Databases

SQL, NoSQL, ORMs, REST/GraphQL, JSON/gRPC
SQL (1974)

- IBM Research
- Relational databases with query languages based on mathematical logic
- SQL (Chamberlin, Boyce)
  - Query language accessible to those without formal training in mathematics or computer programming
- Hugely successful
  - DB2, Oracle, SQL Server, PostgreSQL
- Databases support ACID properties
  - Atomic (transactions fully completed or not begun at all)
  - Consistent (system must remain in valid state, errors rolled back)
  - Isolated (each transaction executes as if it is the only one)
  - Durable (all changes permanent)
- Then, the Internet and WWW happened....
High volume, high velocity data
Non-uniform, "dirty" data
Can not be handled by traditional relational database
ACID properties difficult to scale-up to large amounts of data and not required
Motivates move to non-relational databases
  Bigtable, Dynamo, MongoDB, Cassandra, Redis, etc.
  Eschew the use of SQL to store, query, and retrieve data “NoSQL”
NoSQL

- No standard query language
- Non-standard interfaces between applications and database
- Limited querying support (i.e. no JOINs)
- Often schemaless (e.g. key-value storage of arbitrary types)
  - Like a Python dictionary
- Often with weak consistency (BASE)
  - Basically Available (requests may return failure or a transient inconsistent state)
  - Soft state (system always moving towards consistent state)
  - Eventual consistency (when input stops, system will reach a consistent state)
- But, get performance and scale!
- Huge variety of implementations tailored to specific application
  - \[http://nosql-database.org/\]
  - How to choose? (Take 486/586 + Cloud and Cluster course?)
# SQL vs. NoSQL

<table>
<thead>
<tr>
<th>SQL</th>
<th>NoSQL (initially)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational (structured) data</td>
<td>Non-relational (unstructured) data</td>
</tr>
<tr>
<td>Complex querying (JOINs)</td>
<td>Simple key-value lookup</td>
</tr>
<tr>
<td>Schema</td>
<td>No schema</td>
</tr>
<tr>
<td>ACID (Atomic, Consistent, Isolated, Durable) consistency</td>
<td>BASE (Basic Availability Soft-state Eventual) consistency</td>
</tr>
</tbody>
</table>
NewSQL: Revenge of SQL?

- Bolt on SQL features to NoSQL
  - Transaction processing (OLTP)
  - SQL interfaces on top of Hadoop and Spark
- Bolt on NoSQL features to SQL
  - Horizontally scalable SQL systems (H-Store, Spanner)
- Get the best of both worlds
  - SQL querying and consistency
  - NoSQL scaling
**SQL as the unifying language?**

- Timescale blog
SQL

http://www.w3schools.com/sql
SQL: Structured Query Language

- Standard language for accessing and manipulating databases
  - MySQL, Postgres SQL, SQLite, SQL Server, Access, Oracle, Sybase, DB2, etc.
  - Major commands are same between them
- Includes two things
  - Data Definition Language (DDL)
  - Data Manipulation Language (DML)
SQL Data Definition Language
SQL DDL (Data Definition)

- Permits databases and database tables to be created or deleted (among other operations).
  - CREATE creates a new database, table, index
  - DROP : deletes a database, table, index
CREATE DATABASE
• Used to create a database

```
CREATE DATABASE database_name;
CREATE DATABASE my_db;
```

