from course definitions



Normalized School Output

Students

	student_id	first_name	last_name	email	age
1	1	Jordan	Lee	jlee@student.edu	20
2	2	Maya	Robinson	mrobinson@student.edu	19
3	3	Lucas	Nguyen	lnguyen@student.edu	21
4	4	Ava	Patel	apatel@student.edu	22
5	5	Ethan	Carter	ecarter@student.edu	20

Instructors

	instructor_id	first_name	last_name	email
1	1	Sarah	Mitchell	smitchell@university.edu
2	2	James	Thornton	jthornton@university.edu
3	3	Linda	Park	lpark@university.edu
4	4	Elena	Garcia	egarcia@university.edu

Classes

	class_id	course_code	room_number	instructor_id
1	1	CS101	A-204	1
2	2	MATH201	B-110	2
3	3	ENG105	C-302	3
4	4	BIO150	E-125	4

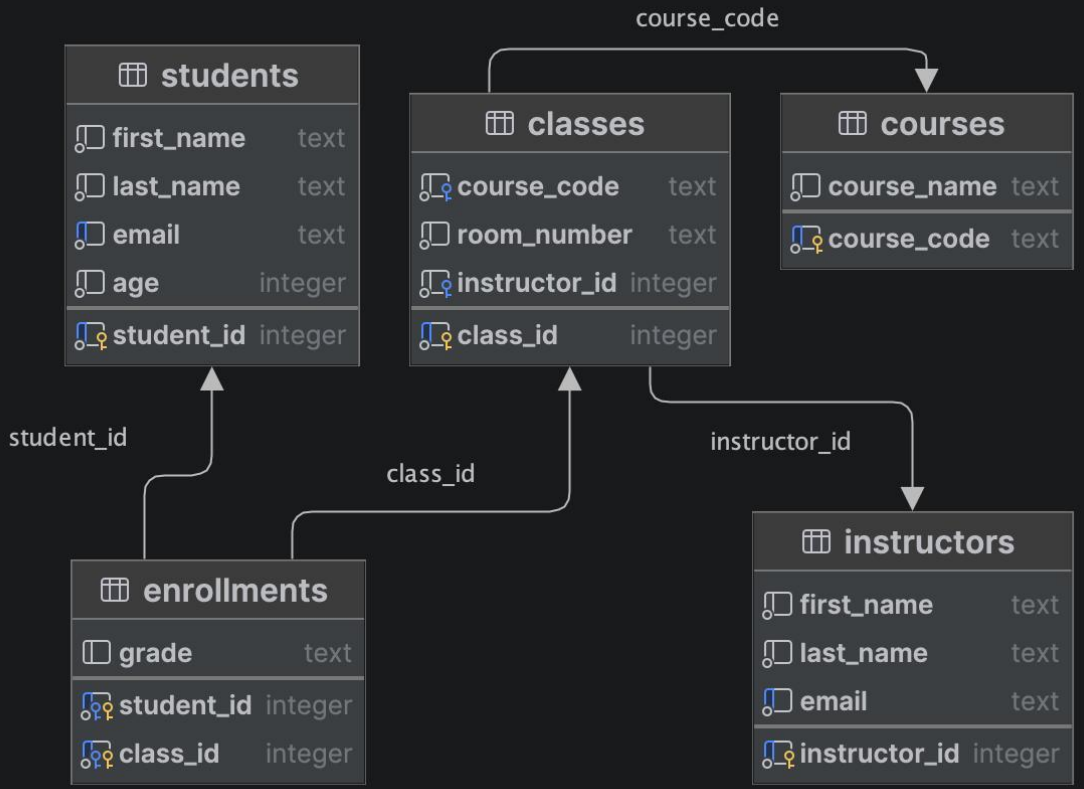
Courses

	course_code	course_name
1	BIO150	General Biology
2	CS101	Intro to Python
3	ENG105	Composition I
4	MATH201	Calculus I

Enrollments

	student_id	class_id	grade
1	1	1	A
2	1	2	B+
3	2	1	A-
4	2	3	B
5	3	1	B+
6	4	4	A-
7	5	3	A

Normalized School ERD

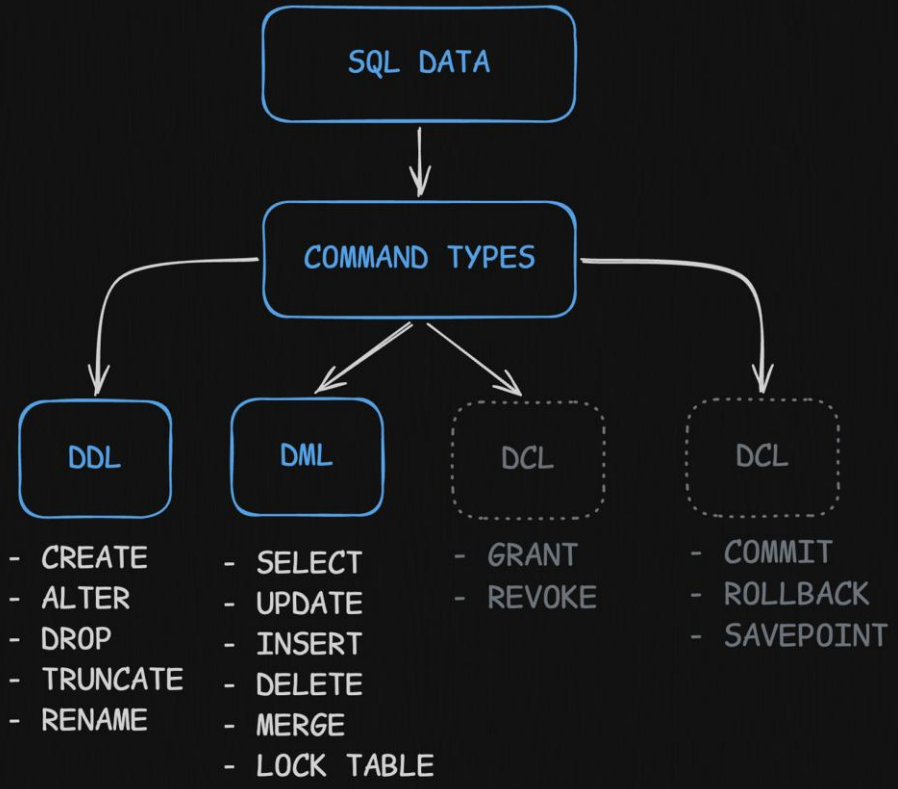


What is SQL?

SQL: *Structured Query Language*

SQL is the language used to create, manage, and query relational databases. Commands are grouped into 4 categories.

*For building a normalized schema, we primarily use **DDL** to define structure and **DML** to work with data.*

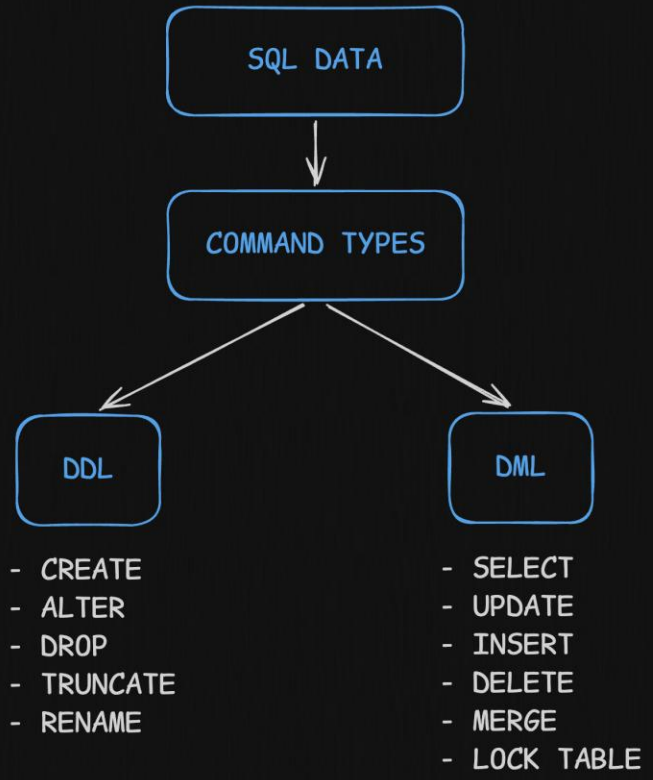


DDL and DML

DDL (Data Definition Language) and DML (Data Manipulation Language) will be the primary focus of these workshops.

DDL commands define the *structure* of your database. This is how we create our normalized tables.

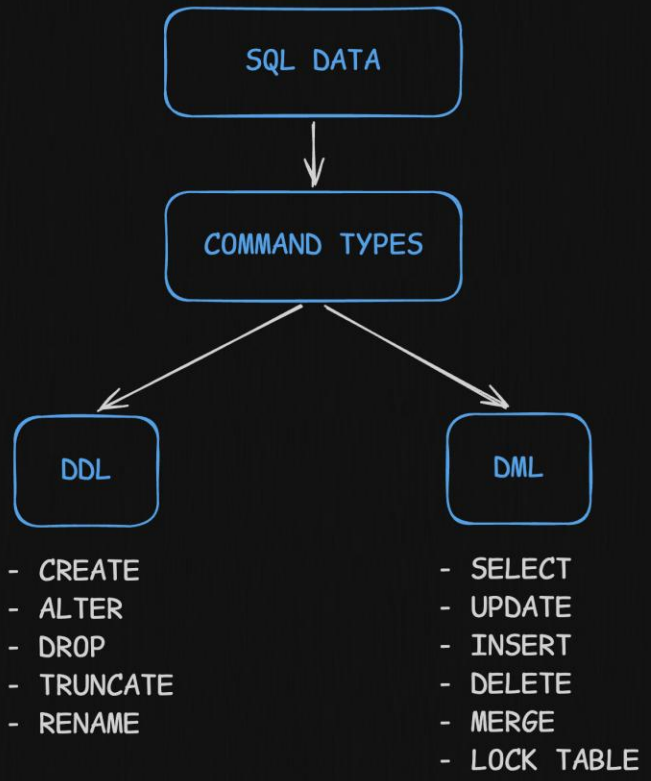
DML commands let you read and modify data inside your tables.



