Week 1: SQL and NoSQL

DIANE WOODBRIDGE, PH.D

About Diane

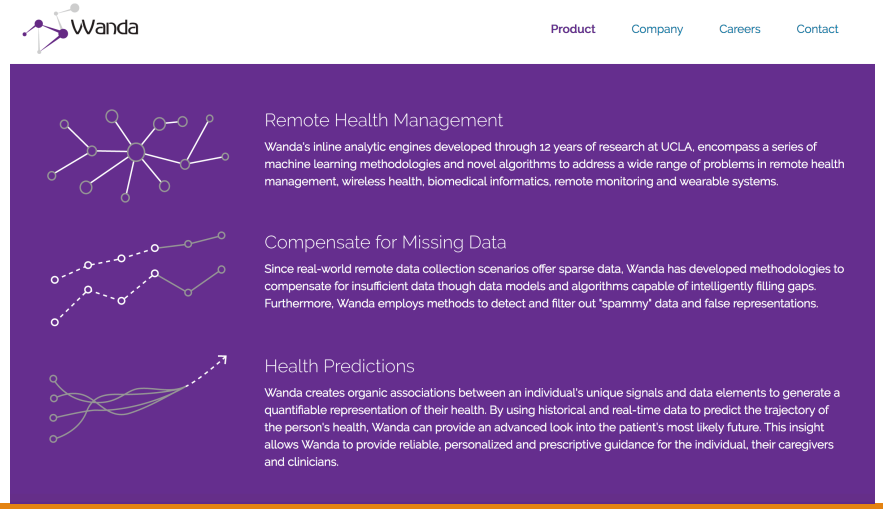
Technical Me

Ph.D. (and M.S) in Computer Science, UCLA. 2012, 2010

B.S. in Computer Science, Sogang Univ, South Korea. 2007

SMTS, Sandia National Laboratories. 2012 – 2016

Research Interest: Database management and data mining in health, remote sensing, seismicity, etc.



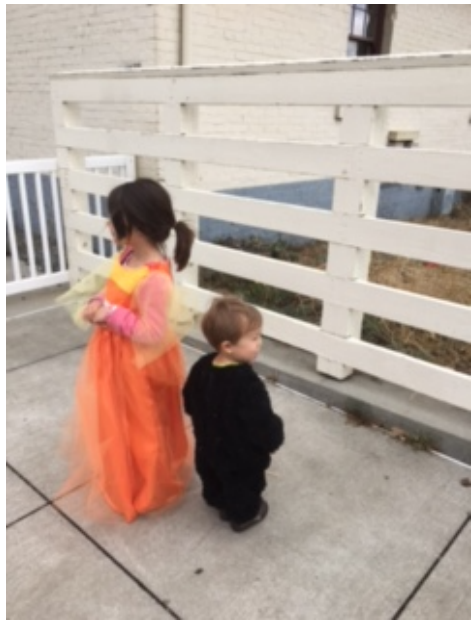
Non-Technical Me

Married.

Two kids.

One cat.

Yoga, Sewing, Cooking, Musical.

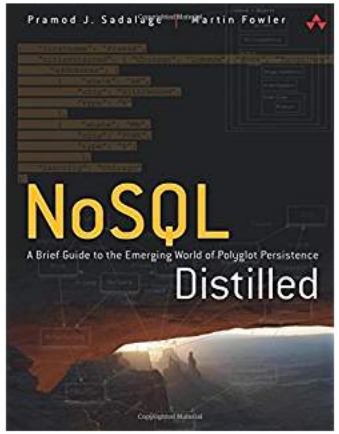


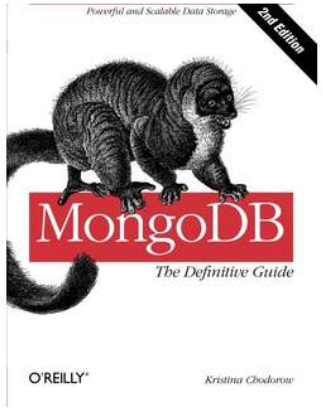
Objectives of This Course

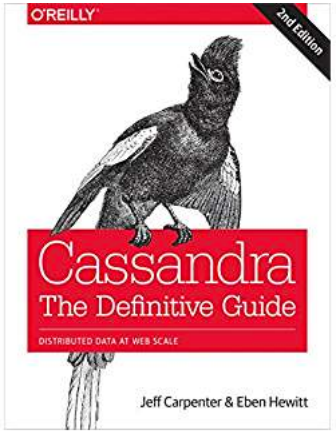
Understand the needs and characteristics of schemaless non-relational databases for storing and processing data in distributed computing settings.

Gain experience with NoSQL databases including MongoDB and Cassandra through homework and in-class exercises.

Gain insights to make judgments to choose SQL or NoSQL (and which NoSQL) with various hands-on exercises.







Evaluation Criteria

Attendance and Participation - 5 %

- No Cellphone
- No web-surfing
- Q&A

Programming Assignment - 15 x 3 %

Quiz - 20 %

Final - 30 %

Class Info

Office Hours: Tue 12:00 - 1:00 pm, Howard 101 Room 522. Plus, anytime when my door is open.

Final Exam: Dec 9, 10 AM - 12 PM, 101 Howard 154, 155 and 156

Schedule

- Week 1 - SQL and NoSQL
- Week 2 - NoSQL Overview
- Week 3 - Document Database (MongoDB), HW 1 Due
- Week 4 - Document Database (MongoDB), Quiz
- Week 5 - Column Family Database (Cassandra), HW 2 Due
- Week 6 - Thanksgiving Break
- Week 7 - Column Family Database (Cassandra), HW 3 Due

Poll

Please turn on:

https://pollev.com/dianewoodbri311

ANSWER?
https://PollEv.com/multiple_choice_polls/wMNOKfamXPd9kYf/web

Relational Database

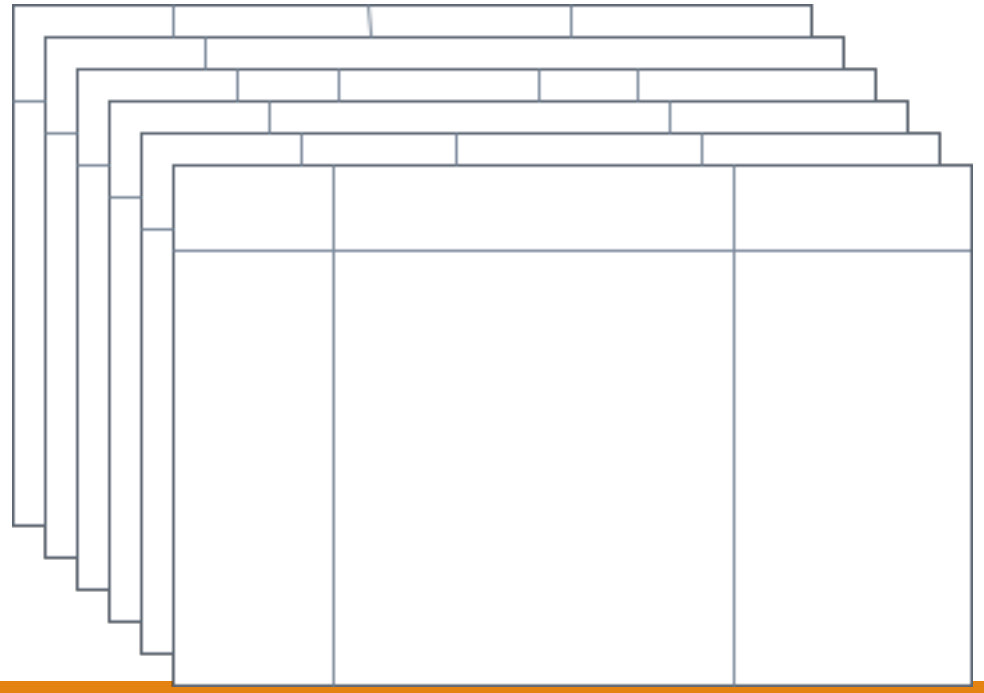
Relational Model

- Database is a collection of relations.

Relational Database

Relational Model

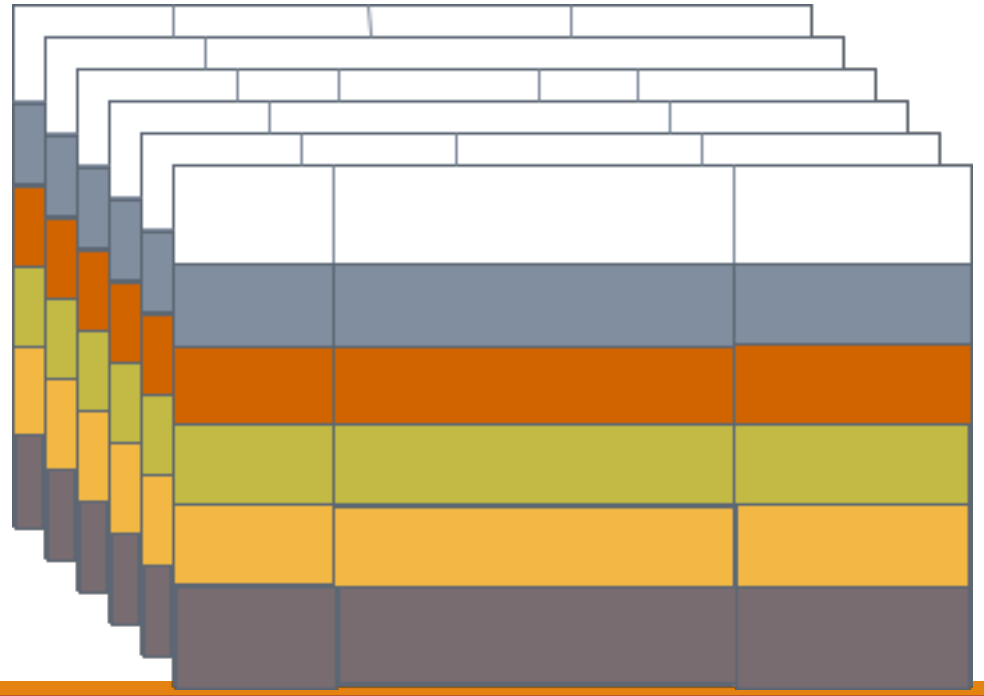
- Database is a collection of relations
- Each relation has attributes