CREATE TABLE
• Used to create a table within database

```
CREATE TABLE table_name
(column_name1 data_type,
column_name2 data_type,
column_name3 data_type,
....);
```

```
CREATE TABLE persons ( 
    P_Id INT,
    LastName VARCHAR(25),
    FirstName VARCHAR(25),
    Address VARCHAR(25),
    City VARCHAR(15),
    PRIMARY KEY (P_Id)
);```
## Basic SQL Datatypes

- Data must be clean! (Not easy to guarantee)

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>Allows whole numbers from 0 to 255</td>
<td>1 byte</td>
</tr>
<tr>
<td>Integer</td>
<td>Allows whole numbers between -32,768 and 32,767</td>
<td>2 bytes</td>
</tr>
<tr>
<td>Long</td>
<td>Allows whole numbers between -2,147,483,648 and 2,147,483,647</td>
<td>4 bytes</td>
</tr>
<tr>
<td>Single</td>
<td>Single precision floating-point. Will handle most decimals</td>
<td>4 bytes</td>
</tr>
<tr>
<td>Double</td>
<td>Double precision floating-point. Will handle most decimals</td>
<td>8 bytes</td>
</tr>
<tr>
<td>Currency</td>
<td>Use for currency. Holds up to 15 digits of whole dollars, plus 4 decimal places. <strong>Tip:</strong> You can choose which country's currency to use</td>
<td>8 bytes</td>
</tr>
<tr>
<td>Date/Time</td>
<td>Use for dates and times</td>
<td>8 bytes</td>
</tr>
<tr>
<td>Yes/No</td>
<td>A logical field can be displayed as Yes/No, True/False, or On/Off. In code, use the constants True and False (equivalent to -1 and 0). <strong>Note:</strong> Null values are not allowed in Yes/No fields</td>
<td>1 bit</td>
</tr>
<tr>
<td>OLE Object</td>
<td>Can store pictures, audio, video, or other BLOBs (Binary Large OBjects)</td>
<td>up to 1GB</td>
</tr>
<tr>
<td>Hyperlink</td>
<td>Contain links to other files, including web pages</td>
<td></td>
</tr>
<tr>
<td>Lookup Wizard</td>
<td>Let you type a list of options, which can then be chosen from a drop-down list</td>
<td>4 bytes</td>
</tr>
</tbody>
</table>
CS 201 Tutoring Python/Flask app

- [https://bitbucket.org/wuchangfeng/cs410c-src](https://bitbucket.org/wuchangfeng/cs410c-src)
  - WebDev_itscs201
  - Create a per-user table of questions and answers

```python
import sqlite3

userList=['llawrens','wuchang']

for user in userList:
    conn = sqlite3.connect('..//db/{0}.db'.format(user))
    conn.execute('''CREATE TABLE QUESTIONBANK
        (NUMBER VARCHAR(10) NOT NULL,
        CHAPTER INT NOT NULL,
        QUESTION TEXT NOT NULL,
        ANSWER TEXT,
        STATUS VARCHAR(50),
        SCOREID INT NOT NULL,
        HINT TEXT,
        PRIMARY KEY(NUMBER,CHAPTER));''')
    conn.close()
```
SQL Data Manipulation Language
Basic functions used in web apps

- CRUD
  - Create = INSERT INTO
  - Read = SELECT
  - Update = UPDATE
  - Delete = DELETE
SELECT

- Used to read data from a database
- Result stored in a table, called the result-set

Persons Table

<table>
<thead>
<tr>
<th>P_Id</th>
<th>LastName</th>
<th>FirstName</th>
<th>Address</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hansen</td>
<td>Ola</td>
<td>Timoteivn 10</td>
<td>Sandnes</td>
</tr>
<tr>
<td>2</td>
<td>Svendson</td>
<td>Tove</td>
<td>Borgvn 23</td>
<td>Sandnes</td>
</tr>
<tr>
<td>3</td>
<td>Pettersen</td>
<td>Kari</td>
<td>Storgt 20</td>
<td>Stavanger</td>
</tr>
</tbody>
</table>

SELECT column_name(s) FROM table_name;

SELECT FirstName, LastName FROM Persons;

<table>
<thead>
<tr>
<th>Firstname</th>
<th>Lastname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ola</td>
<td>Hansen</td>
</tr>
<tr>
<td>Tove</td>
<td>Svendson</td>
</tr>
<tr>
<td>Kari</td>
<td>Pettersen</td>
</tr>
</tbody>
</table>

SELECT * FROM Persons;

<table>
<thead>
<tr>
<th>P_Id</th>
<th>LastName</th>
<th>FirstName</th>
<th>Address</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hansen</td>
<td>Ola</td>
<td>Timoteivn10</td>
<td>Sandnes</td>
</tr>
<tr>
<td>2</td>
<td>Svendson</td>
<td>Tove</td>
<td>Borgvn23</td>
<td>Sandnes</td>
</tr>
<tr>
<td>3</td>
<td>Pettersen</td>
<td>Kari</td>
<td>Storgt20</td>
<td>Stavanger</td>
</tr>
</tbody>
</table>
WHERE clause

- Add a predicate to limit what is returned

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator value;
```