Activity 1

Using your ERD from the first activity, create the tables for the FIU Business Department.

- Consider useful columns
- Consider tradeoffs between normalization and denormalization
- Consider useful queries and the tables that are involved in those.



DDL

Data Definition Language

CREATE TABLE

In PostgreSQL, tables are created with:

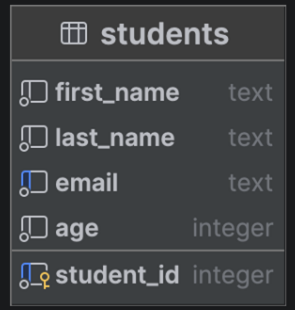
CREATE TABLE

Key components:

- table name
- columns
- data types
- constraints

```sql
CREATE TABLE IF NOT EXISTS students
(
    student_id SERIAL PRIMARY KEY,
    first_name TEXT    NOT NULL,
    last_name  TEXT    NOT NULL,
    email      TEXT    NOT NULL UNIQUE,
    age        INTEGER NOT NULL
);
```

Results in



And output looks like:

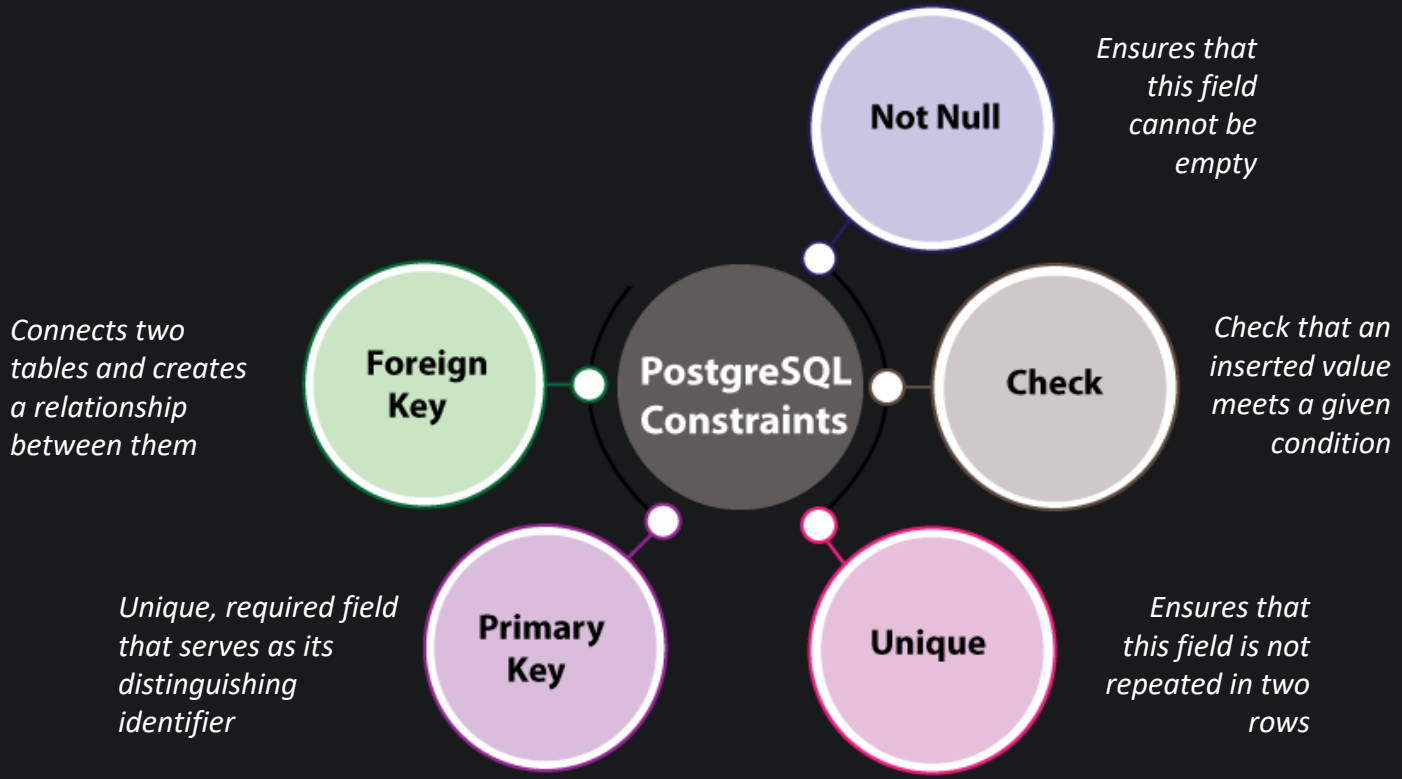
	student_id	first_name	last_name	email	age
1	1	Jordan	Lee	jlee@student.edu	20
2	2	Maya	Robinson	mrobinson@student.edu	19
3	3	Lucas	Nguyen	lnguyen@student.edu	21
4	4	Ava	Patel	apatel@student.edu	22
5	5	Ethan	Carter	ecarter@student.edu	20

Common Data Types

Category	Data Type	Description	Example Use
Numeric	`INTEGER`, `BIGINT`	Whole numbers of varying sizes	Unique IDs, counts
	`NUMERIC`, `DECIMAL`	Fixed-point numbers with precision	Balances, financial data
Character	`VARCHAR`, `TEXT`	Variable-length text	Names, descriptions
Date/Time	`DATE`, `TIMESTAMP`	Dates and timestamps	Event dates, logs
Boolean	`BOOLEAN`	True/False values	Status flags
Array	`ARRAY`	List of values of the same type	List of tags, multiple emails
Special	`UUID`, `JSON`	Unique identifiers, JSON data	User IDs, product attributes
Geometric	`POINT`, `LINE`	Geometric data types for coordinates	Location data

*

Constraints



DEFAULT and CHECK

DEFAULT: automatically fills a value for the column if none is given

CHECK: Validates that inserted data meets a condition (such as **`age >= 16`**)

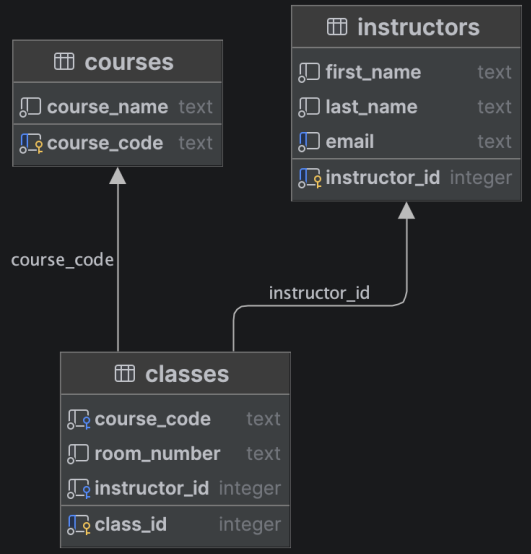
```
CREATE TABLE students (
    student_id SERIAL PRIMARY KEY,
    first_name TEXT NOT NULL,
    last_name  TEXT NOT NULL,
    email      TEXT NOT NULL UNIQUE,
    age        INTEGER NOT NULL CHECK (age >= 16),
    status     TEXT DEFAULT 'active'
);
```

```
-- This would be REJECTED by the CHECK constraint:
INSERT INTO students (first_name, last_name, email, age)
VALUES ('Sam', 'Reed', 'sreed@student.edu', 14);
```

Foreign Keys

Foreign keys connect one table to another. In our example, we indicate that each class must have a valid instructor and course.

```
CREATE TABLE IF NOT EXISTS classes
(
    class_id      SERIAL PRIMARY KEY,
    course_code   TEXT    NOT NULL,
    room_number   TEXT    NOT NULL,
    instructor_id INTEGER NOT NULL,
    FOREIGN KEY (course_code) REFERENCES courses (course_code),
    FOREIGN KEY (instructor_id) REFERENCES instructors (instructor_id)
);
```



1-1 and 1-N

To implement a many-to-one relationship, it's just a plain foreign key *without* a **UNIQUE** constraint.

```
CREATE TABLE IF NOT EXISTS classes
(
    class_id       SERIAL PRIMARY KEY,
    course_code    TEXT NOT NULL,
    instructor_id  INTEGER NOT NULL,
    FOREIGN KEY (instructor_id) REFERENCES instructors (instructor_id)
);
```

*Many classes -> one instructor. Multiple rows in classes can share the same **instructor_id**.*

Each row in one table maps to exactly one row in another. Enforce with a FOREIGN KEY that is also UNIQUE.

```
CREATE TABLE IF NOT EXISTS student_profiles
(
    profile_id   SERIAL PRIMARY KEY,
    student_id   INTEGER NOT NULL UNIQUE,
    bio          TEXT,
    gpa          DECIMAL(3,2),
    FOREIGN KEY (student_id) REFERENCES students (student_id)
);
```

*The **UNIQUE** constraint on **student_id** is what makes it one-to-one: without it, multiple profiles could reference the same student (making it many-to-one).*

Many-to-Many Relationships (FK)

We can create the table same as before:, but the key is that we have the two FKs as our composite PK.

Students can take many classes.

Classes can have many students

```sql
CREATE TABLE IF NOT EXISTS enrollments
(
    student_id INTEGER NOT NULL,
    class_id   INTEGER NOT NULL,
    grade      TEXT,
    PRIMARY KEY (student_id, class_id),
    FOREIGN KEY (student_id) REFERENCES students (student_id),
    FOREIGN KEY (class_id) REFERENCES classes (class_id)
);
```

a **junction table**.

ON DELETE

Action	When Parent is Deleted	Use Case
CASCADE	Child is deleted too	Enrollments when student graduates
SET NULL	FK becomes NULL	Course instructor retires
RESTRICT	Block deletion of parent if child(s) are referencing it	Course with enrolled students

ALTER and DROP

ALTER TABLE: Modify an existing table's structure.

```
-- Add a column
ALTER TABLE students ADD COLUMN gpa DECIMAL;

-- Remove a column
ALTER TABLE students DROP COLUMN age;

-- Rename a column
ALTER TABLE students RENAME COLUMN email TO student_email;

-- Change data type
ALTER TABLE students ALTER COLUMN gpa TYPE NUMERIC(3,2);
```

DROP TABLE: permanently deletes a table and all its data.

```
-- WARNING: DROP is irreversible.
-- Drop child tables (with foreign keys)
-- BEFORE parent tables.
DROP TABLE IF EXISTS enrollments;
```

Activity 1

Take the ERD you designed last class and write the **CREATE TABLE** statements for your FIU Business database.