Relational Database

Relational Model

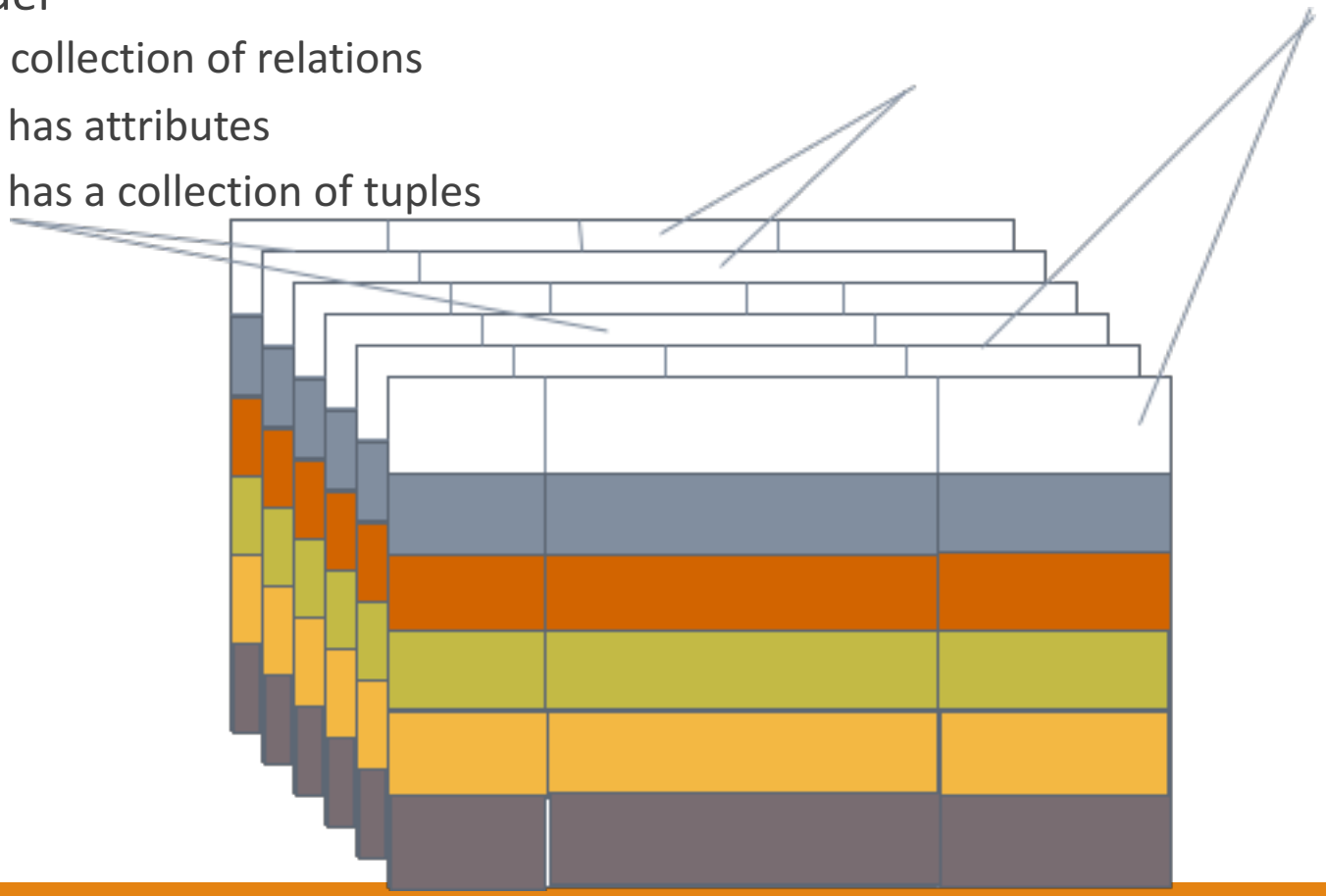
- Database is a collection of relations
- Each relation has attributes
- Each relation has a collection of tuples



Relational Database

Relational Model

- Database is a collection of relations
- Each relation has attributes
- Each relation has a collection of tuples



Relational Database

SQL (Structured Query Language)

- Manage data in a relational database
- Selects rows from a relation satisfying a given condition.
- Basic structure
  - SELECT, FROM, WHERE

SELECT DATA _FIELD

FROM TABLE

WHERE CONDITIONS

Ex) CUSTOMER

ID	Name	Email

EX)
```
SELECT *
FROM Customer
WHERE Name = "Diane Woodbridge"
```

Pros : Relational Database

Concurrency Control

- ACID (Atomic, Consistent, Isolated and Durable) Transaction Management
  - Atomicity :
    - An operation either succeeds or fails entirely.
    - Many rows spanning many tables are updated as a single operation.
  - Consistency:
    - Any given transaction must change affected data only in allowed ways.
  - Isolation:
    - Defines how/when the changes made by one operation become visible to other.
    - Concurrent operations are isolated from each other so that they can't see a partial update.
  - Durablility:
    - Once a transaction has been committed, it will remain permanently.

https://en.wikipedia.org/wiki/ACID

Pros : Relational Database

Standard Model

- Different vendors' query languages are similar.
- Transaction operations work in the similar way.

Relational Database

Important concepts to remember (Interview Questions!)

- Basic Operations
  - Create
  - Insert
  - Select
  - Update
  - Join (Inner, Outer, Left, Right, etc.)*
  - Union All/Union/ Union Distinct
  - Minus
  - Intersect
- Normalization (1NF, 2NF and 3NF)*
- Transaction(Concurrency Control) – ACID*
- Indexing* - B tree, Hash
- Truncate vs Delete
- Difference Where vs. Having

Relational Databases

Create tables

- CREATE TABLE *table_name*
  (
  column_name1 data_type(*size*),
  column_name2 data_type(*size*),
  column_name3 data_type(*size*),
  ....
  );

```
dwoodbridge=# create table customer(name varchar, id char(10), email varchar);
CREATE TABLE
```

Relational Databases

Insert rows

- INSERT INTO *table_name* (*col1*,*col2*,...) VALUES (*value1*,*value2*,...);

```
dwoodbridge=# insert into customer(name, id, email)  values ('Diane Woodbridge',
 1234567890, 'dwoodbridge@usfca.edu');
INSERT 0 1
```

Select rows

- SELECT *column_name*,*column_name* FROM *table_name*;

```
dwoodbridge=# select * from customer where email = 'dwoodbridge@usfca.edu';
       name       |     id     |         email
------------------+------------+-----------------------
 Diane Woodbridge | 1234567890 | dwoodbridge@usfca.edu
(1 row)
```

Update rows

- UPDATE *table_name* SET *column1=value1*,*column2=value2*,... WHERE *some_column=some_value*;

```
dwoodbridge=# update customer set name='Diane MK Woodbridge' where email='dwoodb
ridge@usfca.edu';
UPDATE 1
```

Delete rows

- DELETE FROM *table_name* WHERE *some_column=some_value*;

```
dwoodbridge=# delete from customer where true;
DELETE 1
```

Relational Databases

Join

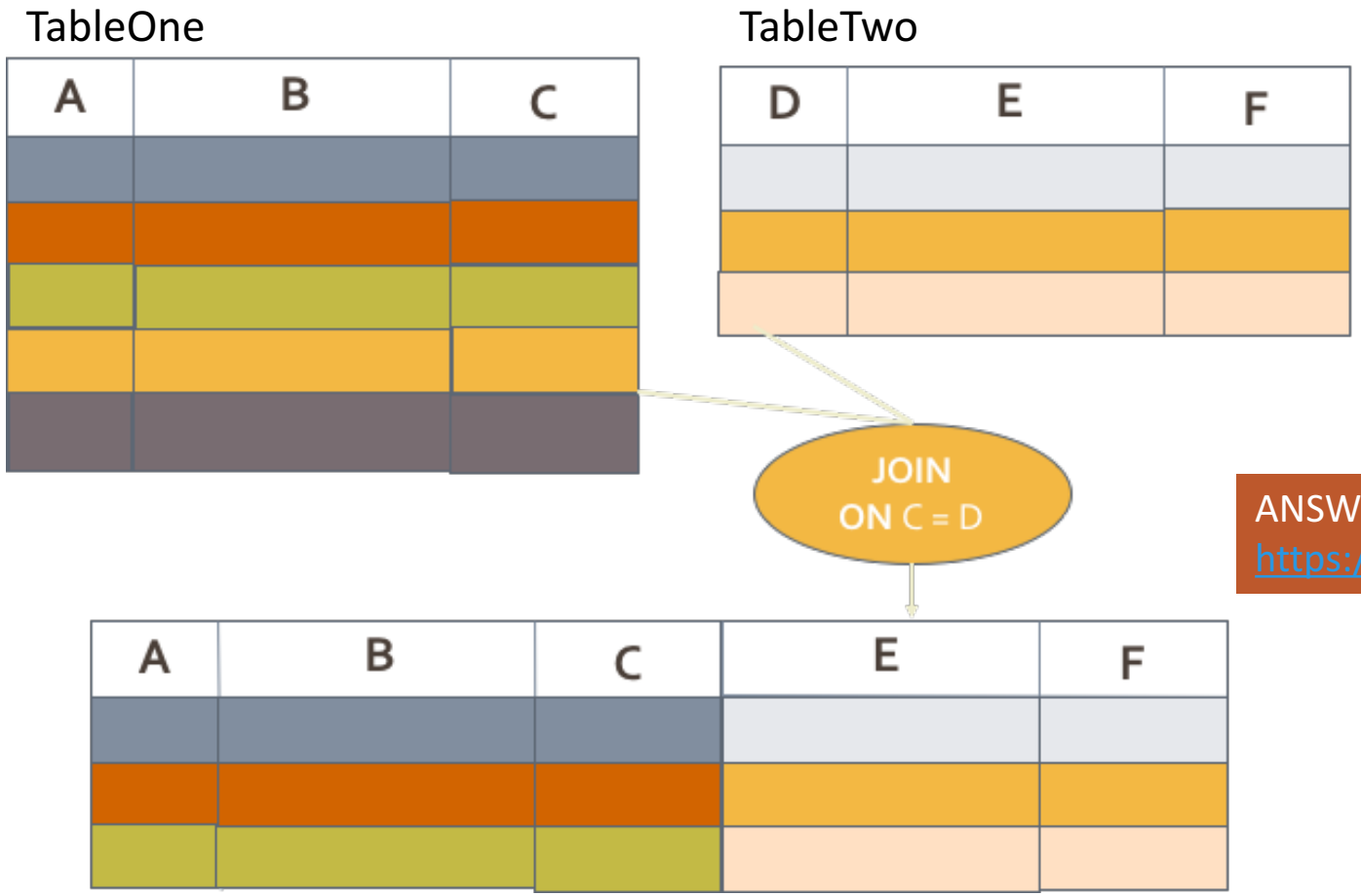
- Purpose
  - Relate rows from two different tables based on constraints
- Input
  - Two tables
- Output
  - Merged rows from two tables satisfying given constraints