Persons Table

<table>
<thead>
<tr>
<th>P_Id</th>
<th>LastName</th>
<th>FirstName</th>
<th>Address</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hansen</td>
<td>Ola</td>
<td>Timoteivn 10</td>
<td>Sandnes</td>
</tr>
<tr>
<td>2</td>
<td>Svendson</td>
<td>Tove</td>
<td>Borgvn 23</td>
<td>Sandnes</td>
</tr>
<tr>
<td>3</td>
<td>Pettersen</td>
<td>Kari</td>
<td>Storgt 20</td>
<td>Stavanger</td>
</tr>
</tbody>
</table>

```
SELECT * FROM persons
WHERE city=‘Sandnes’;
```

- Text values should be quoted by single quotes or double quotes
- Numeric values do not need to be enclosed in quotes
INSERT INTO

- Used to create new records in a table
- Example

```sql
CREATE DATABASE test;
CREATE TABLE persons (
    P_Id INT,
    LastName VARCHAR(25),
    FirstName VARCHAR(25),
    Address VARCHAR(25),
    City VARCHAR(15),
    PRIMARY KEY (P_Id)
);

INSERT INTO persons VALUES (1, 'Hansen', 'Ola', 'Timoteivn10', 'Sandnes');
INSERT INTO persons VALUES (2, 'Svendson', 'Tove', 'Borgvn23', 'Sandnes');
INSERT INTO persons VALUES (3, 'Pettersen', 'Kari', 'Storgt20', 'Stavanger');
```

Persons Table

<table>
<thead>
<tr>
<th>P_Id</th>
<th>LastName</th>
<th>FirstName</th>
<th>Address</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hansen</td>
<td>Ola</td>
<td>Timoteivn 10</td>
<td>Sandnes</td>
</tr>
<tr>
<td>2</td>
<td>Svendson</td>
<td>Tove</td>
<td>Borgvn 23</td>
<td>Sandnes</td>
</tr>
<tr>
<td>3</td>
<td>Pettersen</td>
<td>Kari</td>
<td>Storgt 20</td>
<td>Stavanger</td>
</tr>
</tbody>
</table>
INSERT INTO table_name
VALUES (value1, value2, value3,...);


INSERT INTO table_name (column1, column2, column3, ...
VALUES (value1, value2, value3,...);

INSERT INTO persons (P_Id, lastname, firstname)
VALUES (5, ‘Tjessem’, ‘Jakob’);

<table>
<thead>
<tr>
<th>P_Id</th>
<th>LastName</th>
<th>FirstName</th>
<th>Address</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hansen</td>
<td>Ola</td>
<td>Timotei vn10</td>
<td>Sandnes</td>
</tr>
<tr>
<td>2</td>
<td>Svendson</td>
<td>Tove</td>
<td>Borgvn23</td>
<td>Sandnes</td>
</tr>
<tr>
<td>3</td>
<td>Pettersen</td>
<td>Kari</td>
<td>Storgt20</td>
<td>Stavanger</td>
</tr>
<tr>
<td>4</td>
<td>Nilsen</td>
<td>Tom</td>
<td>Vingvn23</td>
<td>Stavanger</td>
</tr>
<tr>
<td>5</td>
<td>Tjessem</td>
<td>Jakob</td>
<td>(NULL)</td>
<td>(NULL)</td>
</tr>
</tbody>
</table>
UPDATE

• Used to update records in a table

```sql
UPDATE table_name
SET column=value, column2=value2,...
WHERE some_column=some_value;
```

```sql
UPDATE Persons
SET Address='Nissestien 67', city='Sandnes'
WHERE lastname='Tjessem' AND firstname='Jakob';
```
DELETE statement

- Used to delete records in a table

```
DELETE FROM table_name
WHERE some_column=some_value;
```

```
DELETE FROM persons
WHERE lastname='Tjessem' AND firstname='Jakob';
```
Other common SQL primitives

- Pattern matching
  - % wildcard for matching 0 or more characters
  - _ wildcard for matching exactly 1 character

- Result processing
  - LIMIT
  - ORDER BY / DESC / ASC
  - DISTINCT

- Table combining
  - JOIN (INNER, LEFT, RIGHT, FULL)

