CS6 Practical System Skills

Fall 2019 edition

Leonhard Spiegelberg
lspiegel@cs.brown.edu

19 Databases

CS6 Practical System Skills

Fall 2019

Leonhard Spiegelberg *lspiegel@cs.brown.edu*

19.01 What is a database?

actually a very broad term, in the end

⇒ a (very large) collection of (inter)related data

DBMS = Database Management System

⇒ a software system that can be used to store, manage, retrieve and transform data from a database

→ often the term database / DBMS are used interchangeably

⇒ Brown offers several database courses: CS127, CS227, CS295

19.02 Why should we use a database?

⇒ storing, updating and retrieving data

→ becomes difficult when multiple users, clients, connections, nodes, … are involved.

→ fault-tolerance, i.e. a good database is resilient towards node failures, connection drops, …

→ retrieving specific records efficiently, indexing data.

⇒ data security, rights management for accessing data.

⇒ data integrity, ensuring data follows one format.

⇒ data analytics, information aggregation.

19.02 When to use a database?

⇒ at their core, databases provide two core functionalities:

→ **persistence:** keeping data, tolerant to all sorts of events. I.e. non-volatile storage of state.

→ **data access:** adding and extracting data easily.

19.02 When to use a database?

When to use a database…?

… use one if your project needs to access/update many (small) records (for multiple users) with different access patterns!

19.03 When to NOT use a database

KISS = Keep it simple stupid

⇒ a database usually provides a lot of functionality and might have a large memory footprint.

⇒ Does your project really need a database?

Examples where a database might be a bad choice:

- static website like a blog / API documentation
- log files
- statistical analysis over small datasets (< 1 GB) or large datasets (> 50GB)
- storage of few and/or large and/or rarely changing records like videos/images/mp3s/...

19.04 Classification of databases OLAP/OLTP

⇒ databases can be classified according to their primary purpose

OLTP = Online transaction processing

→ primary objective is data processing, not analysis.
→ high volume of transactions, usually many small inserts/updates

OLAP = Online analytical processing

→ primary objective is data analysis, i.e. data warehousing
→ low volume of transactions, complex queries with aggregations

19.04 Classification of databases - model

⇒ Another way of classifying databases is by their data model

1. **Relational databases** (SQL-databases)
   ⇒ data is stored in tables
   ⇒ based on the relational data model
2. **Non-relational databases** (NoSQL-databases, NoSQL = not only sql)
   ⇒ do not follow traditional, relational data model
   ⇒ models include Key/Value-Stores, Document-Stores, Graph-databases, ...

there are many more ways to classify databases ⇒ CS127

Relational databases

19.06 Relational databases

⇒ Store data in tables,
query data using SQL = Structured Query Language

⇒ popular relational databases include

open source	**commercial**
MySQL* / MariaDB **PostgreSQL ⇐ we'll be using this** SQLlite ...	MySQL SAP Database Oracle Database Microsoft SQL Server IBM DB2 Database VoltDB ...

19.06 Tables in relational databases

⇒Data is stored in tables which consist of columns(attributes) and rows(tuples).

→ order of columns(attributes) does not matter.

name	year	country
Wu Tang Clan	1992	USA
Notorious BIG	1992	USA
Ice Cube	1989	USA
Beatles	1960	United Kingdom

Row(tuple)

Column(attribute)

19.06 Tables in relational model

⇒ we can create multiple tables and relate them using keys (id fields).

⇒ A primary key is a column which must not be NULL and contains a unique value for each row (i.e. a unique ID)

⇒ A foreign key is a column which references a primary key of another table

Pokemon

id	name	type
100	Bulbasaur	Grass, Poison
120	Charmander	Fire
130	Squirtle	Water

captures

pokemonID	trainerID
120	99
100	100

Trainer

id	name
99	Leonhard
100	Shriram

Note: Keys can also consist of multiple columns/attributes

19.07 How to work with a database

⇒ There are typically two ways how to interact with a database:

1. **Database shell**

   Nearly all databases come with a shell providing a REPL to issue SQL commands to query data or perform other administrative tasks.

2. **Database connectors/adapters**

   Libraries available for multiple programming languages which allow to issue queries and retrieve/store data.