Relational Databases

Join

- SELECT *column_name(s)*
  FROM *table1*
  JOIN_TYPE *table2*
  ON *table1.column_name=table2.column_name*;

Relational Databases

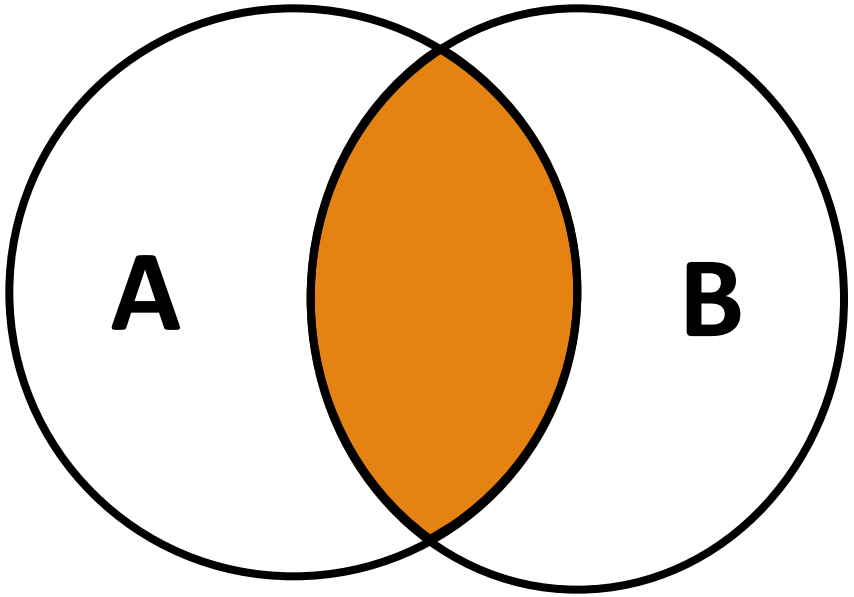


ANSWER?
https://pollev.com/dianewoodbri311

Relational Databases

Different types of joins

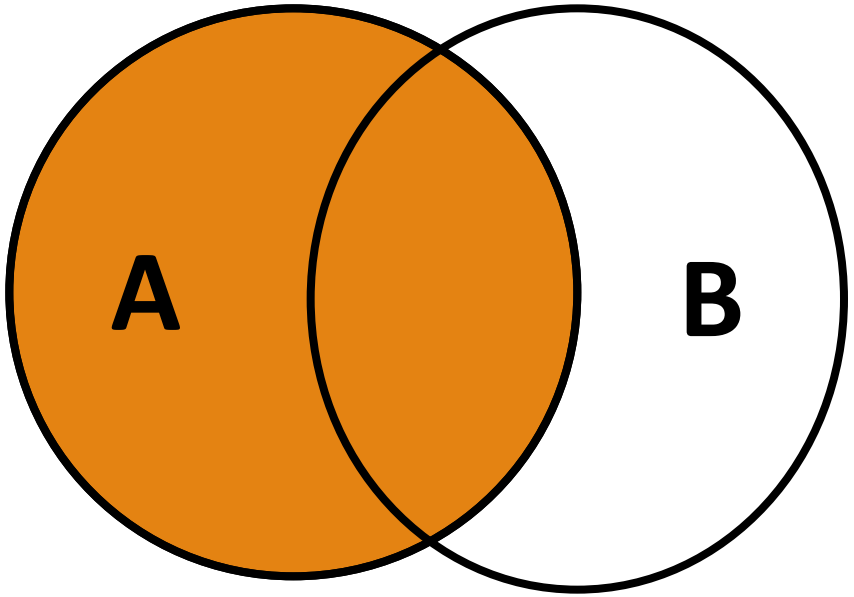
- Inner Join (= Join)
  - Returns rows appearing in both tables.
- Left Join
  - Return all rows from the left table, and the matched rows from the right table.
- Right Join
  - Return all rows from the right table, and the matched rows from the left table.
- Full Join (Full Outer Join)
  - Return all rows appearing in any of the tables.



Relational Databases

Different types of joins

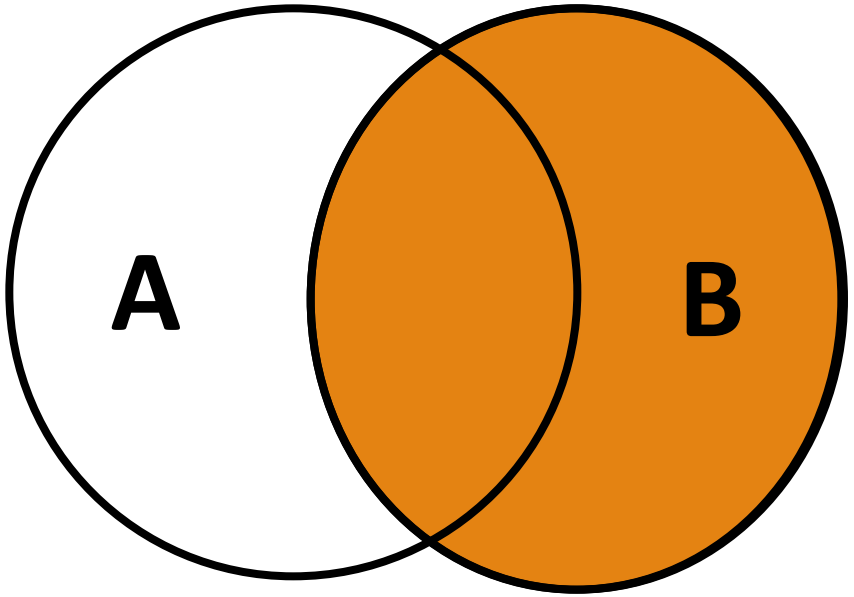
- Inner Join (= JOIN)
  - Returns rows appearing in both tables.
- Left Join
  - Return all rows from the left table, and the matched rows from the right table.
- Right Join
  - Return all rows from the right table, and the matched rows from the left table.
- Full Join (Full Outer Join)
  - Return all rows appearing in any of the tables.



Relational Databases

Different types of joins

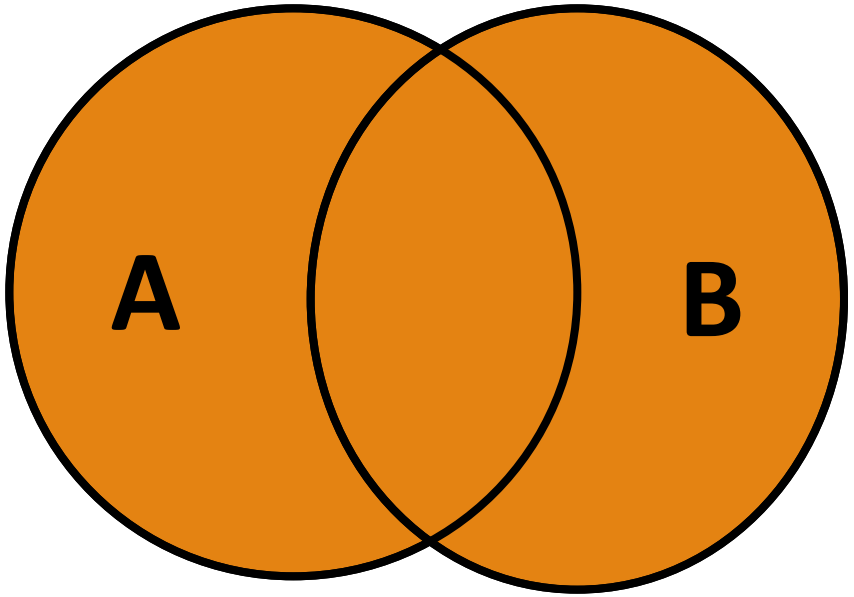
- Inner Join (= JOIN)
  - Returns rows appearing in both tables.
- Left Join
  - Return all rows from the left table, and the matched rows from the right table.
- Right Join
  - Return all rows from the right table, and the matched rows from the left table.
- Full Join (Full Outer Join)
  - Return all rows appearing in any of the tables.



