Check-In Code: s3qu3l

SQL and Relational Databases

ACM Hack

Slides: acmurl.com/sql-slides

Check-in code: s3qu3l

Join our Hack Discord @
acmurl.com/hack-disc

ACM Hack

What is a database?

ACM Hack

Databases

- A data structure that stores organized information
  - Ex: UCSD's database holds information about courses, students, faculty, dining halls, etc.
- Different types of databases store data differently
- In databases, data can be:
  - Created
  - Read
  - Updated
  - Deleted
- Data is generally stored with specific **data types**
  - Ex: A student's first name -> string
  - Ex: A student's graduation year -> integer

Types of Databases

There are many types of databases that store data differently and have their own use cases

ACM Hack

1. Key-value Database

Stores data entries in key-value pairs

- Keys are unique identifiers that map to a value for a given entry
  - Ex: Student PID as key, student information as value (firstName, lastName, major, etc.)
- To lookup data, provide the database a key and it will return the value
- Most key-value databases only support simple types such as strings

```
"A12345678": "John:Johnson:Computer Science:2024"
"A87654321": "Tom:Thompson:Computer Engineering:2021",
"A87654321": "Ron:Ronson:Data Science:2022",
```

ACM Hack

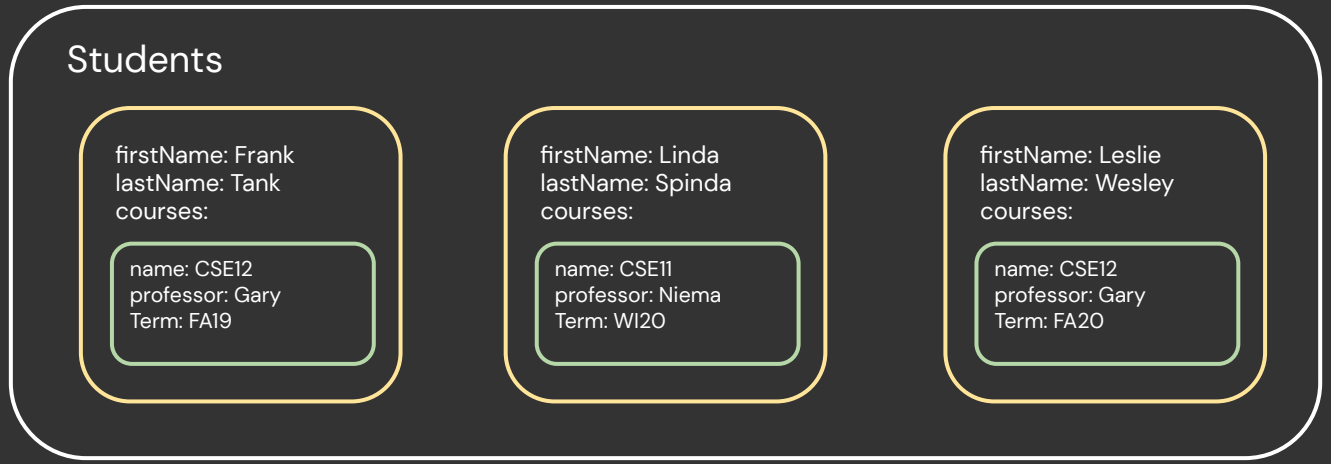
Key-Value Database Tradeoffs

- Advantages:
  - Very, very fast lookup and insertion
- Disadvantages:
  - Cannot filter data efficiently
  - Cannot store complex data models

2. Document Database

A more complex type of database storing data in collections

- Stores data as collections with documents as entries
  - Ex: Collection "Students" with documents as student info (firstName, lastName, courses)
  - Can have sub-documents within documents (e.g. students with courses)



Document Database Tradeoffs

- Advantages:
  - Very fast reads for data
    - Ex: I can easily get all information about a student and the classes they have taken
- Disadvantages:
  - Reading/writing data that has complex relationships is slow
    - Ex: Finding all the students who are taking CSE12

ACM Hack

3. Relational Database

Defines organized data and relationships in tables

- Data is organized in tables with columns as fields and rows as data entries
- Ex: "Professors" table storing name, department, and salary columns
- Can query data using SQL

Professors	
id	**int**
first_name	varchar
last_name	varchar
department	varchar
salary	int
year_started	int

Courses	
id	**int**
name	varchar
code	varchar
professor_id	int

Relational Database Tradeoffs

- Advantages:
  - Can query data with complex relationships
  - Can easily organize related data
  - Can use transactions to rollback complex writes/updates that cause errors
    - Wraps queries in a single logical unit that either commits or fails
- Disadvantages:
  - Joining relationships together is slow
  - Cannot store unstructured/dynamically modeled data
  - Knowledge of a separate query language is needed

Students	
id	**int**
firstName	varchar
lastName	varchar
major	varchar
graduationYear	int

DiningHalls	
id	**int**
name	varchar
location	varchar

Structured Query Language (SQL)

ACM Hack

Structured Query Language (SQL)

A query language used for managing data within a relational database system

In SQL, you can write queries that:

- Create database tables with defined schemas
- Read data from tables
- Read data that match certain constraints
- Write data to a specific table
- Delete data

Local Database Setup

ACM Hack

Setting Up a Database

To create our own tables and store data, we first need to setup a database.