- Constraints
  - NOT NULL
  - UNIQUE
  - PRIMARY KEY
  - FOREIGN KEY
  - CHECK (constraint)
CS 201 Tutoring Python/Flask app

- [https://bitbucket.org/wuchangfeng/cs410c-src](https://bitbucket.org/wuchangfeng/cs410c-src)
- WebDev_it.scss201
- Insert question, answers, and hint into question bank table

```python
# Store the Questions and answers in database
conn = sqlite3.connect('.../db/{0}.db'.format(username))

for qkey, question in self.questions.items():
    conn.execute("INSERT INTO QUESTIONBANK (NUMBER,CHAPTER,QUESTION,STATUS,SCOREID) VALUES('{0}',2,'{1}','UNSOLVED',{2})".format(qkey, question, id))
    id += 1

for akey, answer in self.answers.items():
    conn.execute("UPDATE QUESTIONBANK SET ANSWER='{0}' WHERE CHAPTER=2 AND NUMBER='{1}"").format(answer, akey)

for akey, hint in self.hints.items():
    conn.execute("UPDATE QUESTIONBANK SET HINT='{0}' WHERE CHAPTER=2 AND NUMBER='{1}"").format(hint, akey)

conn.commit()
conn.close()
```
SQL Functions
SQL aggregate functions

- SQL aggregate functions: return a single value calculated from values in a column
  - AVG(), COUNT(), MAX(), MIN(), SUM()

<table>
<thead>
<tr>
<th>P_Id</th>
<th>ProductName</th>
<th>UnitPrice</th>
<th>UnitsInStock</th>
<th>UnitsOnOrder</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jarlsberg</td>
<td>10.45</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>Mascarpone</td>
<td>32.56</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Gorgonzola</td>
<td>15.67</td>
<td>9</td>
<td>20</td>
</tr>
</tbody>
</table>

SELECT AVG(column_name) FROM table_name

SELECT AVG(UnitPrice) AS UnitAverage FROM Products

```
<table>
<thead>
<tr>
<th>UnitAverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.6667</td>
</tr>
</tbody>
</table>
```

SELECT ProductName FROM products
WHERE UnitPrice>=( SELECT AVG(UnitPrice) FROM Products)

```
<table>
<thead>
<tr>
<th>ProductName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mascarpone</td>
</tr>
</tbody>
</table>
```
Issues

- Computation (SQL functions) and data selection (WHERE) on large-scale, high-velocity data
  - Can a database scale to store a non-stop firehose of data?
  - Can a database handle queries and functions on top of the firehose?
- Motivates
  - Non-standard data storage
    - NoSQL, Distributed key-value stores
  - Non-standard querying, parallel data stream processing approaches
    - Map-Reduce, Hadoop, Spark and other data processing pipelines
  - "Data Science"
    - Will revisit both when we get to cloud topics
Object-Relational Mapping
(Write Python instead of SQL)
ORMs

- Take objects and map them directly to relational database
  - SQL taken care of by underlying library
- Sacrifice a little bit of performance and SQL purity for faster development
  - Developers do not have to learn a second language!
  - Potentially cleaner code
  - Minimal SQL injection risks
- Unify insertion, querying, updating and deleting to/from SQL, NoSQL, and NewSQL databases
  - Will see examples in GCP cloud database APIs
import sqlite3
conn = sqlite3.connect('example.db')
c = conn.cursor()
c.execute('CREATE TABLE person
(id INTEGER PRIMARY KEY ASC, name varchar(250) NOT NULL)
')
c.execute('CREATE TABLE address
(id INTEGER PRIMARY KEY ASC, street_name varchar(250), street_number varchar(250),
    post_code varchar(250) NOT NULL, person_id INTEGER NOT NULL,
    FOREIGN KEY(person_id) REFERENCES person(id))
')
```python
class Person(Base):
    __tablename__ = 'person'
    # Here we define columns for the table person
    # Notice that each column is also a normal Python instance attribute.
    id = Column(Integer, primary_key=True)
    name = Column(String(250), nullable=False)

class Address(Base):
    __tablename__ = 'address'
    # Here we define columns for the table address.
    # Notice that each column is also a normal Python instance attribute.
    id = Column(Integer, primary_key=True)
    street_name = Column(String(250))
    street_number = Column(String(250))
    post_code = Column(String(250), nullable=False)
    person_id = Column(Integer, ForeignKey('person.id'))
    person = relationship(Person)

# Create an engine that stores data in the local directory's
# sqlalchmy_example.db file.
engine = create_engine('sqlite:///sqlalchemy_example.db')

# Create all tables in the engine. This is equivalent to "Create Table"
# statements in raw SQL.
Base.metadata.create_all(engine)
```
SQL vs. SQLAlchemy: Insertion