Relational Databases

Different types of joins

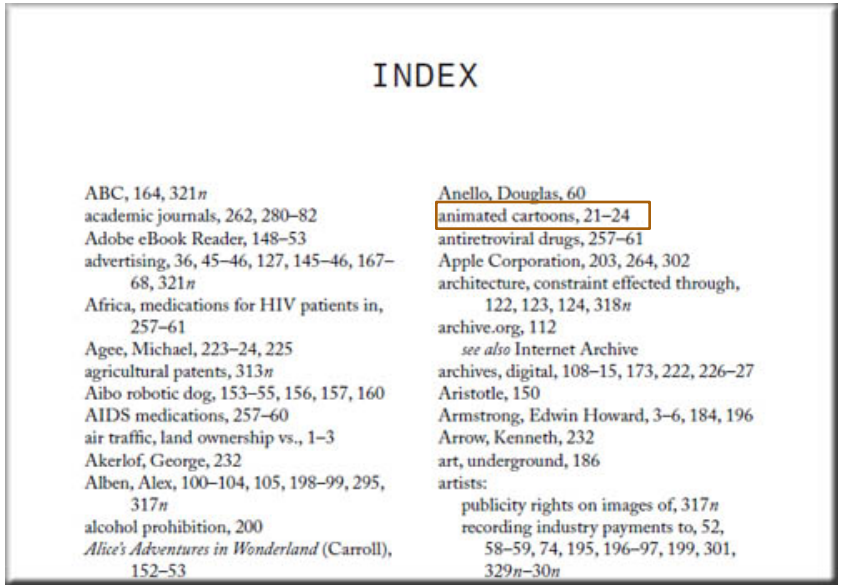
- Inner Join (= JOIN)
  - Returns rows appearing in both tables.
- Left Join
  - Return all rows from the left table, and the matched rows from the right table.
- Right Join
  - Return all rows from the right table, and the matched rows from the left table.
- Full Join (Full Outer Join)
  - Return all rows appearing in any of the tables.



Relational Databases

Animated Cartoons?

Relational Databases



INDEX

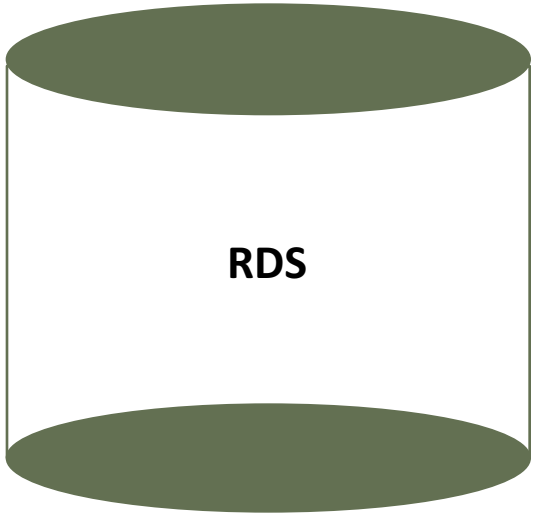
Index

- Organize data by a given field. ➔ Optimize queries and enhance query performance.
- Consider.
  - Which fields to index?
- Popular index scheme : balanced search tree (B-tree), hash, etc.

Relational Databases on AWS

Relational Database Services (RDS)

- Relational DB on Cloud
- Supports Aurora, Oracle, Microsoft SQL Server, PostgreSQL, MySQL and MariaDB.

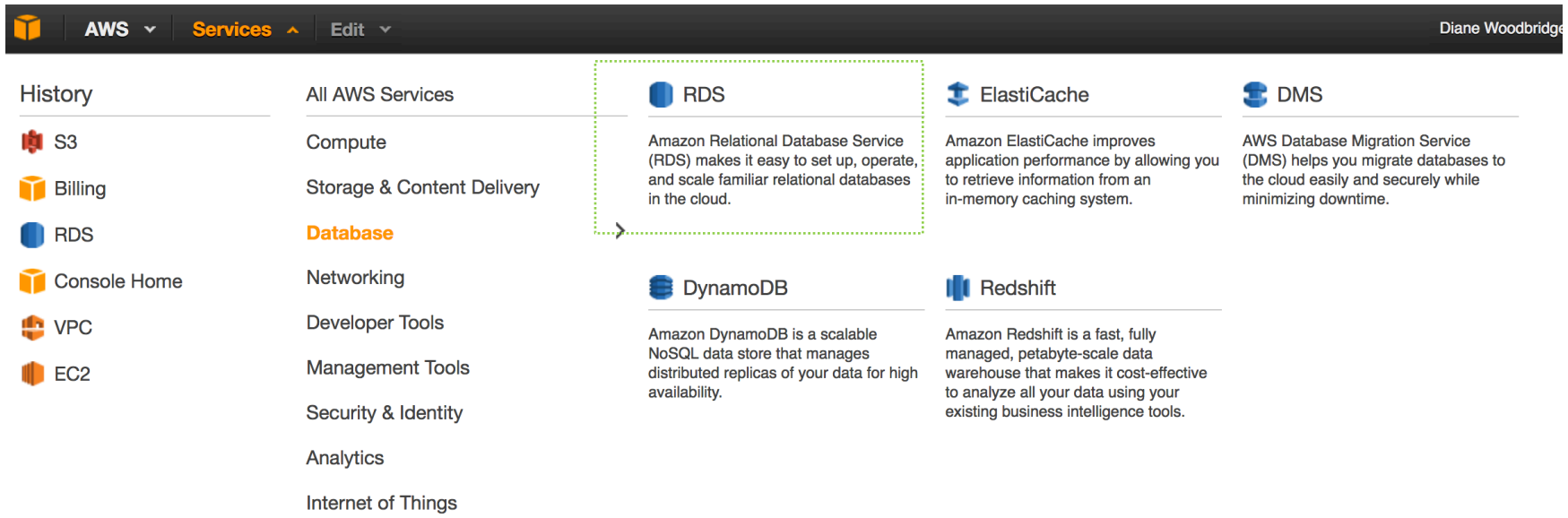


https://aws.amazon.com/rds/

Relational Databases on AWS

Relational Database Services (RDS)

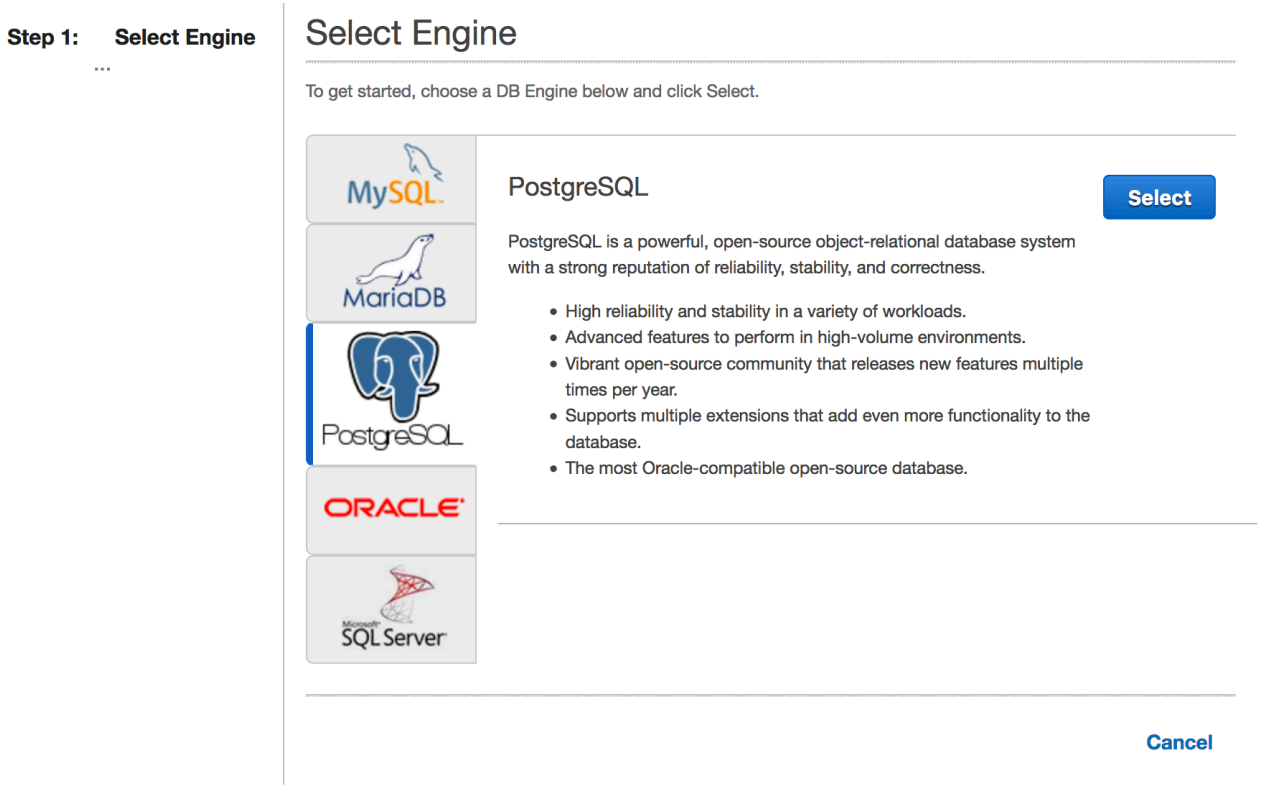
- Relational DB on Cloud
- Supports Aurora, Oracle, Microsoft SQL Server, PostgreSQL, MySQL and MariaDB.