We will be using PostgreSQL (a.k.a. "Postgres") as our database of choice, mainly because it:

- Comes built in with tons of useful SQL features and data types
- Is a very popular relational database management system (RDBMS)
- Is straightforward to install locally

Postgres Installation

Download and install the latest version for your OS here:

https://www.postgresql.org/download/

For installation:

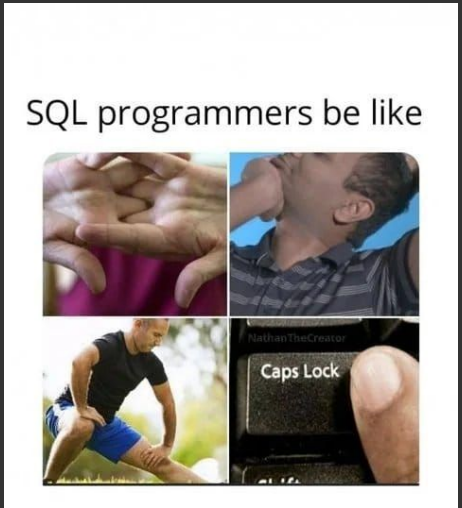
- Leave default installation values as is for directories, ports, data, etc.
- For superuser password, fill whatever (doesn't have to be strong)
- No need to launch Stack Builder on completion

Open the postgres shell by running the **SQL Shell (psql)** application

- Windows: find in Start Menu
- Mac: find in Applications -> Postgres 13 -> SQL Shell

ACM Hack

SQL Queries



hack ACM Hack

SQL Query Syntax

SQL QUERIES ARE TYPED IN UPPER CASE, because there is no syntax highlighting

Syntax rules specifically in Postgres:

- All property names are automatically case-folded to lowercase

  Ex: `CREATE TABLE Students;` will create a table called `students`

- Any uppercase properties need to be under double quotes
- Any uppercase values need to be under single quotes
- Semicolons are required;

```
SELECT "firstName", "lastName" FROM "Users" WHERE "firstName" = 'Bo';
```

ACM Hack

Creating Database Tables

ACM Hack

CREATE TABLE Query

Creates a table with the specified columns and their data types.

```
CREATE TABLE "TableName" (column1 datatype, column2 datatype, column3 datatype, ... );
```

- Each column is required to have a data type
- Basic data types include:
  - integer
  - varchar(size):
    - size of 255 is convention for "small-ish" text fields since the count can be stored in 8 bits
  - boolean

```
CREATE TABLE "Students" (first_name VARCHAR(255), last_name VARCHAR(255));
```

Example Table - Students

first_name (varchar)	last_name (varchar)
Brad	Davis
Peter	Parker
M	J

But what if we need to insert another student named "Peter Parker"? There is no way of differentiating the current one and any future ones

Primary Keys

- A way of uniquely identifying a row in a table
- Every table should have a primary key (id) column, as it:
  - Allows us to store rows that have the exact same fields
  - Makes storing relationships possible (out of the scope of this workshop)
- Type SERIAL, and auto-increments for each new row (1, 2, 3...)
  - Specify primary key property with PRIMARY KEY

```
CREATE TABLE "Students" (id SERIAL PRIMARY KEY, first_name VARCHAR(255), last_name VARCHAR(255));
```

Example Table – Students

id	first_name	last_name
1	Brad	Davis
2	Peter	Parker
3	M	J
4	Peter	Parker

Now we can store multiple rows with the same fields

ALTER TABLE Query

Alters the structure of an existing table.

Can be used to add, drop, and rename columns.