- Insert into DB with SQL (sqlite3)

```python
    with connection:
        c = connection.cursor()
        c.execute(""
            INSERT INTO person VALUES(1, 'pythoncentral')
        "")
        c.execute(""
            INSERT INTO address VALUES(1, 'python road', '1', '00000', 1)
        "")
        conn.commit()
        conn.close()
```

- Insert into DB via ORM

```python
    from sqlalchemy.orm import sessionmaker
    Session = sessionmaker(bind=engine)
    session = Session()
    # Insert a Person in the person table
    new_person = Person(name='new person')
    session.add(new_person)
    session.commit()
    # Insert an Address in the address table
    new_address = Address(post_code='00000', person=new_person)
    session.add(new_address)
    session.commit()
```
**SQLAlchemy example: Querying**

- SQL (sqlite3)

```python
import sqlite3
conn = sqlite3.connect('example.db')
c = conn.cursor()
c.execute('SELECT * FROM person')
print(c.fetchall())
c.execute('SELECT * FROM address')
print(c.fetchall())
conn.close()
```

- SQLAlchemy

```python
>>> session = DBSession()
>>> # Make a query to find all Persons in the database
>>> session.query(Person).all()

>>> # Find all Address whose person field is pointing to the person object
>>> session.query(Address).filter(Address.person == person).all()
```
Database queries via REST/JSON
(Write URLs instead of SQL)

Web APIs, Web EndPoints
Programmatically call into database via Web APIs (EndPoints)

- Key for modern client-side web frameworks
  - Directly interacting with "model"
- Explosive growth
  - Protecting an estimated $2.2 billion in assets
- Each API with 5-6 versions per year (PeachTech)

![Growth in Web APIs since 2005](image-url)
Growth in Web API security issues

Security

Instagram's leaky API exposed celebrities' contact details

This could be how Justin Bieber's bare butt popped out

By Richard Chirgwin 31 Aug 2017 at 02:06

Instagram is blaming a bug in its API for the partial breach of verified users' accounts.
Web APIs/Endpoints

- 2 parts
  - API (REST, GraphQL)
  - Data-exchange format (JSON, gRPC)
REST

- Representational State Transfer
- Style of web software architecture that simplifies application
- Not a standard, but a design pattern
REST APIs

- Take all resources for web application (data, files, functions)
  - Identify each resource and action on resource via an HTTP method and URL.
  - Method selects action
  - Send arguments via the HTTP request (e.g. in URL, URL parameters, or payload)
REST toy example

- http://foo.com/bar/users
  - Server foo.com
  - Database bar
  - Table users
  - URL returns table users in database bar in a particular format (XML, JSON)
- Effectively turn URLs into precanned SQL
- Common examples
  - Twitter, Flickr, Amazon
REST and HTTP methods

- HTTP request methods indicate the desired action
- **GET**
  - Requests a representation of the specified resource.
  - Use for operations that have NO side-effects (safe operations)
  - Works with robots and crawlers.
- **POST**
  - Submits data to be processed (e.g., from an HTML form) to the identified resource. Data is included in the body of the request.
- **PUT**
  - Uploads a representation of the specified resource.
- **DELETE**
  - Deletes the specified resource.
JSON

- **JavaScript Object Notation**
  - De-facto web object data format
    - Subset of JavaScript
    - Minimal, lightweight, text-based syntax
    - Easy to parse and generate
  - Prevalent in most web sites and web APIs, often as part of a REST architecture
  - Designed to enable stateful, real-time communication between browser and web application
    - Often used via AJAX to allow web server to directly modify elements of a page without refresh
**JSON objects**

- Objects are unordered containers of key/value pairs similar to Python dictionaries
  - Keys are strings
  - Values are JSON values
  - Wrapped in `{ }`
    - `,` separates key/value pairs
    - `:` separates keys and values
-Parsed into local data structures as struct, record, hashtable, or dictionary
JSON Values