For each table:

- Choose appropriate data types (INTEGER, TEXT, DECIMAL, TIMESTAMP, etc.)
- Add a PRIMARY KEY
- Add constraints where they make sense (NOT NULL, UNIQUE, CHECK, DEFAULT)
- Define FOREIGN KEYs that match your ERD's relationships
- Choose an ON DELETE strategy for each foreign key, explain why

TIP: Start with tables that have no foreign keys (e.g. games, accounts, vendors), then work outward to tables that depend on them (tickets, menu_items, purchases, ticket_scans).

Deliverable: Complete CREATE TABLE statements that could run in PostgreSQL without errors.

DML

Data Manipulation Language

SELECT

Used to retrieve data from one or more tables. Can choose fields or use `*` for all fields.

```
SELECT first_name, last_name
FROM students;
```

	first_name	last_name
1	Jordan	Lee
2	Maya	Robinson
3	Lucas	Nguyen
4	Ava	Patel
5	Ethan	Carter

```
SELECT * FROM students;
```

	student_id	first_name	last_name	email	age
1	1	Jordan	Lee	jlee@student.edu	20
2	2	Maya	Robinson	mrobinson@student.edu	19
3	3	Lucas	Nguyen	lnguyen@student.edu	21
4	4	Ava	Patel	apatel@student.edu	22
5	5	Ethan	Carter	ecarter@student.edu	20

WHERE

Filters rows based on a condition.

```
SELECT * FROM students
WHERE last_name = 'Lee'
   OR last_name = 'Patel';
```

OUTPUT:

	id	first_name	last_name	email	age
1	1	Jordan	Lee	jlee@student.edu	20
2	4	Ava	Patel	apatel@student.edu	22

Combine conditions with **AND / OR**

```
SELECT * FROM students
WHERE age >= 20
  AND last_name = 'Lee';
```

; ro

```
WHERE age > 20;
```

Use parentheses to control logic

OUTPUT:

	student_id	first_name	last_name	email	age
1	3	Lucas	Nguyen	lnguyen@student.edu	21
2	4	Ava	Patel	apatel@student.edu	22

```
AND email LIKE '%@student.edu';
```

OUTPUT:

	id	first_name	last_name	email	age
1	1	Jordan	Lee	jlee@student.edu	20
2	3	Lucas	Nguyen	lnguyen@student.edu	21
3	4	Ava	Patel	apatel@student.edu	22

Filtering

OFFSET	ORDER BY	LIMIT

`-- Page 2: skip first 10,`
`-- show next 10`
`SELECT * FROM students`
`ORDER BY student_id`
`LIMIT 10 OFFSET 10;`
	`-- Sort students by`
	`-- age then last name`
	`SELECT * FROM students`
	`ORDER BY age DESC,`
	`last_name ASC;`
		`-- Top 3 oldest students`
		`SELECT * FROM students`
		`ORDER BY age DESC`
		`LIMIT 3;`

Skip rows (this is useful for pagination for example).	*Use to sort query results, default is **ASC***	*Restrict the output of a query*

LIKE and Wildcards

LIKE uses two wildcards for pattern matching:

Wildcard	Meaning	Example	Matches
%	Any number of characters	'P%'	Patel, Powell, Park
_	Exactly one character	'_ava'	Java, Cava, etc.

EXAMPLE:

```
SELECT * FROM students
WHERE email
LIKE '%@student.edu';
```

OUTPUT:

	id	first_name	last_name	email	age
1	1	Jordan	Lee	jlee@student.edu	20
2	2	Maya	Robinson	mrobinson@student.edu	19
3	3	Lucas	Nguyen	lnguyen@student.edu	21
4	4	Ava	Patel	apatel@student.edu	22
5	5	Ethan	Carter	ecarter@student.edu	20

More operators

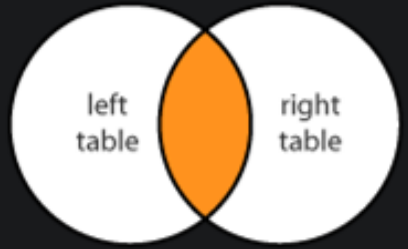
Operator	Example
`BETWEEN`	`WHERE age BETWEEN 18 AND 25`
`IN`	`WHERE last_name IN ('Lee', 'Patel')`
`IS NULL`	`WHERE grade IS NULL`
`IS NOT NULL`	`WHERE grade IS NOT NULL`
`<> or !=`	`WHERE status <> 'inactive'`

Types of JOINs

A JOIN combines rows from two tables
based on a related column.

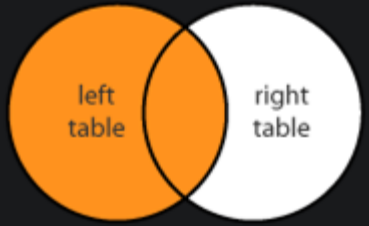
INNER JOIN (JOIN)

Returns only rows that match in **both** tables.



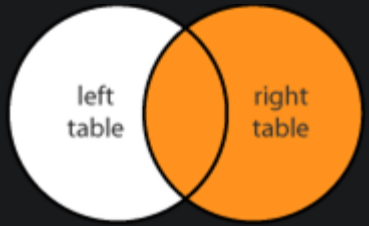
LEFT JOIN

Returns all rows from the **left** table. NULLs where no match on right.



RIGHT JOIN

Returns all rows from the **right** table. NULLs where no match on left.



JOIN Examples

INNER JOIN

```sql
SELECT s.first_name, e.grade
FROM students s
    INNER JOIN enrollments e
    ON s.id = e.student_id;
```

1	Jordan	A
2	Jordan	B+
3	Maya	A-
4	Maya	B
5	Lucas	B+
6	Ava	A-
7	Ethan	A

Students with NO enrollments are excluded

LEFT JOIN

```sql
SELECT s.first_name, e.grade
FROM students s
    LEFT JOIN enrollments e
    ON s.id = e.student_id;
```

13	Priya	B
14	Priya	A-
15	Noah	B
16	Noah	C
17	Camila	A
18	Camila	B+
19	Lucas	<null>
20	Ethan	<null>

Students with no enrollments show NULL for grade

RIGHT JOIN

```sql
SELECT s.first_name, e.grade
FROM students s
    RIGHT JOIN enrollments e
    ON s.id = e.student_id;
```

15	Intro to Statistics	9
16	General Biology	9
17	Intro to Python	10
18	Composition I	10
19	Intro to Statistics	3
20	Data Structures	5
21	Calculus I	<null>

Enrollments with no matching student show NULL for name

Join Across Many Tables

```sql
SELECT s.first_name,
       s.last_name,
       co.course_code,
       co.course_name,
       i.first_name AS instructor_first_name,
       i.last_name  AS instructor_last_name,
       c.room_number,
       e.grade
FROM students s
        JOIN enrollments e
             1<->1..n: ON s.student_id = e.student_id
        JOIN classes c
             1..n<->1: ON e.class_id = c.class_id
        JOIN courses co
             1..n<->1: ON c.course_code = co.course_code
        JOIN instructors i
             1..n<->1: ON c.instructor_id = i.instructor_id
ORDER BY s.last_name, s.first_name, co.course_code;
```

	·	la	c	course_…		in.		grade
1	Ethan	Carter	ENG105	Composition I	Linda	Park	C-302	A
2	Jordan	Lee	CS101	Intro to Python	Sarah	Mitchell	A-204	A
3	Jordan	Lee	MATH201	Calculus I	James	Thornton	B-110	B+
4	Lucas	Nguyen	CS101	Intro to Python	Sarah	Mitchell	A-204	B+
5	Ava	Patel	BIO150	General Biology	Elena	Garcia	E-125	A-
6	Maya	Robinson	CS101	Intro to Python	Sarah	Mitchell	A-204	A-
7	Maya	Robinson	ENG105	Composition I	Linda	Park	C-302	B

In this example, we join across:

enrollments -> classes -> courses -> instructors.

This lets us see all enrollments, including the student, class, and course information as well.

Aggregate

Aggregate functions summarize data.

Function	What It Does	Example
COUNT()	Number of rows	COUNT(student_id)
SUM()	Total	SUM(credits)
AVG()	Average	AVG(age)
MIN()	Smallest value	MIN(age)
MAX()	Largest value	MAX(age)

Aggregate Example

Use GROUP BY to aggregate per group.

Use HAVING to filter groups.

```sql
SELECT c.course_code,
       COUNT(e.student_id) AS num_students,
       AVG(s.age)          AS avg_age
FROM classes c
    LEFT JOIN enrollments e ON c.class_id = e.class_id
    LEFT JOIN students s    ON e.student_id = s.student_id
GROUP BY c.course_code
HAVING COUNT(e.student_id) > 1
ORDER BY num_students DESC;
```



	course_code	num_students	avg_age
1	CS101	5	19.8
2	ENG105	3	19.6666666666…
3	STAT300	3	21.6666666666…
4	CS202	3	20.3333333333…
5	BIO150	2	21.5
6	HIST101	2	18.5
7	MATH201	2	21.5

Subqueries

A query nested inside another query. The inner query runs first, and its result is used by the outer query.

```sql
-- Students older than average
SELECT first_name, last_name, age
FROM students
WHERE age > (SELECT AVG(age) FROM students);
```



	first_name	last_name	age
1	Lucas	Nguyen	21
2	Ava	Patel	22
3	Liam	Chen	23
4	Noah	Brooks	21

INSERT, UPDATE, DELETE

INSERT: add new rows to a table.