Relational Databases on AWS

Relational Database Services (RDS)

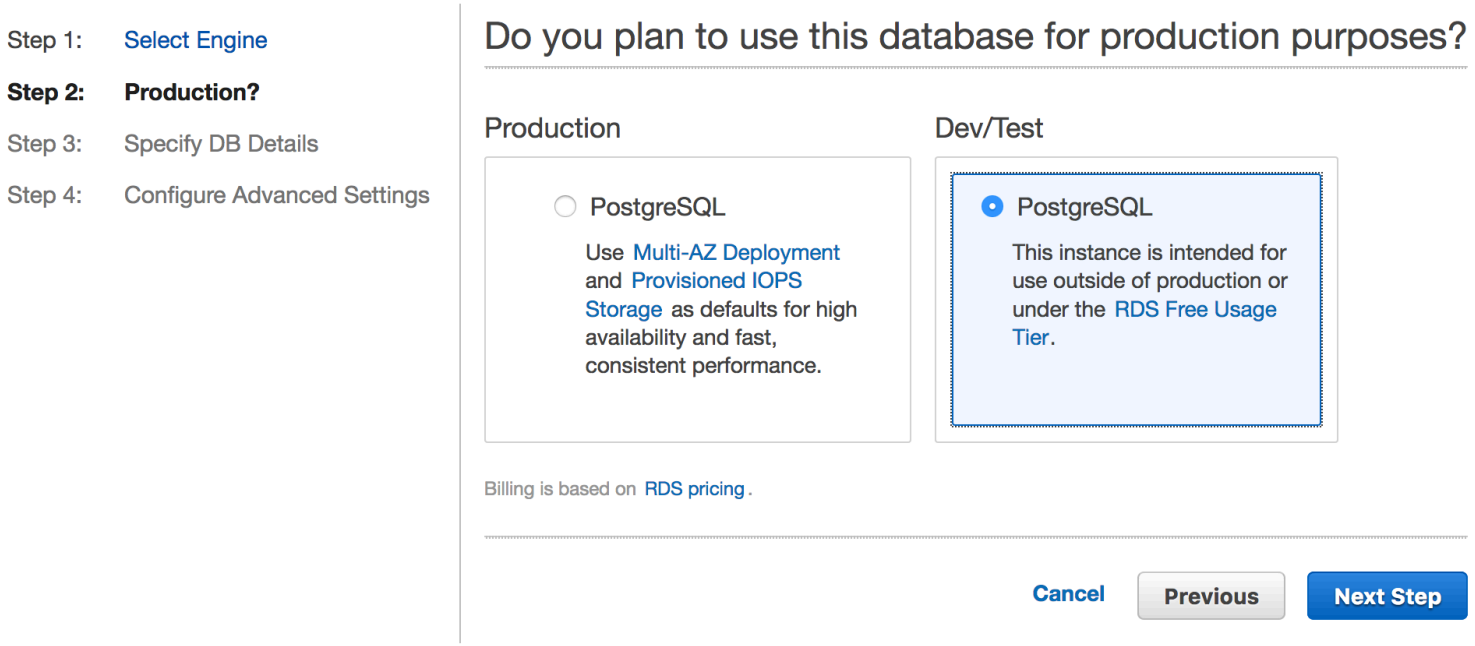
- Relational DB on Cloud
- Supports Aurora, Oracle, Microsoft SQL Server, PostgreSQL, MySQL and MariaDB.



Relational Databases on AWS

Relational Database Services (RDS)

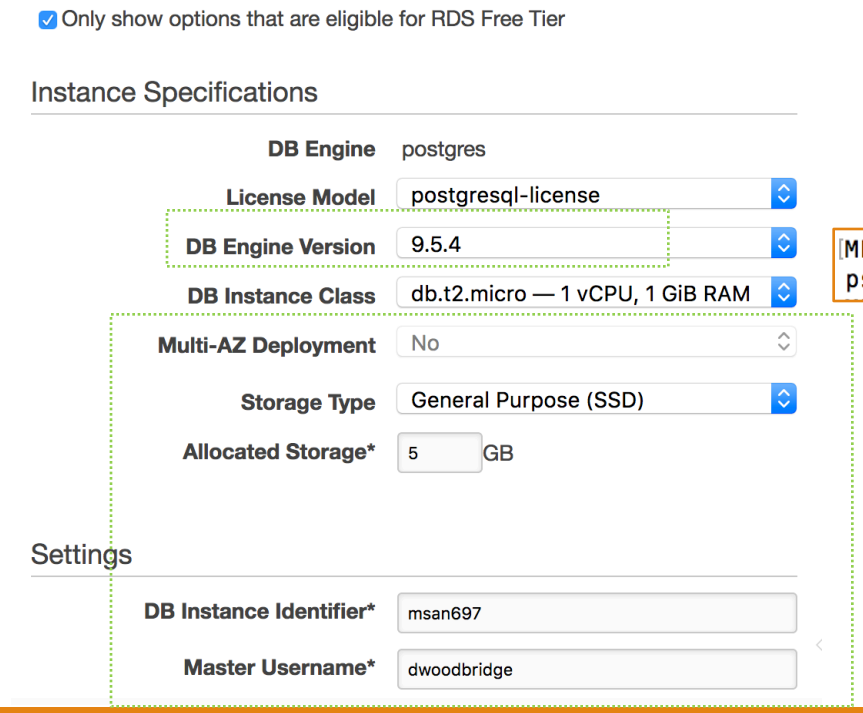
- Relational DB on Cloud
- Supports Aurora, Oracle, Microsoft SQL Server, PostgreSQL, MySQL and MariaDB.



Relational Databases on AWS

Relational Database Services (RDS)

- Relational DB on Cloud
- Supports Aurora, Oracle, Microsoft SQL Server, PostgreSQL, MySQL and MariaDB.



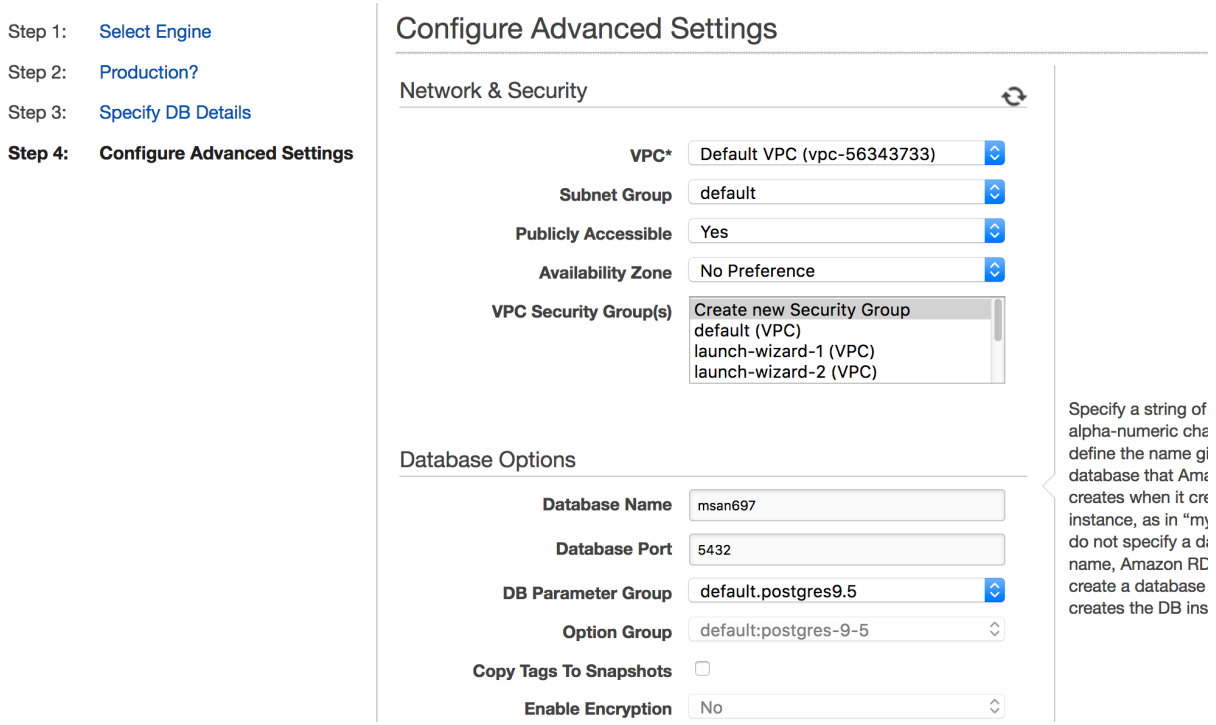
I'll use version 9.5.4 for the class. Make sure that your local version is also 9.5.4.

```
ML-ITS-603436:MSAN697 dwoodbridge$ psql --version
psql (PostgreSQL) 9.5.4
```

Relational Databases on AWS

Relational Database Services (RDS)