- **Strings**
  - Sequence of 0 or more Unicode characters wrapped in double quotes
- **Numbers**
  - Integer, Real, Scientific
  - No octal or hex
  - No `NaN` or `Infinity` (Uses `null` instead!)
- **Booleans**
  - `true`, `false`
  - `null`
    - A value that isn't anything
- **Objects**
- **Arrays**
JSON example

{
    "firstName": "John",
    "lastName": "Smith",
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumbers": [
        "212 555-1234",
        "646 555-4567"
    ]
}

Name/Value Pairs
Child properties
String Array
Number data type
JSON issues

- Text format
  - Binary data must be encoded (Base64)
  - Space overhead in using text
- Flat format
  - Object treated as a string
  - Must be parsed to obtain original object (e.g. JSON.parse())
  - Processing overhead in going to/from string and into programming-language specific data structure
gRPC

- Initially developed by Google as ProtoBufs (protocol buffers)
- Support serialization of structured data in an arbitrary (potentially binary) format
  - Derived from FlatBuffer's zero-copy deserialization idea
  - Minimal processing to go from wire-line format to language-specific representation
  - ProtoBuf generators for C++, Java, C#, Python, etc.
- Each protobuf contains an interface description language specifying format of the stream of bytes transmitted
- Uses HTTP/2 as a transport
- Used throughout Google's infrastructure, but now open-sourced
  - Mandatory for certain cloud services that are data-intensive (e.g. BigQuery)
- When to use…
  - [https://codeclimate.com/blog/choose-protocol-buffers/](https://codeclimate.com/blog/choose-protocol-buffers/)
Issues with REST

- URLs into SQL
- Problem
  - Must provide a fixed mapping between what app wants and what the API actually implements
- Overfetching
  - API returns much more data than user asks for
- Underfetching
  - API must be queried multiple times to get all the data user needs
    - Example: queries to different tables in a backend require multiple API requests
- Slows down DevOps
  - Front-end developers must coordinate with backend engineers to get additional API endpoints defined when needed
GraphQL

- Facebook (2015)
- REST mapping to SQL query too coarse
  - Rigid queries based on URL and a JSON result that must then be parsed
  - Queries on multiple tables require multiple requests
  - Results returned as text that must be parsed
- GraphQL
  - Language that allows client to specify details of what it wants
    - Give front-end programmable control over query
  - Specify the structure of how the result data should be returned
    - Helps strongly typed client code ingest data
Cloud SQL

AWS RDS (Relational Database Service)
Azure SQL Database
Google Cloud SQL

- Fully-managed, drop-in replacement for MySQL (or Postgres) relational database
  - Hosted by Google
  - No need to reconfigure code
- Secure
  - No access to database from outside project unless specified
  - Data encrypted in flight and at rest
- Automatic backups and software updates
Labs
Databases Lab #1

Do this quiz

- [http://www.w3schools.com/sql/sql_quiz.asp](http://www.w3schools.com/sql/sql_quiz.asp)
- Screenshot of "Check your answers"
- Does not have to be perfect (mine wasn't), but check the ones you got incorrect
Databases Lab #2 (Cloud SQL)

- Create a Managed MySQL database with Cloud SQL
- Set-up and query using a Cloud MySQL instance
Setup

- Download data

```
    git clone https://github.com/GoogleCloudPlatform/training-data-analyst
```

- Examine schema and data in CSV (comma separated values) format

```
    cd training-data-analyst/CPB100/lab3a
    less cloudsql/table_creation.sql
    head cloudsql/*.csv
```

- Stage data definition file and values in a storage bucket so it can be imported by Cloud SQL service
  - Create a bucket if you've deleted your prior ones

```
    gsutil mb -c regional -l us-west1 gs://<BUCKET-NAME>
    gsutil cp cloudsql/* gs://<BUCKET-NAME>/sql/
```

- Verify files are in Cloud Storage bucket via web console
Create a new Cloud SQL instance

- In the Google Cloud Console, scroll down and select SQL in the Storage subsection
Create a new Cloud SQL instance

- Choose MySQL (Second Generation)
- Name the Instance ID: rentals
- Set and remember the root password for the database
- Place in us-west1-b
Configure access to instance from Cloud Shell