- Column order in **INSERT** must match **VALUES** order
- Columns with **DEFAULT** or **SERIAL** can be omitted
- **NOT NULL** columns without a **DEFAULT** must be included

UPDATE: modify existing rows.
DELETE: remove rows

*Without **WHERE**, every row is affected on **UPDATE** or **DELETE***

```sql
-- INSERT a new row
INSERT INTO students (first_name, last_name, email, age)
VALUES ('Sofia', 'Martinez', 'smartinez@student.edu', 20);

-- INSERT multiple rows
INSERT INTO students (first_name, last_name, email, age)
VALUES
    ('Alex', 'Kim', 'akim@student.edu', 21),
    ('Priya', 'Shah', 'pshah@student.edu', 19);

-- UPDATE existing rows
UPDATE students SET age = 21 WHERE student_id = 1;

-- DELETE rows
DELETE FROM students WHERE student_id = 5;
```

Activity 2

Write at least 4 queries against your schema.
Two are given:

1. Total ticket revenue per game.
2. Find all accounts who purchased a ticket but never had it scanned.
    a. Must use: JOIN or subquery, NULL check.

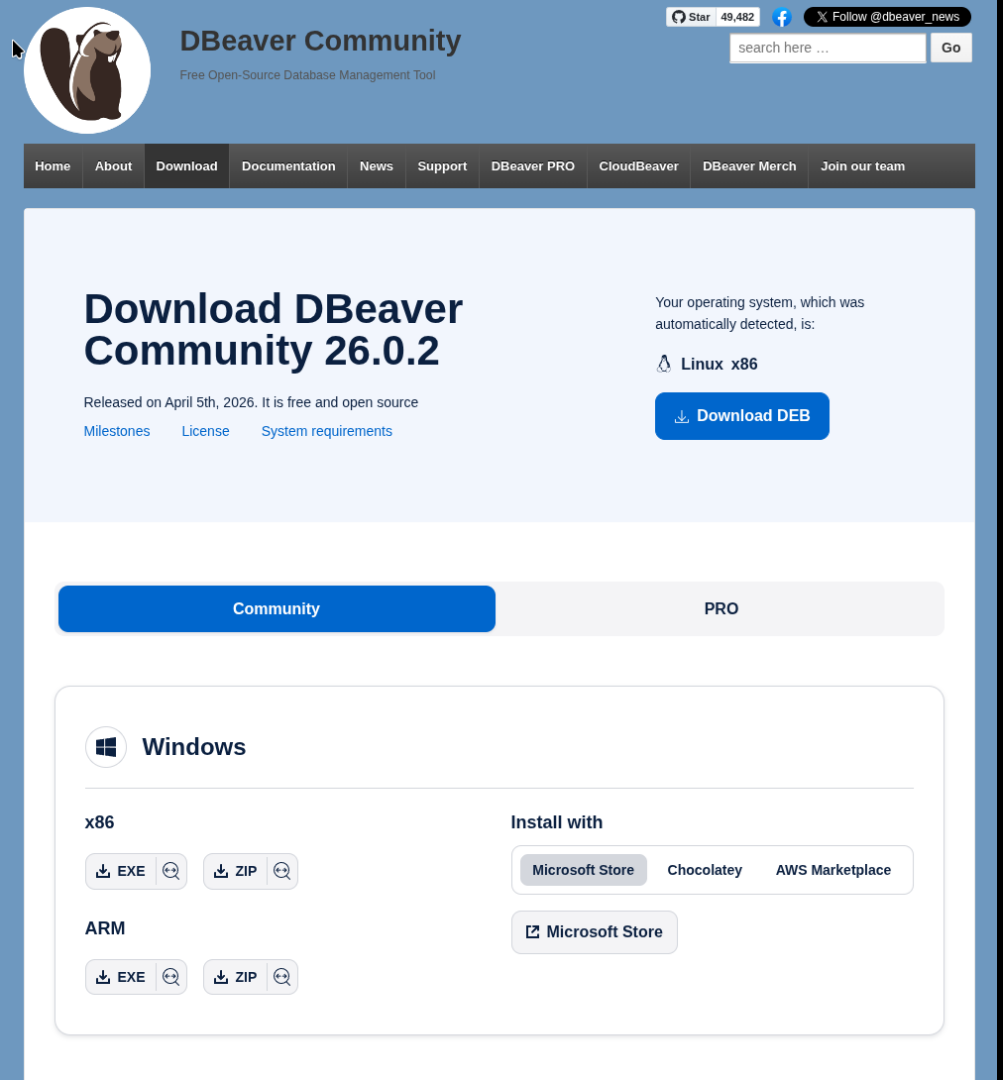
Two are of you own design; write queries that would be valuable to the FIU Business Department:

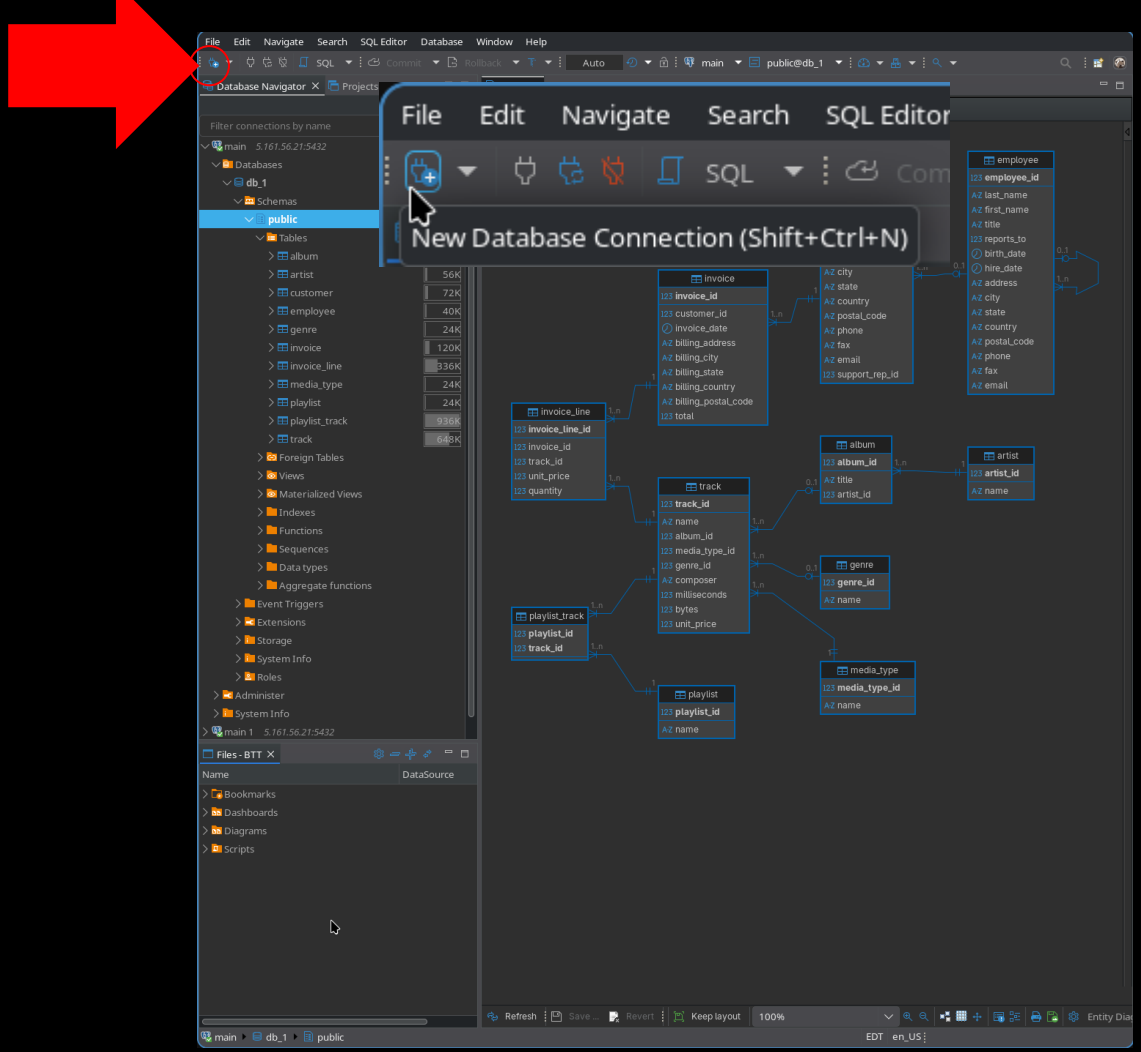
1. Must include at least two JOIN.
2. Must include a subquery, HAVING, or an aggregate.

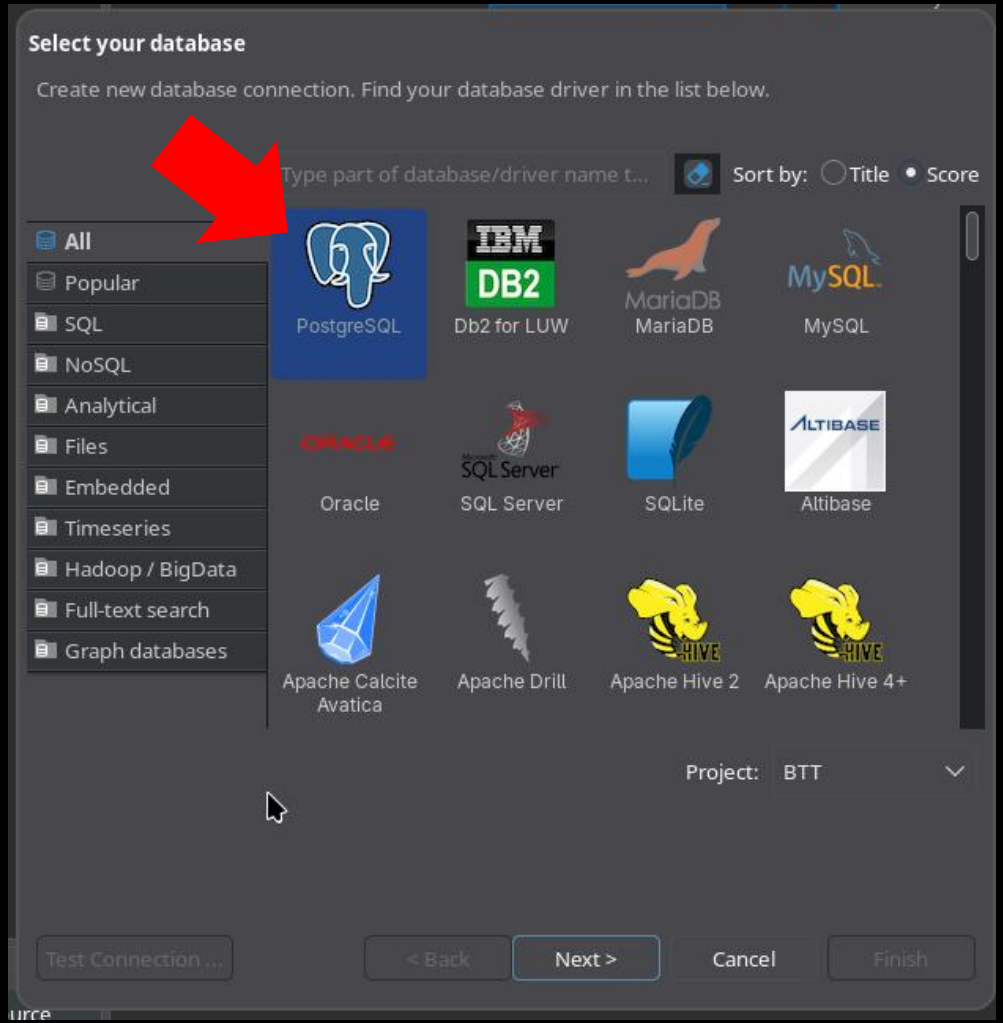
Think like a stakeholder: What would the business office want to know? Top-selling vendors, average spend per account, unsold seat sections, busiest purchase times, food revenue vs ticket revenue.
Deliverable: 4+ SQL queries, each with a one-sentence description of the business question it answers.