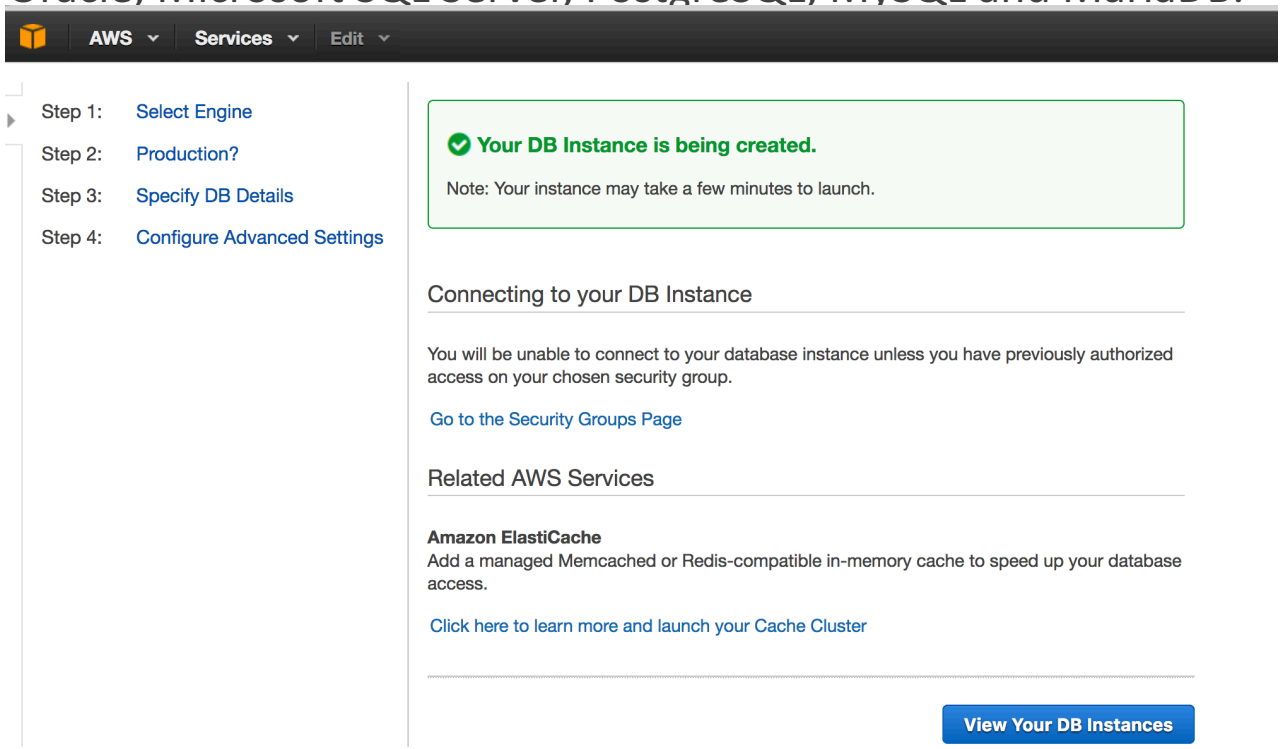
- Relational DB on Cloud
- Supports Aurora, Oracle, Microsoft SQL Server, PostgreSQL, MySQL and MariaDB.



Relational Databases on AWS

Relational Database Services (RDS)

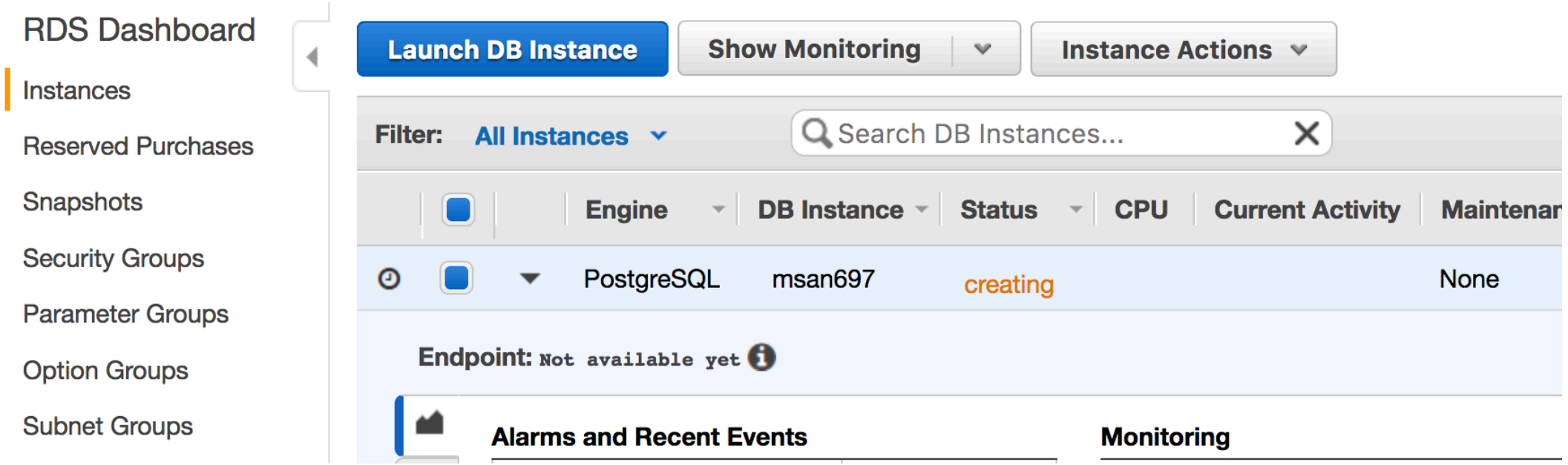
- Relational DB on Cloud
- Supports Aurora, Oracle, Microsoft SQL Server, PostgreSQL, MySQL and MariaDB.



Relational Databases on AWS

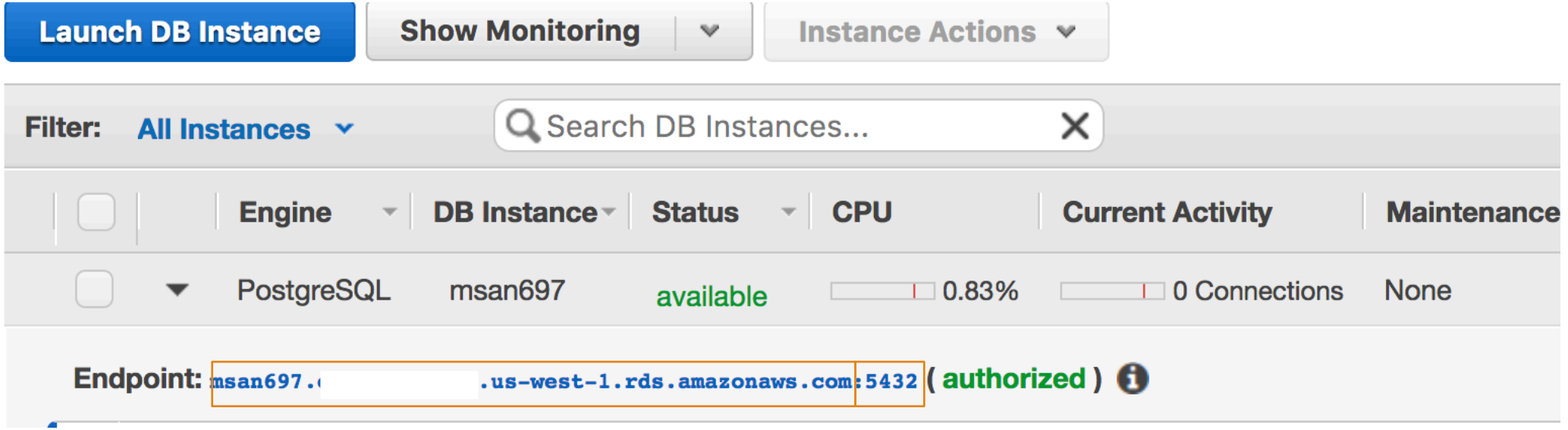
Relational Database Services (RDS)

- Relational DB on Cloud
- Supports Aurora, Oracle, Microsoft SQL Server, PostgreSQL, MySQL and MariaDB.



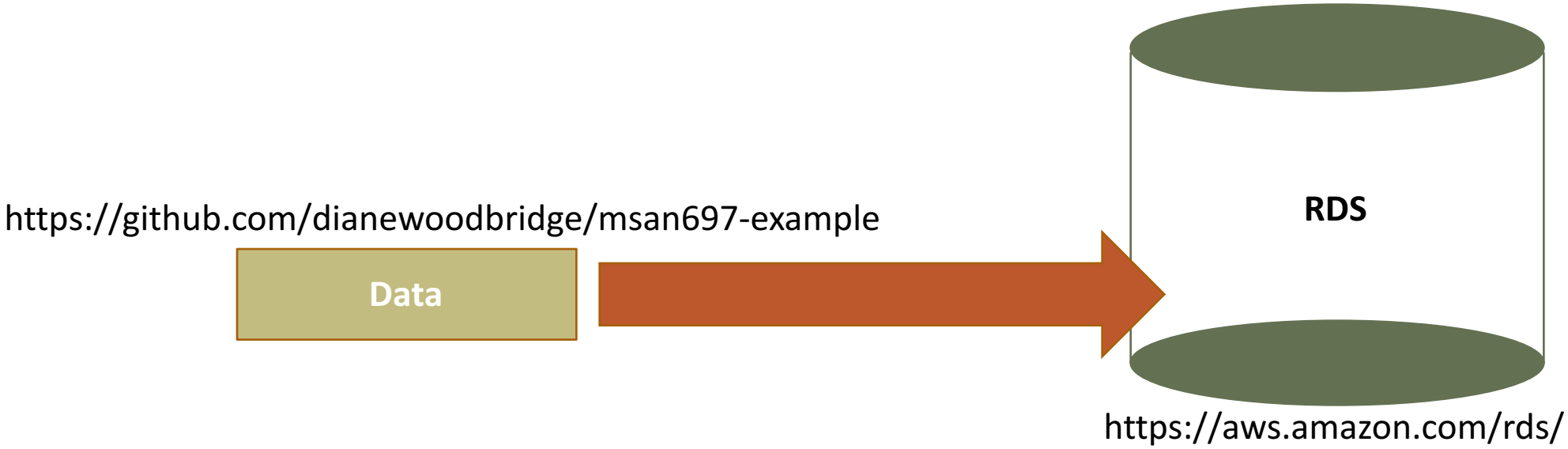
Relational Databases on AWS

Relational Database Services (RDS)



Relational Databases on AWS

Example.
Load data (SFBusiness) in a file to a database.



Relational Databases on AWS

Example.

Load data in a file to a database.