19.08 PostgreSQL

⇒ free and open-source relational DBMS, evolved from Ingres project at UC Berkeley (Michael Stonebraker, A.M. Turing Award 2015)

⇒ "*The World's Most Advanced Open Source Relational Database*"

⇒ Resources:

postgresql.org

postgresqltutorial.com

19.08 PostgreSQL shell

⇒ start via `psql <db-name>` or run command directly
via `psql -c '<cmd>' <db-name>`

⇒ You can also execute SQL commands stored in a file via `psql -f <file>`

⇒ github.com/browncs6/DBIntro holds examples of this lecture

⇒ you can use the shell to directly issue SQL statements
(e.g. creating, modifying tables and retrieving/storing data)

⇒ A database administrator typically works in the database shell.

→ shell is useful to develop/test database queries!

19.08 A first SQL query - Creating a table

```
CREATE TABLE table_name (
   column_name TYPE column_constraint,
   table_constraint
);
```

⇒ a new table can be created using `CREATE TABLE` command

⇒ column_constraint can be `NOT NULL, UNIQUE, PRIMARY KEY, CHECK, REFERENCES`

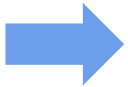
⇒ instead of defining constraints on columns, they can be also defined as table constraints on multiple columns at once, e.g.

`PRIMARY KEY (role_id, role_name)`

⇒ `TYPE` must be one of the supported datatypes of the database. For a complete list see http://www.postgresqltutorial.com/postgresql-data-types/

19.08 Creating a table - example

```
CREATE TABLE pokemon(name
VARCHAR(128) NOT NULL, height_ft
DECIMAL NOT NULL, weight_lbs DECIMAL
NOT NULL, category VARCHAR(128) NOT
NULL, PRIMARY KEY (name));
```



name	category	height_ft	weight_lbs

Tip:

When you create your application use a file `setup.sql` (or so) where you add all the `CREATE TABLE` statements required to setup your database. Helps to setup things when deploying!

19.08 More SQL

SQL offers many powerful commands which can be used to write a query.

As a start, you should get comfortable using

```
CREATE TABLE
INSERT INTO
SELECT
UPDATE
DELETE
```

⇒ demo time, Jupyter notebook!

19.09 Transactions

⇒ Sometimes we want to run multiple statements at once, but they should be atomic with respect to other users.

⇒ PostgreSQL supports ACID transactions

ACID = **A**tomic, **C**onsistent, **I**solated, **D**urable

Atomicity:	Multiple statements behave as single unit, i.e. either all succeed or none.
Consistency:	Transaction does not violate any database rules.
Isolation:	No transaction will be affected by any other transaction.
Durability:	Once transaction is committed, it's persisted.

⇒ typical usage: start a new transaction, execute SQL statements, COMMIT

19.09 Transactions in PostgreSQL

```
BEGIN;
```

you can also use `BEGIN TRANSACTION` or `BEGIN WORK`

```
...
```

put all SQL commands you want to treat as single UNIT (i.e. one ACID transaction) here.

```
COMMIT;
```

you can also use `COMMIT TRANSACTION` or `COMMIT WORK`

Tip: Via `ROLLBACK` you can undo your last transaction.

Connecting to a database using Python

Python Database adapter

⇒ so far we worked in the database shell, good to develop queries and manipulate data within the database

⇒ How can we use a database in an application/script?

⇒ There are libraries which allow us to interact with a PostgreSQL database!

⇒ We'll be using psycopg (Version 2)

→ `pip3 install psycopg2`

→ http://initd.org/psycopg/docs/

Psycopg2

⇒ demo time

→ Jupyter notebook

Flask & databases

Object - relational mapping

⇒ You can use a python-database adapter like `psycopg2` directly in your Flask application if you want.

⇒ to use a database, you need to write queries and define how to map python data to relational data → Object-relational mapping

⇒ **Problem:** You might want to exchange the database and avoid wasting too much time on defining the mapping/common queries.

⇒ Solution: There exist high-level libraries like SQLalchemy which allow to map python data structures to a relational database (schema)

SQLalchemy

⇒ following slides are based on Chapter 5, Flask book.

⇒ we'll be using flask-sqlalchemy, which integrates sqlalchemy with a flask application

→ `pip3 install flask-sqlalchemy`

⇒ demo!

End of lecture.

Next class: Tue, 4pm-5:20pm @ CIT 477