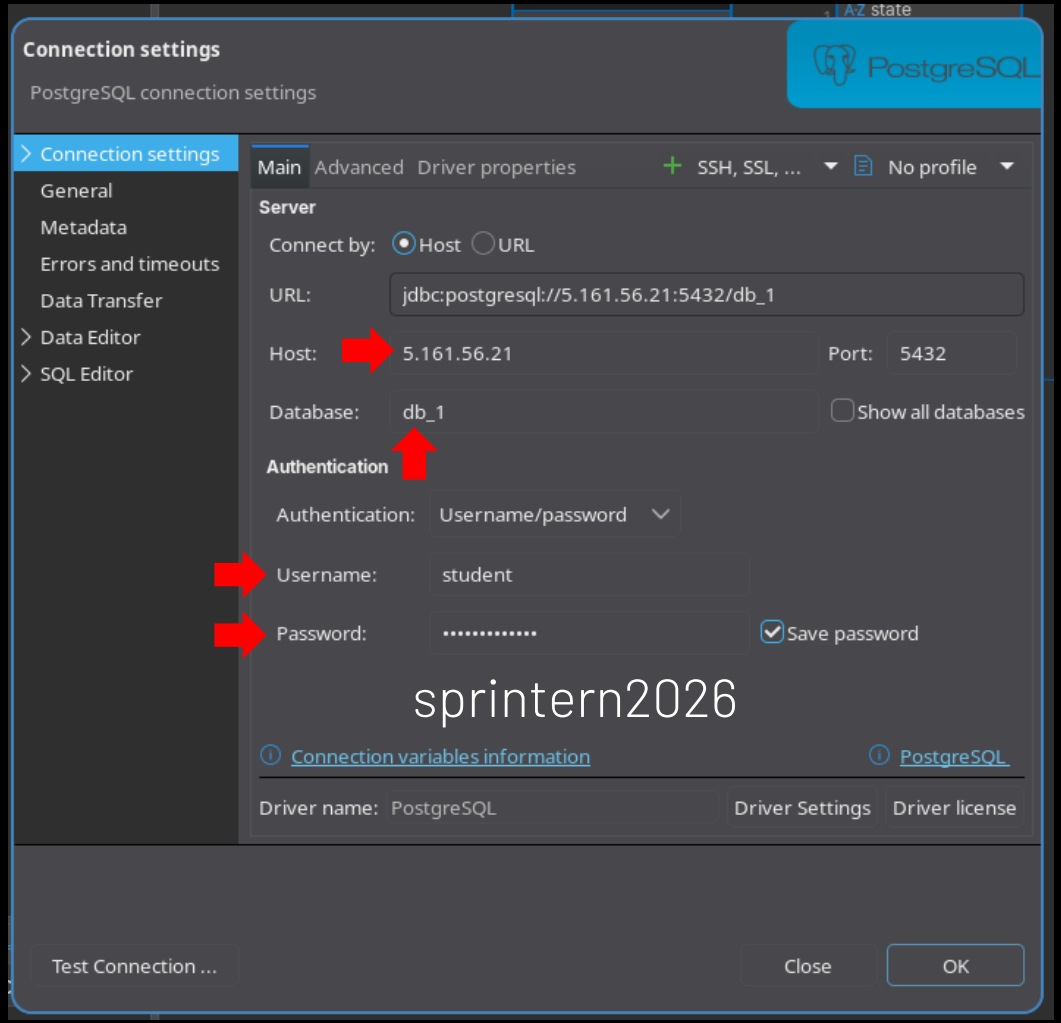
Database Clients

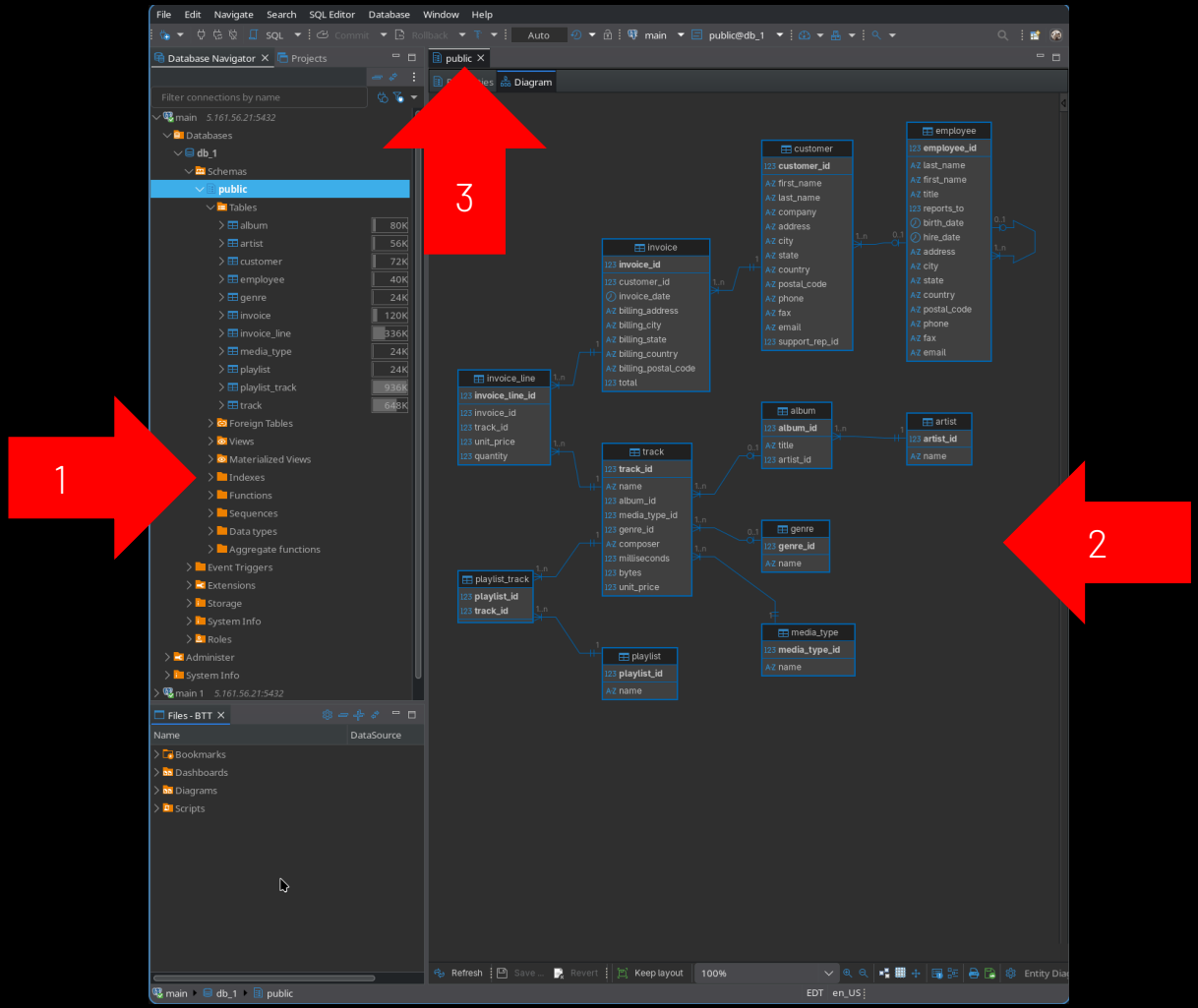


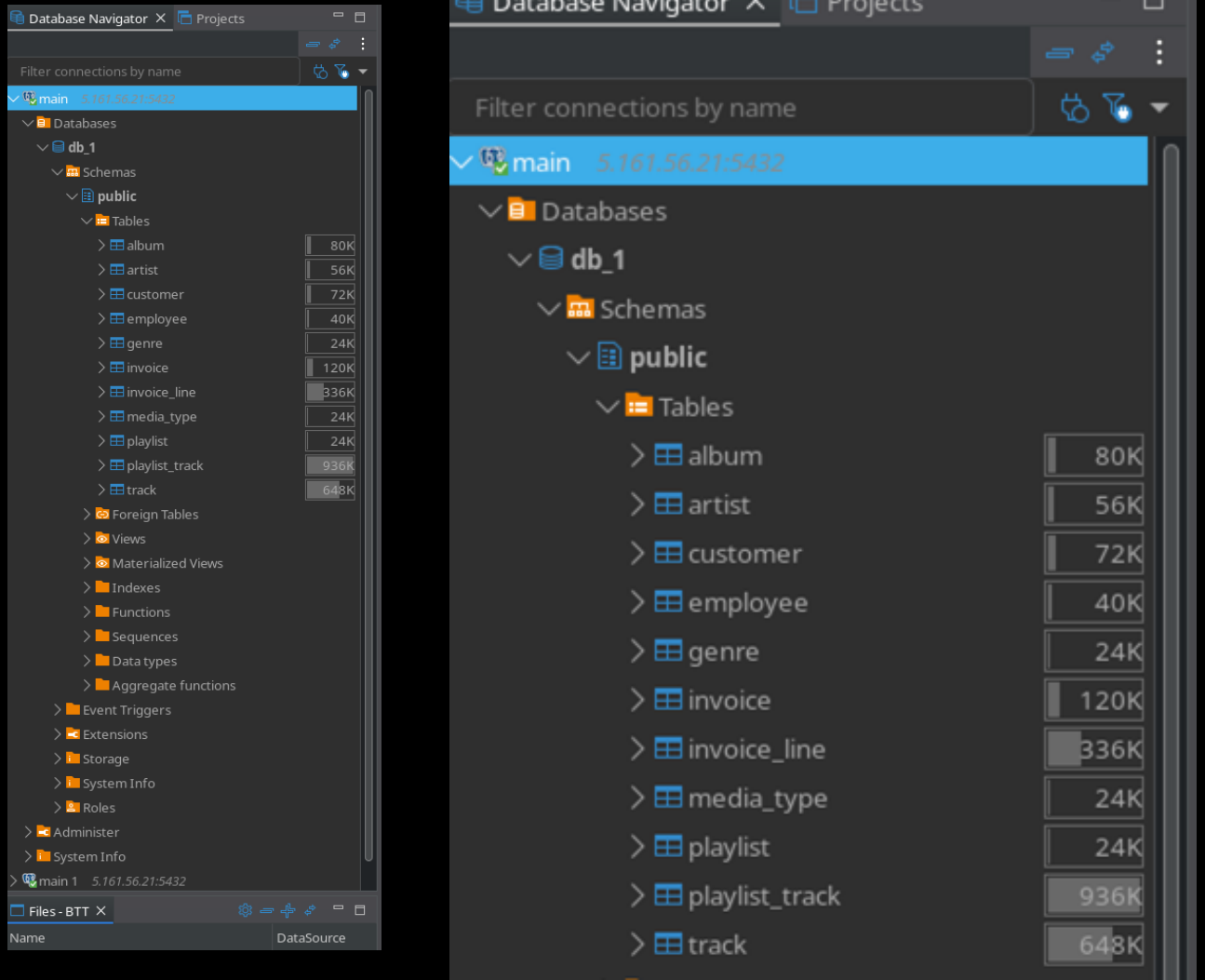


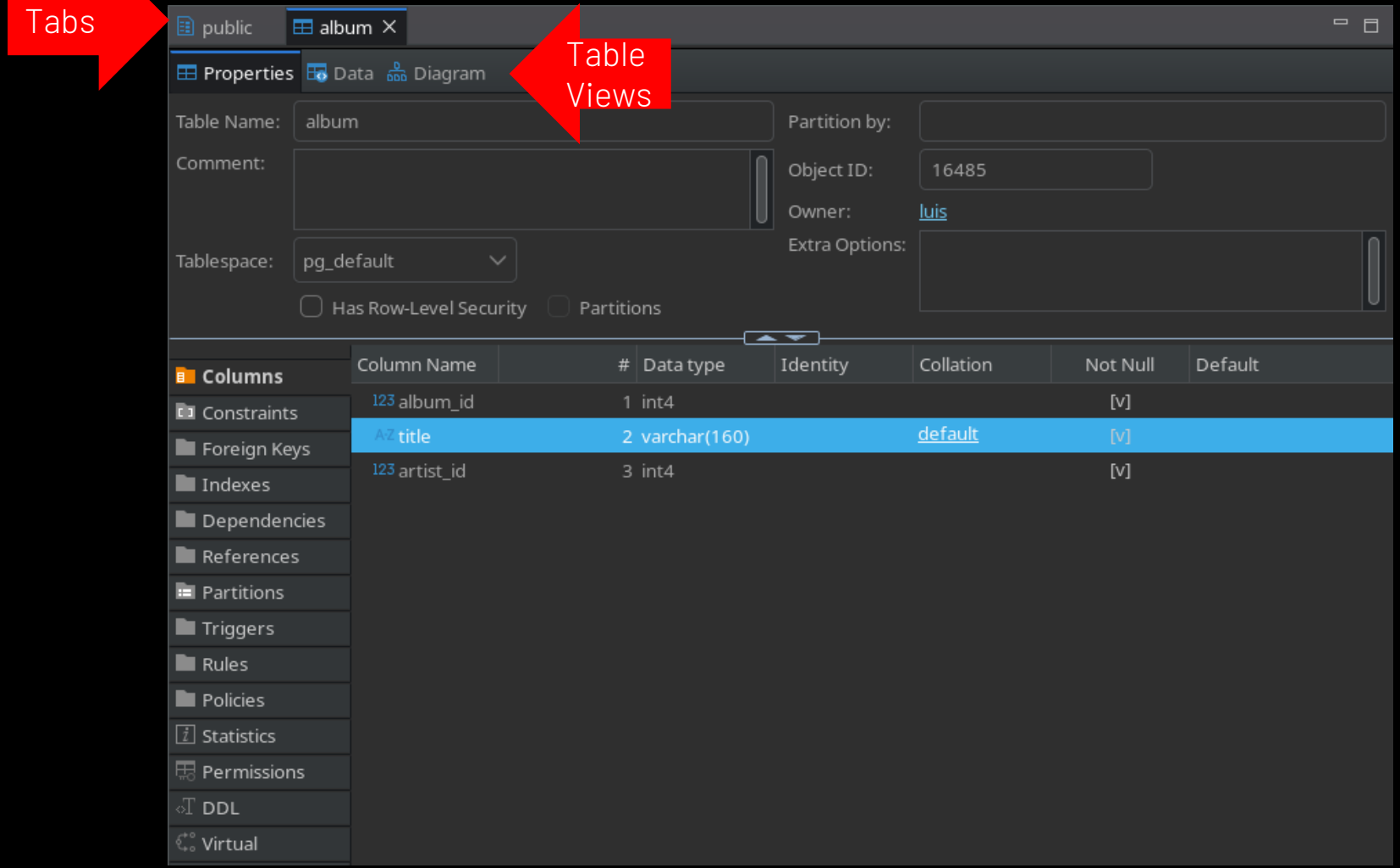

File Edit Navigate Search SQL Editor
SQL
New Database Connection (Shift+Ctrl+N)

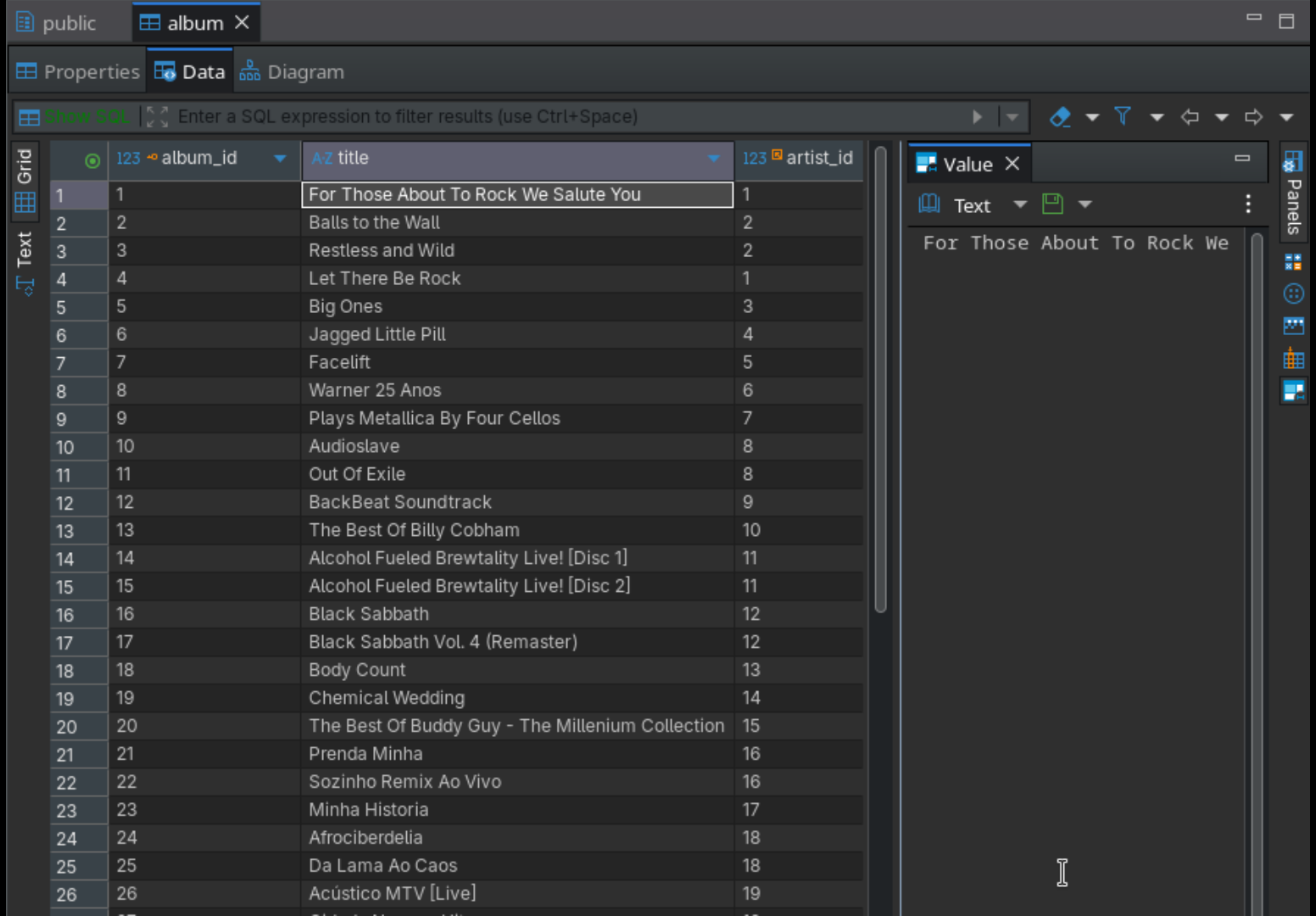

Select your database
Create new database connection. Find your database driver in the list below.
Type part of database/driver name t...
Sort by:
Title
Score
All
Popular
SQL
NoSQL
Analytical
Files
Embedded
Timeseries
Hadoop / BigData
Full-text search
Graph databases
PostgreSQL
Db2 for LUW
MariaDB
MySQL
Oracle
SQL Server
SQLite
Altibase
Apache Calcite Avatica
Apache Drill
Apache Hive 2
Apache Hive 4+
Project:
BTT
Test Connection ...
< Back
Next >
Cancel
Finish