- Access to RDS

```
psql --host msan697.c7hpe.us-west-1.rds.amazonaws.com --port 5432 --username
dwoodbridge --dbname msan697
```

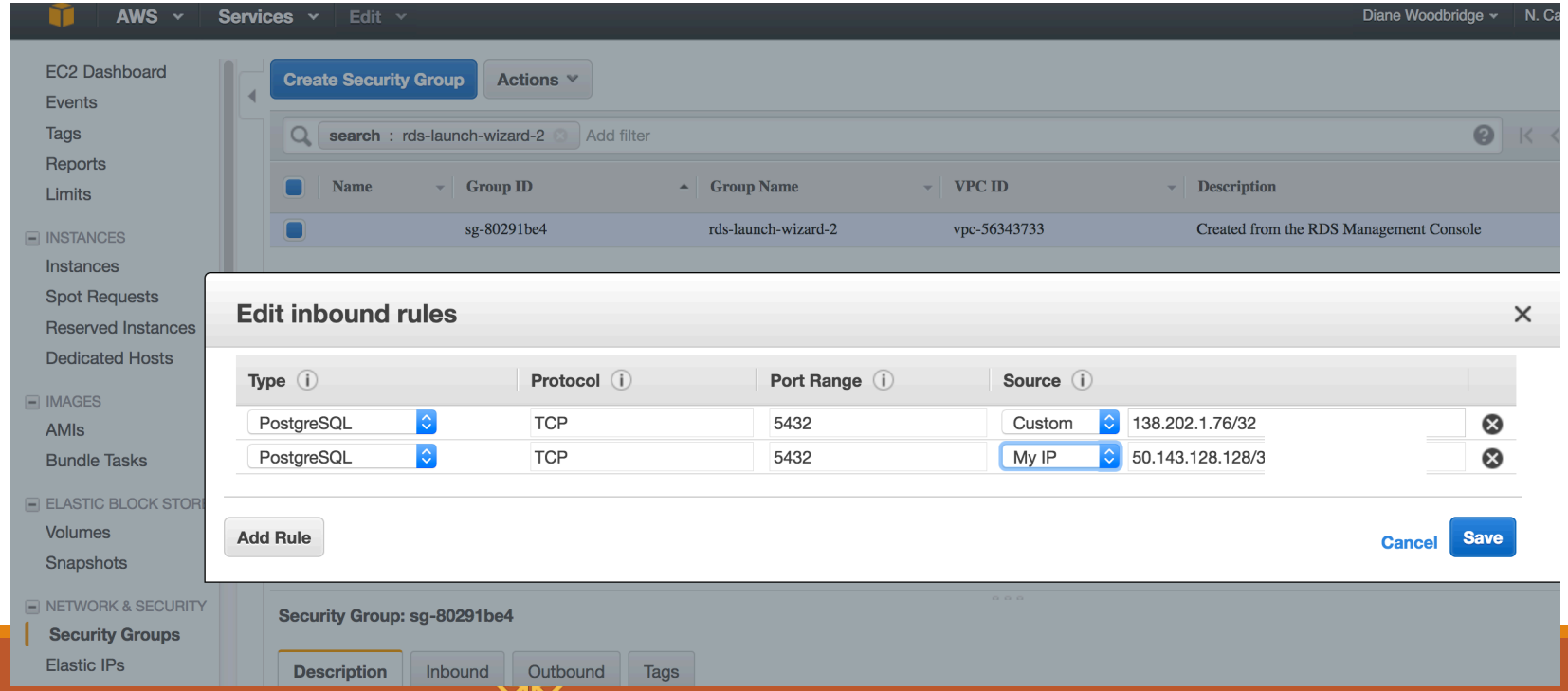
Data :
https://data.sfgov.org/

Relational Databases on AWS

Example.

Load data in a file to a database.

- Access to RDS
  - If you have trouble to connect to aws rds, make sure that it is publicly available and your IP is added to the security group or allow all the access from outside.



Relational Databases on AWS

Example.
Load data in a file to a database.

- Create table

```
CREATE TABLE Businesses(business_id numeric, name char(200), address
char(200), city char(100), state char(10), postal_code char(20), latitude
real, longitude real, phone_number char(200));
```

- Use postgres command \COPY to copy downloaded data to RDS.

```
\COPY Businesses FROM 'businesses.csv' CSV HEADER
```

Data :
https://data.sfgov.org/

Example 1

Repeat it for inspections.csv and violations.csv.

How many records are in each table?

Example 2

What are the names of the businesses and their lowest score, which had “Improper food storage” violations?

Homework 1

https://github.com/dianewoodbridge/msan697-homework.git

Due by November 3rd (Midnight).

NoSQL

Why No SQL?

- Impedence Mismatch
  - Relation model ≠ In-memory data structure (object)
  - For application Development Productivity,
    - Better mapping with in-memory data structures for the application.
- Large volumes of data (2000s)
  - Scaling up vs. Scaling out?
    - Run large data on clusters of many smaller and cheaper machines.
    - Cheaper and reliable.
- Example of non-relational database.
  - Google BigTable and Amaon Dynamo.

NoSQL

Generally,

- Take schemaless data.
- Non-relational.
- Open-source.
- Trade off traditional consistency for other properties.
- Run on clusters.

SQL and NoSQL

SQL

Pros
Persistent data storage.
Concurrency.
Standard Model.

Cons
Impedence mismatch.
Hard to scale.
Fixed schema.

Example
MySQL, PostgreSQL,
SQL Server, Oracle

VS

NoSQL

Pros
Mostly open-source.
Schemaless.
Good for non-relational data.
Scalable.
Runs well on distributed systems.

Cons
Installation, toolsets still maturing.

Example
Redis, MongoDB, Cassandra,
OrientDB

Aggregate-oriented NoSQL Databases

Aggregate: Collection of related objects treated as a unit.

- For analyzing data, you might want to place some data together as an aggregate.
- On a cluster, an aggregate is stored together on a node.
- Aggregate-oriented databases use aggregates indexed by key for data lookup.
- A single aggregate is a unit of atomic updates.
- Aggregate-oriented databases don't have ACID transactions that span multiple aggregates.
- RDBMS is aggregate-ignorant.


```
{
  "_links":{
     "self":{
        "href":"/member/109087/cart"
     },
     "http://example.com/rels/payment":{
        "href":"/payment"
     }
  },
  "_embedded":{
     "http://example.com/rels/cart-item":[
        {
           "_links":{
              "self":{
                 "href":"/member/109087/cart/14418796"
              }
           },
           "id":"14418796",
           "quantity":1,
           "expire_time":"2009-09-11T08:00:00-07:00",
           "_embedded":{
              "http://example.com/rels/sku":[
                 {
                    "_links":{
                       "self":{
                          "href":"/skus/654654"
                       },
                       "http://example.com/images/cart/item":"http://example.com/product/6895/thumbnail.jp
                    },
                    "color":"Black",
                    "size":"S",
                    "returnable":false,
                    "price":49
                 }
              ]
           }
        }
     ]
  }
}
```


https://gist.github.com/hjr3/2289546

Aggregate-oriented NoSQL Databases

Pros

- Provides clearer semantics to consider by focusing on the aggregate unit used by applications.
- Better design choice for running on a cluster.

Cons

- Drawing boundaries of an aggregate is not easy.
- When a goal of data management/analysis is not clear, aggregate models might not be the best choice.
- Doesn’t support ACID transactions.

Aggregate-oriented NoSQL Databases

Types

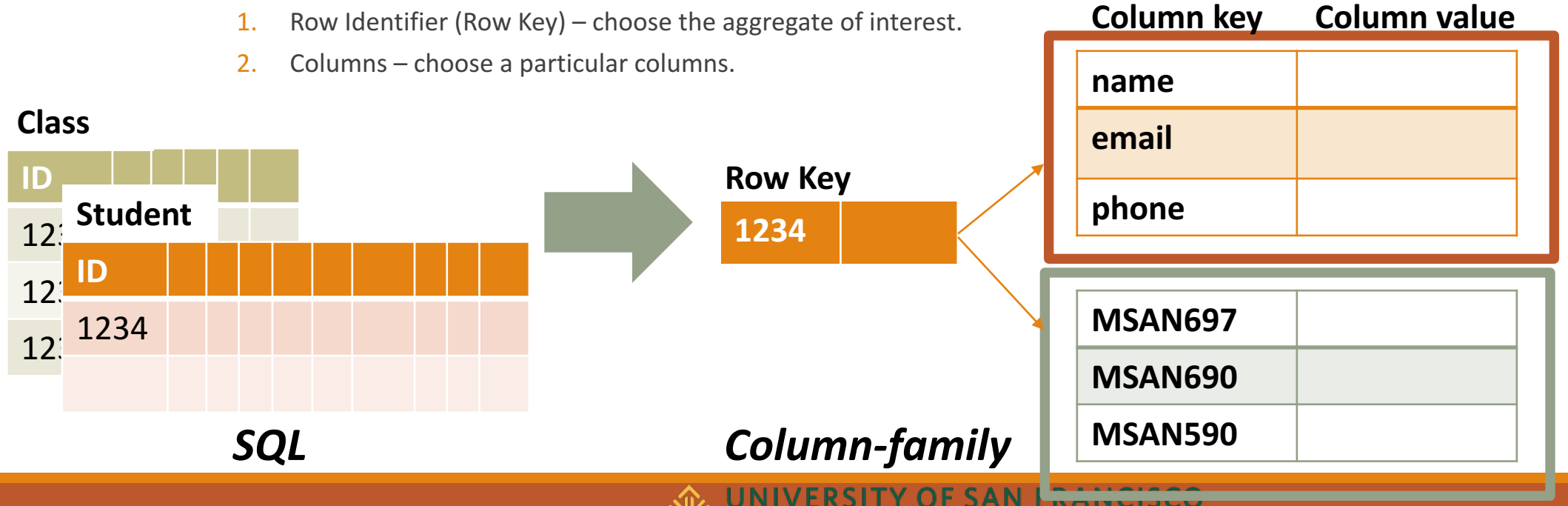
- Key-value and Document Database
  - Each aggregate has a key (ID).
  - Key-value database
    - We can store whatever we want in aggregates.
    - Key lookup for the entire aggregate.
  - Document database
    - It has allowable structures and types.
    - Access by key and also by the fields in the aggregates. (can retrieve fields in the value.)