Sytax:

```
ALTER TABLE "TableName" ADD column_name data_type;

ALTER TABLE "TableName" DROP COLUMN column_name;

ALTER TABLE "TableName" RENAME COLUMN old_column_name TO new_column_name;
```

DROP TABLE Query

Drops an existing database table, deleting all data stored along with the table itself

Syntax:

```
DROP TABLE TableName;
```

Table Design - Workshop Tasks #1 & #2

We will be constructing a database for a new fake university, UCLJ, to be able to store data regarding students and courses

There will be multiple tasks throughout this workshop for incrementally building this database, table by table. We will start with creating 2 tables

Task #1 - Storing Student Data

The university wants to create a database table storing the following columns of student data:

- first_name
- last_name
- pid
- major
- graduation_year

NOTE: Don't forget about a primary key (as the first column)
NOTE: Please construct your table with fields in this exact order.
NOTE: \dt command shows all tables in your database

Task #2 - Storing Course Data

Next, the university wants to create a table to store the following columns for courses:

- name (e.g. Software Engineering)
- code (e.g. CSE 110)
- professor
- quarter_offered (e.g. Winter, Spring)
- year_offered (e.g. 2019, 2022)

Inserting Data

ACM Hack

INSERT INTO Query

Inserts new rows into a table.

```
INSERT INTO "TableName" (column1, column2, column3, ... ) VALUES (value1, value2, value3, ... );
```

- Need to specify the columns you will be inserting data into
- Unspecified columns will automatically store a null value (except id)

```
INSERT INTO "Students" (first_name, last_name, pid, major, graduation_year) VALUES ('Liz', 'Frizz', 'A123456789', 'Data Science', 2025);
```

"id" column will be default 1. Then 2, then 3, etc.

INSERT INTO Examples

Standard:

```
INSERT INTO "Students" (first_name, last_name, pid, major, graduation_year) VALUES
('Liz', 'Frizz', 'A123456789', 'Data Science', 2025);
```

No "pid":

```
INSERT INTO "Students" (first_name, last_name, major, graduation_year) VALUES
('Bob', 'Globb', 'EECS', 2023);
```

Explicit "id":

```
INSERT INTO "Students" VALUES (3, 'Chad', 'Lad', 'A666666666', 'Computer Science',
2024);
```

Note: if a row with id=3 already exists, an error is thrown

Populating our Tables

We will be populating our tables with thousands of rows of fake student data and course data

This may sound like a lot of data, but it's actually tiny (1000 Student rows is roughly 200kb).

Copy and paste the queries into your shell to insert the data

Reading and Analyzing Data

ACM Hack

Reading and Analyzing Data

Using our current SQL knowledge, we set up a storage mechanism for data to be written. But how can we read and analyze this data?

More importantly, how can we help answer questions like:

- How many courses are being taught in one quarter?
- What professors are teaching a specific course?
- How many students are majoring in Computer Science?
- When is Timmy graduating?

SELECT FROM Query

Reads data from one or more tables.

```
SELECT <column names> FROM "<TableName>" WHERE <filter>;
```

- There are tons of ways to filter data by conditions (next slide)
- Specific columns can be selected, or all of them (using "*")
- This is THE MOST important SQL query out there, as data is almost always read more than it is written

WHERE Clause

Provides an optional filter for selecting, updating, and deleting rows.

Common filters include:

- String matches
  - `WHERE property = value`
- Comparisons (>, <, =, !=)
  - `WHERE property < value`
- Element existence
  - `WHERE property IN (value1, value2, value3)`
- Logical operators (AND, OR, NOT)
  - `WHERE property1 = value1 AND property2 < value2 OR NOT property3 IN (values)`

WHERE Clause Usage

String matches:

- Single select: `SELECT * FROM "Students" WHERE first_name = 'Grace';`
- Selecting specific columns: `SELECT first_name, last_name FROM "Students" WHERE major = 'Computer Science';`
- Chaining logical operators: `SELECT pid FROM "Students" WHERE first_name = 'Dom' AND last_name = 'Cobb';`

  Note the usage of single quotes for string values

Comparisons

- Simple comparison: `SELECT * FROM "Merchandise" WHERE price > 500;`
- Chaining logical operators: `SELECT state FROM "CensusData" WHERE population < 40000000 AND votes >= 2000000;`

Demo: SELECT Query

ACM Hack

AS Clause

Selects specific columns under an alias

```
SELECT column1 AS new_column1, column2, column3 AS new_column3 FROM ...
```

Ex:

```
SELECT first_name AS name FROM "Students";
```

Seems contrived for now, but it becomes useful when selecting rows under **aggregate functions**

Aggregate Functions

Perform computations on multiple selected rows.

- Common aggregate functions include:
  - COUNT
  - MAX
  - MIN
  - SUM
  - AVG
- Can specify which columns to select if not all are needed

```
SELECT COUNT(*) FROM "Students" WHERE first_name = "Maokai";

SELECT MAX(price) FROM "Bicycles" WHERE color = 'Blue' OR color = 'Vantablack';
```

Task #3: Analytics

With this knowledge, we are now able to answer questions like the ones below:

1. How many courses are being taught in Winter 2021?
2. Which professors taught CSE 13 in 2020?
3. How many students are not majoring in Computer Science?