Connection settings
PostgreSQL connection settings
PostgreSQL
Connection settings
General
Metadata
Errors and timeouts
Data Transfer
Data Editor
SQL Editor
Main
Advanced
Driver properties
SSH, SSL, ...
No profile
Server
Connect by:
Host
URL
URL:
jdbc:postgresql://5.161.56.21:5432/db_1
Host:
5.161.56.21
Port:
5432
Database:
db_1
Show all databases
Authentication
Authentication:
Username/password
Username:
student
Password:
Save password
sprintern2026
Connection variables information
PostgreSQL
Driver name:
PostgreSQL
Driver Settings
Driver license
Test Connection ...
Close
OK

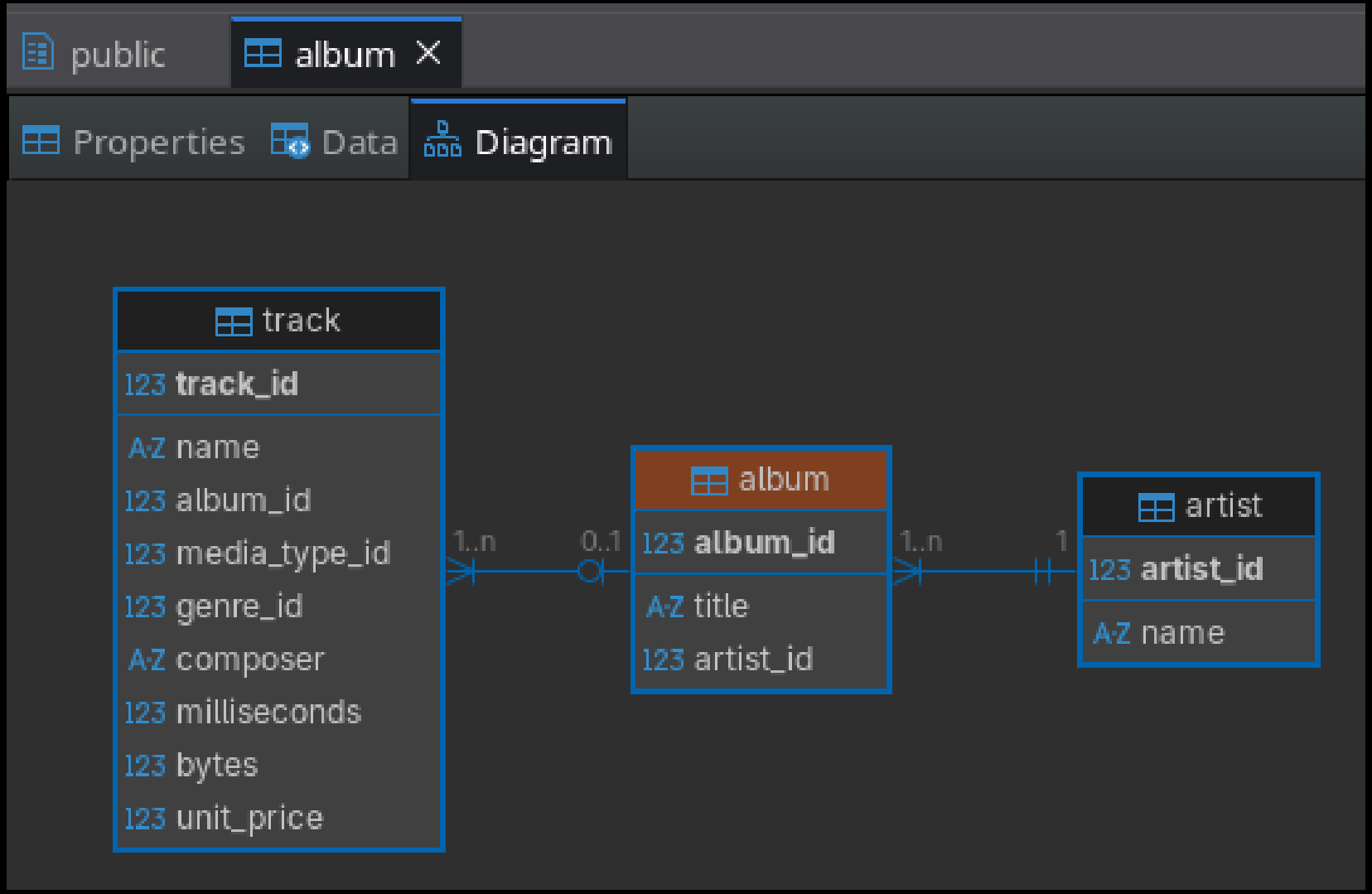

File Edit Navigate Search SQL Editor Database Window Help
Database Navigator
Projects
Filter connections by name
main 5.161.56.21:5432
Databases
db_1
Schemas
public
Tables
album 80K
artist 56K
customer 72K
employee 40K
genre 24K
invoice 120K
invoice_line 336K
media_type 24K
playlist 24K
playlist_track 936K
track 648K
Foreign Tables
Views
Materialized Views
Indexes
Functions
Sequences
Data types
Aggregate functions
Event Triggers
Extensions
Storage
System Info
Roles
Administer
System Info
main 1 5.161.56.21:5432
Files - BTT
Name
DataSource
Bookmarks
Dashboards
Diagrams
Scripts
public
Diagram
customer
customer_id
first_name
last_name
company
address
city
state
country
postal_code
phone
fax
email
support_rep_id
employee
employee_id
last_name
first_name
title
reports_to
birth_date
hire_date
address
city
state
country
postal_code
phone
fax
email
invoice
invoice_id
customer_id
invoice_date
billing_address
billing_city
billing_state
billing_country
billing_postal_code
total
invoice_line
invoice_line_id
invoice_id
track_id
unit_price
quantity
track
track_id
name
album_id
media_type_id
genre_id
composer
milliseconds
bytes
unit_price
album
album_id
title
artist_id
artist
artist_id
name
genre
genre_id
name
media_type
media_type_id
name
playlist_track
playlist_id
track_id
playlist
playlist_id
name
Refresh
Save ...
Revert
Keep layout
100%
Entity Dia
main
db_1
public
EDT
en_US
1
2
3

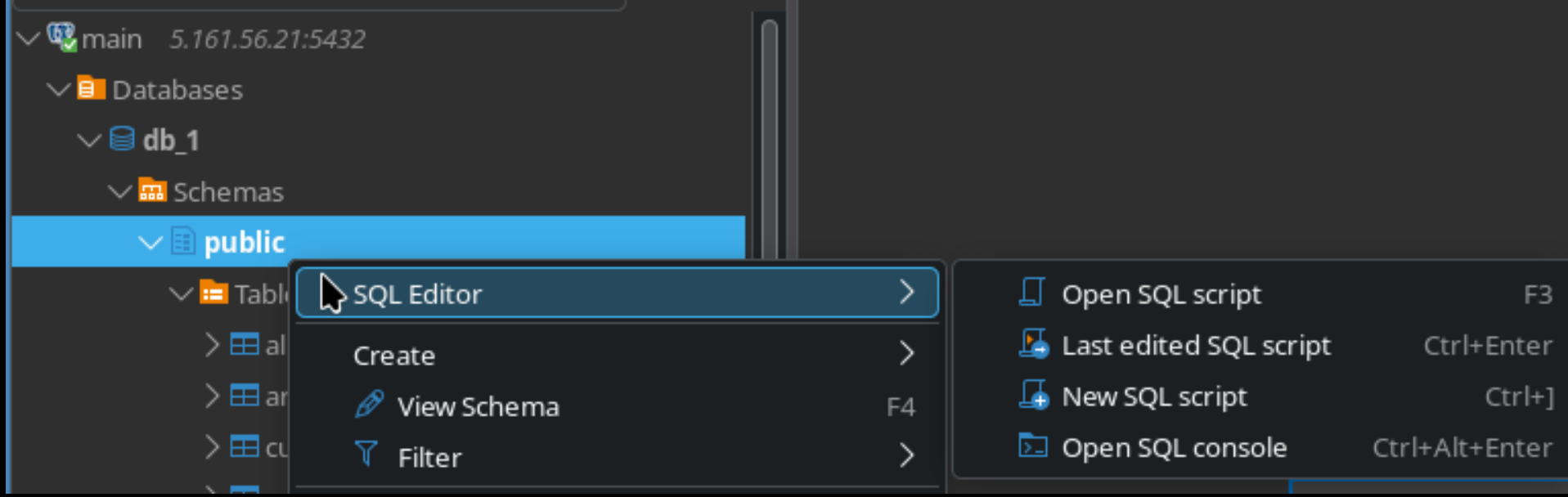

Database Navigator
Projects
Filter connections by name
main 5.161.56.21:5432
Databases
db_1
Schemas
public
Tables
album 80K
artist 56K
customer 72K
employee 40K
genre 24K
invoice 120K
invoice_line 336K
media_type 24K
playlist 24K
playlist_track 936K
track 648K
Foreign Tables
Views
Materialized Views
Indexes
Functions
Sequences
Data types
Aggregate functions
Event Triggers
Extensions
Storage
System Info
Roles
Administer
System Info
main 1 5.161.56.21:5432
Files - BTT
Name
DataSource