- In Cloud Shell, find the external IP address of the shell
  ```
  curl http://ipecho.net/plain; echo
  ```
- Within Cloud SQL, click on the instance, then on "Edit instance", then on "Authorize networks", then on "Add Network"
Configure access to instance from Cloud Shell

- Enter the IP address found previously as a CIDR route
  - e.g. 131.252.220.66/32
- Click on "Done", then "Close"
- Ensure it appears in "Authorized Networks", then "Save"
Import table definitions and tables (from Cloud Storage bucket)

- Click on the hyperlink named rentals (i.e. your Cloud SQL instance name)
- Click on Import (on the top menu bar)
- Click on the Browse button and browse to your storage bucket containing the SQL lab files
- Navigate to table_creation.sql
- Select and click Import.
Populate tables

- Import CSV files from Cloud Storage
- Within Cloud SQL instance, click on Import (top menu)
- Click Browse, then find and select `accommodation.csv`
- Fill out the rest of the dialog
  - Database = `recommendation_spark`
  - Table = `Accommodation`
Use Cloud SQL

- Connect to instance
  
  ```
  mysql --host=<MySQLIP> --user=root --password
  ```

- Set database for MySQL session
  
  ```
  use recommendation_spark;
  ```

- View tables created
  
  ```
  show tables;
  ```

- Verify data
  
  ```
  select * from Rating;
  ```

- Run queries for accommodations at various price levels and types
  
  ```
  select * from Accommodation where...
  ```
Databases Lab #2 link

- [https://codelabs.developers.google.com/codelabs/cpb100-cloud-sql](https://codelabs.developers.google.com/codelabs/cpb100-cloud-sql) (22 min)
Homework #3

Modify your toy Python Flask web application to support review submission and database backend support

- Within private repository, create a directory "hw3" for your application
- Application as before, but application no longer hard-codes the reviews in model script
- Instead, implement a route and form for creating reviews by updating your model and controller/presenter code
- Modify model script to support creation and insertion of reviews to a backend sqlite3 database
- Commit changes to Bitbucket and tag the repository with hw3
Extra
Databases Lab (extra) (Cloud SQL)

- Create a Managed MySQL database with Cloud SQL
- Set-up and query using a Cloud MySQL instance
- [https://codelabs.developers.google.com/codelabs/cloud-create-cloud-sql-db](https://codelabs.developers.google.com/codelabs/cloud-create-cloud-sql-db) (19 min)
In the Google Cloud Console, scroll down and select SQL in the Storage subsection.
- Create new instance
  - Specify a name for your instance, then click **Create**
    - Must be unique within your project, but not across projects
    - Enter "codelab-0" into the Instance ID box.
    - Creation may take a few minutes
- Configure database credentials
- In the instance, click on **Access Control > Users > Change Root Password**
- Connect to instance via Cloud Shell

- `gcloud beta sql command` (update if no longer beta)

```shell
$ gcloud beta sql connect codelab-0 --user=root

Whitelisting your IP for incoming connection for 1 minute...done.
Enter password:

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 11302
Server version: 5.6.29-google-log (Google)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective Owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```
Create database in instance
  
  Via SQL

```
mysql> CREATE DATABASE codelab;
Query OK, 1 row affected (0.00 sec)
```

Exit

```
mysql> exit;
```
• **Load the sample data into database**
  • Download the scripts and data

```
$ wget https://codeload.github.com/datacharmer/test_db/zip/master -0 sampledb.zip
$ unzip sampledb.zip && cd test_db-master

$ gcloud beta sql connect codelab-0 --user=root < employees.sql
```
• Query the database
  • Show the output of queries

```sql
mysql> USE employees;

mysql> SELECT avg(s.salary) avg_salary_by_hire_year,
           YEAR(e.hire_date) FROM employees e, salaries s WHERE
           e.emp_no = s.emp_no GROUP BY YEAR(e.hire_date);

mysql> SELECT de.dept_no, sum(s.salary) sum_salaries_per_department FROM employees e, salaries s,
           dept_emp de WHERE e.emp_no = de.emp_no AND e.emp_no =
           s.emp_no GROUP BY de.dept_no;
```
• **Delete instance**
  • Via command-line

```
$ gcloud beta sql instances delete codelab-0
```

• Via web console