Can you come up with queries that can help answer each of these questions?

Even Richer Analytics

The previous queries only gave us insights into data from specific parameters that we desire.

i.e. professors who taught CSE 13, courses taught in Winter 2021, etc.

What if we want data and analytics to answer more generalized questions, like:

- How many courses are taught during each quarter?
- How many courses is each professor teaching?
- Give me a roster of all currently enrolled students sorted alphabetically.
- Find the average graduation year across all majors.

GROUP BY Clause

Groups rows with the same value of the specified property together.

Note: AGGREGATE can be any aggregate function (e.g. COUNT, MIN, SUM, AVG, etc.)

```
SELECT column1, AGGEGRATE(column2) FROM TableName GROUP BY column1;
```

GROUP BY rules:

- Every column specified must either be aggregated or grouped by
- For the above, column1 is "grouped by" and column2 is aggregated
- Can have multiple GROUP BY columns, not just one
- Examples on next slides

GROUP BY Example

Say we have these rows in a very simple "Orders" table:

id	item_name
1	Orange
2	Apple
3	Orange
4	Peach
5	Apple
6	Apple

->

item_name	count
Orange	2
Apple	3
Peach	1

```
SELECT item_name, COUNT(id) FROM "Orders" GROUP BY item_name;
```

`GROUP BY` defines which column(s) will collapse same values together
`COUNT` defines how the other column(s) will be aggregated
If either of these are missing, the query fails

ACM Hack

GROUP BY Live Example

Let's mess around in our psql shell for a bit

ACM Hack

ORDER BY Clause

Sorts rows based on the column(s) specified

```
SELECT column1, column2 FROM TableName ORDER BY column1;
```

- Order is default ascending (ASC)
- Can specify descending with DESC keyword
- ORDER BY is often used with GROUP BY to sort aggregates alphabetically or by count

```
SELECT first_name, COUNT(first_name) AS count FROM "Students" GROUP BY first_name
ORDER BY count DESC;
```

Task #4 - Richer Analytics

What queries can we use to answer the following questions?

- How many courses are taught during each quarter? Each unique quarter?
- How many courses is each professor teaching?
- Give me a roster of all currently enrolled students sorted alphabetically.
- Find the average graduation year across all majors.

Updating and Deleting Data

ACM Hack

UPDATE Query

Updates record columns in a table based on conditions

```
UPDATE TableName SET column1 = new_value1, column2 = new_value2 WHERE condition1,
condition2, ...;
```

Updates can be:

- New values
- Numerical increments

Examples on next slide

ACM Hack

UPDATE Query Examples

```
UPDATE "Students" SET graduation_year = 2024 WHERE id = 10;

UPDATE "Courses" SET code = 'CSE 14', name = 'Mediocre Data Structures' WHERE
code = 'CSE 13';

UPDATE "Merchandise" SET price = price + 50 WHERE id = 20;
```

DELETE Query

Deletes rows from a table based on filters

```
DELETE FROM TableName WHERE condition1, condition2, ...;
```

Filters use exact same syntax as with SELECT query

Ex:

```
DELETE FROM "Students" WHERE id = 10;
DELETE FROM "Students" WHERE graduation_year = 2022 AND major = 'Computer Science';
DELETE FROM "Merchandise" WHERE product_name = 'Algorithms, 3rd Edition';
```

ACM Hack

Some Notes on SQL

ACM Hack

You Don't Actually Write SQL Yourself

We wrote a lot of commands in the psql shell ourselves.

But this isn't how storing data is actually done in real applications. Programs need a way to store data from within code that is triggered by events.

This is solved by an object-relational mapping (ORM)

- ORMs provide functions to perform database operations
  - `ORM.find(table_name, filter)`
  - `ORM.insert(table_name, data)`
  - `ORM.createTable(table_name, columns)`
- These functions get turned into SQL that gets executed in your database

Further Topics

We barely, *barely* scratched the surface of SQL and relational databases.

Here are some more topics you can explore that we didn't cover:

- Relationships (one-to-many, many-to-one, many-to-many)
- Foreign keys
- JOIN Clause
- Database Indexing
- Advanced data structures (arrays, blobs, json, bson, etc.)

Thanks for coming!

If you want to learn more cool topics this quarter,
fill out https://acmurl.com/hack-requests to help us decide what to teach!

Interested in helping out during workshops? Join the HackSquad!
https://acmurl.com/hacksquad

hack ACM Hack