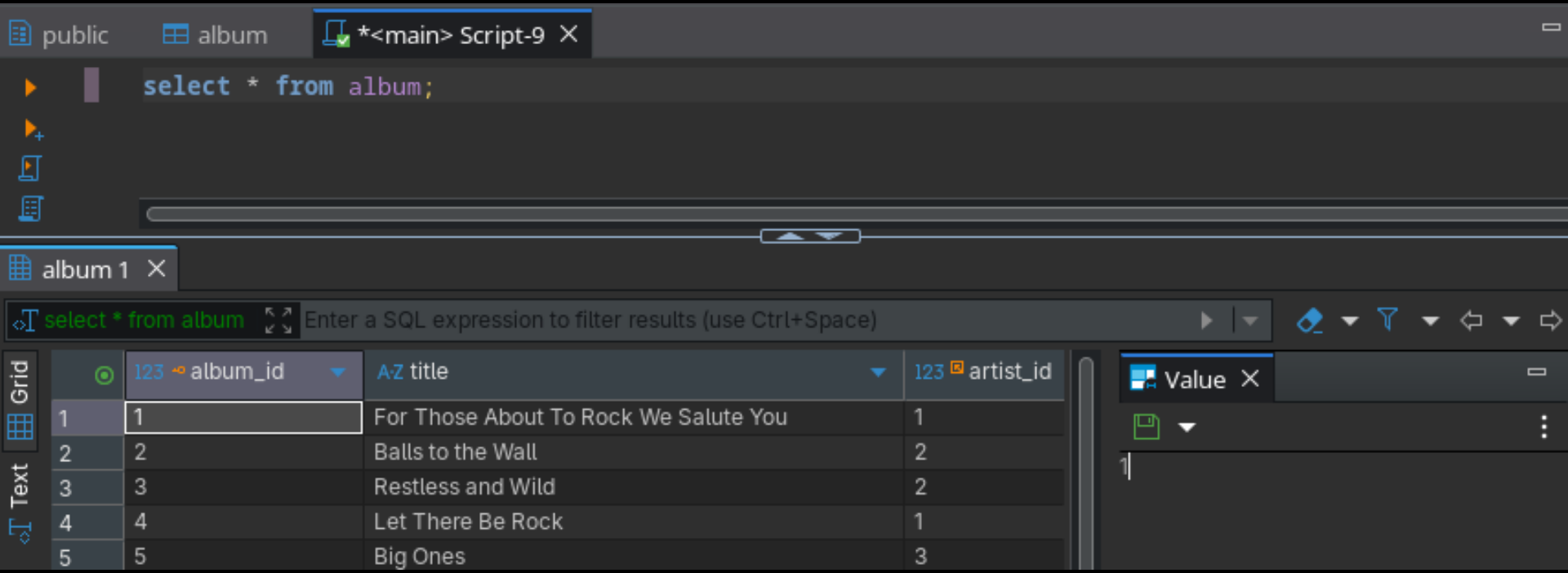

Tabs
public
album
Table Views
Properties
Data
Diagram
Table Name:
album
Partition by:
Comment:
Object ID:
16485
Owner:
luis
Tablespace:
pg_default
Extra Options:
Has Row-Level Security
Partitions
Columns
Constraints
Foreign Keys
Indexes
Dependencies
References
Partitions
Triggers
Rules
Policies
Statistics
Permissions
DDL
Virtual
Column Name
#
Data type
Identity
Collation
Not Null
Default
album_id
1
int4
[v]
title
2
varchar(160)
default
[v]
artist_id
3
int4
[v]



	album_id	title	artist_id
1	1	For Those About To Rock We Salute You	1
2	2	Balls to the Wall	2
3	3	Restless and Wild	2
4	4	Let There Be Rock	1
5	5	Big Ones	3
6	6	Jagged Little Pill	4
7	7	Facelift	5
8	8	Warner 25 Anos	6
9	9	Plays Metallica By Four Cellos	7
10	10	Audioslave	8
11	11	Out Of Exile	8
12	12	BackBeat Soundtrack	9
13	13	The Best Of Billy Cobham	10
14	14	Alcohol Fueled Brewtality Live! [Disc 1]	11
15	15	Alcohol Fueled Brewtality Live! [Disc 2]	11
16	16	Black Sabbath	12
17	17	Black Sabbath Vol. 4 (Remaster)	12
18	18	Body Count	13
19	19	Chemical Wedding	14
20	20	The Best Of Buddy Guy - The Millenium Collection	15
21	21	Prenda Minha	16
22	22	Sozinho Remix Ao Vivo	16
23	23	Minha Historia	17
24	24	Afrociberdelia	18
25	25	Da Lama Ao Caos	18
26	26	Acústico MTV [Live]	19


public
album
Properties
Data
Diagram
track
123 track_id
A-Z name
123 album_id
123 media_type_id
123 genre_id
A-Z composer
123 milliseconds
123 bytes
123 unit_price
1..n
0..1
album
123 album_id
A-Z title
123 artist_id
1..n
1
artist
123 artist_id
A-Z name


main 5.161.56.21:5432
Databases
db_1
Schemas
public
SQL Editor
Create
View Schema F4
Filter
Open SQL script F3
Last edited SQL script Ctrl+Enter
New SQL script Ctrl+]
Open SQL console Ctrl+Alt+Enter


public
album
*<main> Script-9
select * from album;
album 1
select * from album
Enter a SQL expression to filter results (use Ctrl+Space)
Grid
Text
album_id
title
artist_id
1 1 For Those About To Rock We Salute You 1
2 2 Balls to the Wall 2
3 3 Restless and Wild 2
4 4 Let There Be Rock 1
5 5 Big Ones 3
Value
1

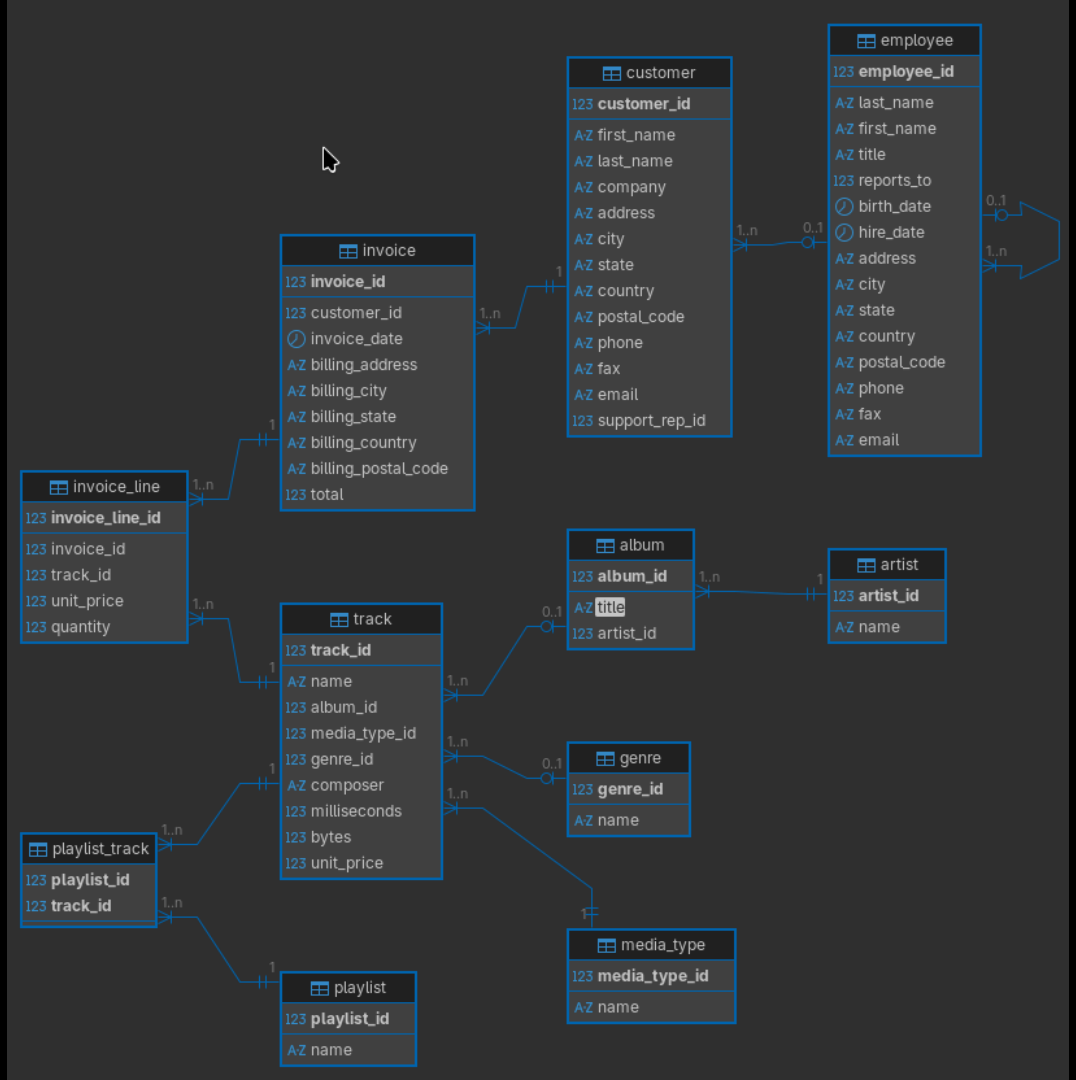
```sql
select * from album;
select * from artist;
select count(*) from invoice where billing_state  = 'FL'
```

esults 1

elect count(*) from invoice where billing_state = '

Enter a SQL expression to filter resu

	123 count
1	7


employee
employee_id
last_name
first_name
title
reports_to
birth_date
hire_date
address
city
state
country
postal_code
phone
fax
email
customer
customer_id
first_name
last_name
company
address
city
state
country
postal_code
phone
fax
email
support_rep_id
invoice
invoice_id
customer_id
invoice_date
billing_address
billing_city
billing_state
billing_country
billing_postal_code
total
invoice_line
invoice_line_id
invoice_id
track_id
unit_price
quantity
album
album_id
title
artist_id
artist
artist_id
name
track
track_id
name
album_id
media_type_id
genre_id
composer
milliseconds
bytes
unit_price
genre
genre_id
name
playlist_track
playlist_id
track_id
playlist
playlist_id
name
media_type
media_type_id
name
1..n
0..1
1

Chinook Activity
Connect to the database
In DBeaver and answer questions

Create a new Schema fiu, then create the FIU Soccer Game Tables in that schema