➔Key-value and Document DB are similar, and their distinction is often blurry.
But with Document DB, you can submit a query based on the internal structure of the document.

Aggregate-oriented NoSQL Databases

Types

- Column Family
  - Optimize for cases when write is rare, but columns are read together in many rows together .
  - Organizes columns into column families (Unit of access).
  - Two level map structure.
    1. Row Identifier (Row Key) – choose the aggregate of interest.
    2. Columns – choose a particular columns.



Relationship-oriented NoSQL Database

Needs of relationship-oriented DB

- Relational database with complex schema
  - Hard to understand, query, generalize and integrate data.
- Aggregate-oriented NoSQL
  - Atomicity is only supported within a single aggregate.

Relationship-oriented NoSQL Database

Graph Database

- Nodes (Object) and edges (Relationships) representation.
- For data with complex relationships.
  - Focuses on graph traverse (more than insert.).
  - Cf) RDBMS : Many joins can cause poor performance.
- Running on a single server rather than distributed across clusters.

NoSQL Database Examples

Key-value

- Redis, Riak, Berkeley DB, etc.

Document

- MongoDB, CouchDB, OrientDB, RavenDB, etc.

Column Family

- Cassandra, Hbase, Amazon SimpleDB, etc.

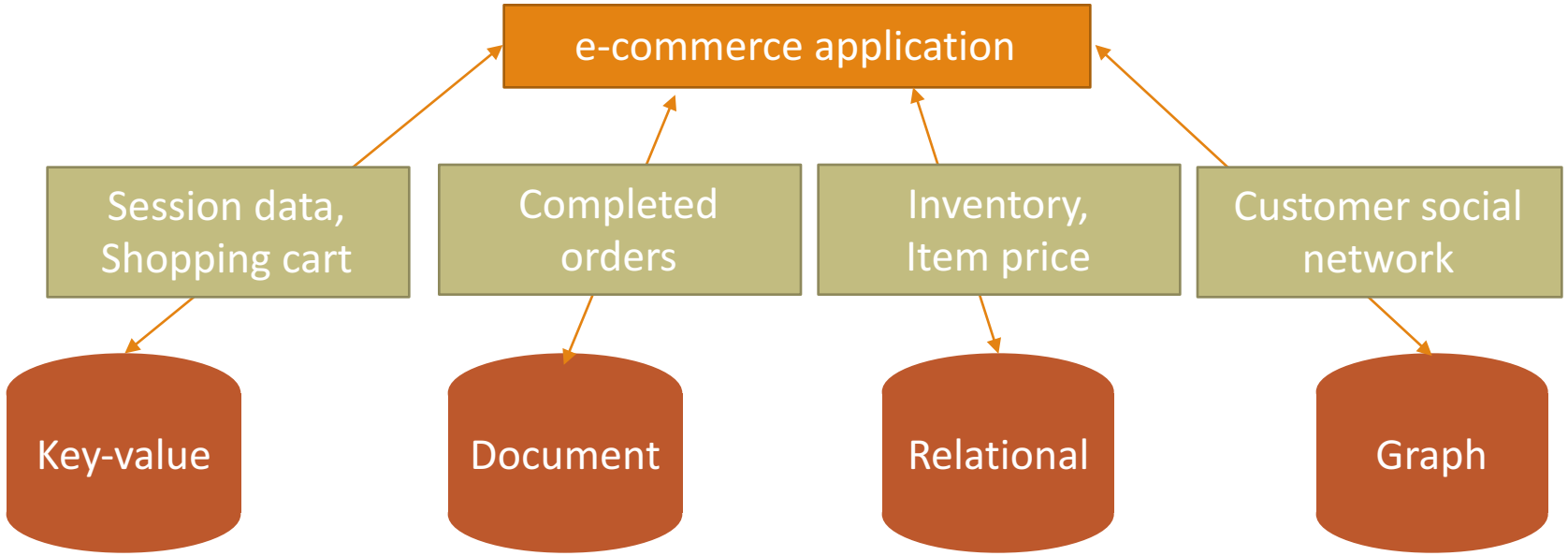
Graph

- OrientDB, Neo4J, FlockDB, etc.

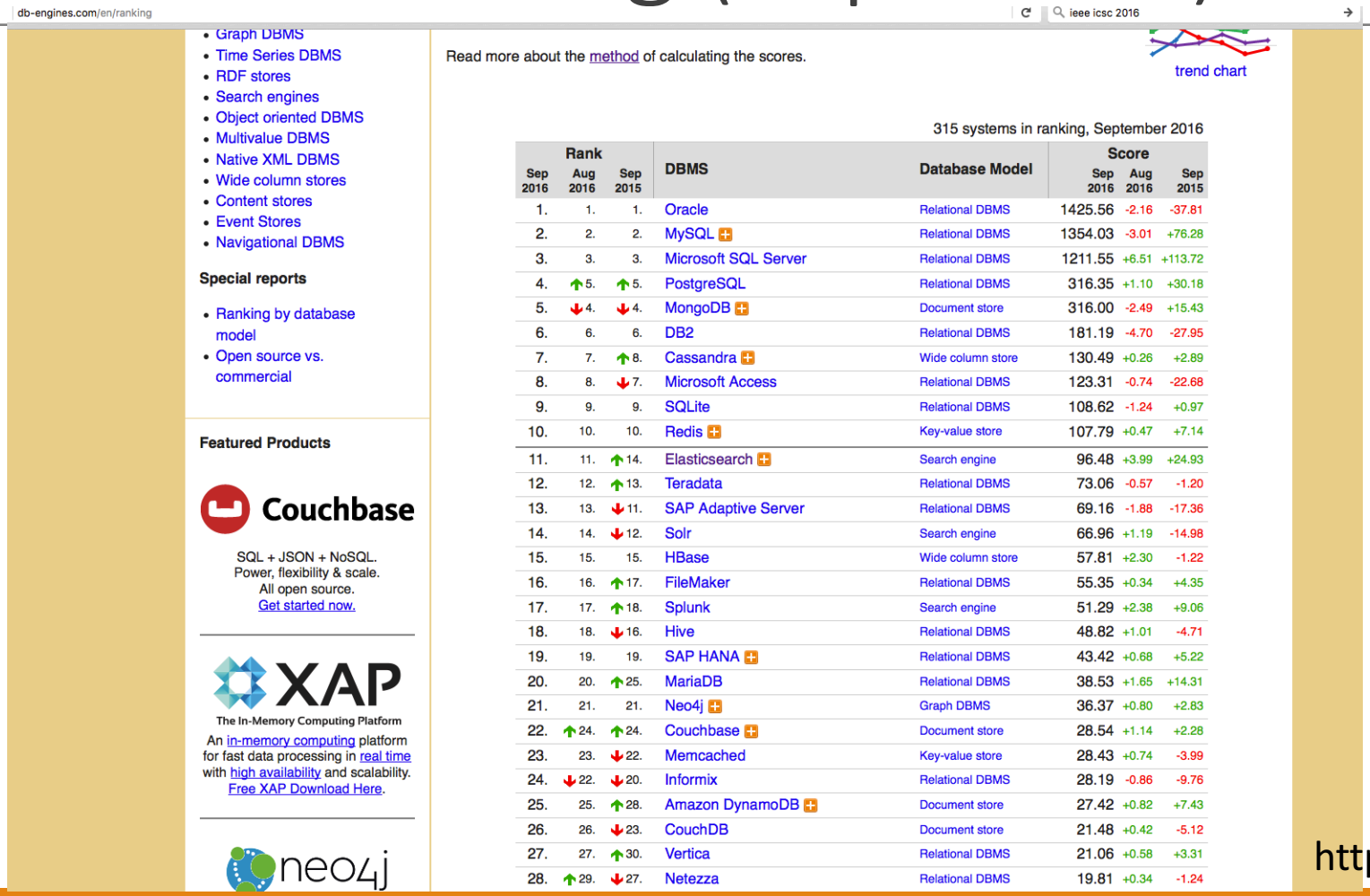
Choice of DBMS

Polygot Persistence

- Using multiple data storage technologies, chosen based on the way data is being used by individual applications.
- NoSQL data stores do not replace relational databases.



Database Ranking (Sept 2016)



db-engines.com/en/ranking

ieee icsc 2016

- Graph DBMS
- Time Series DBMS
- RDF stores
- Search engines
- Object oriented DBMS
- Multivalue DBMS
- Native XML DBMS
- Wide column stores
- Content stores
- Event Stores
- Navigational DBMS

Special reports

- Ranking by database model
- Open source vs. commercial

Featured Products

Couchbase

SQL + JSON + NoSQL.
Power, flexibility & scale.
All open source.
Get started now.

XAP
The In-Memory Computing Platform
An in-memory computing platform for fast data processing in real time with high availability and scalability.
Free XAP Download Here.

neo4j

Read more about the method of calculating the scores.

trend chart

315 systems in ranking, September 2016

Rank Sep 2016	Rank Aug 2016	Rank Sep 2015	DBMS	Database Model	Score Sep 2016	Score Aug 2016	Score Sep 2015
1.	1.	1.	Oracle	Relational DBMS	1425.56	-2.16	-37.81
2.	2.	2.	MySQL	Relational DBMS	1354.03	-3.01	+76.28
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1211.55	+6.51	+113.72
4.	↑5.	↑5.	PostgreSQL	Relational DBMS	316.35	+1.10	+30.18
5.	↓4.	↓4.	MongoDB	Document store	316.00	-2.49	+15.43
6.	6.	6.	DB2	Relational DBMS	181.19	-4.70	-27.95
7.	7.	↑8.	Cassandra	Wide column store	130.49	+0.26	+2.89
8.	8.	↓7.	Microsoft Access	Relational DBMS	123.31	-0.74	-22.68
9.	9.	9.	SQLite	Relational DBMS	108.62	-1.24	+0.97
10.	10.	10.	Redis	Key-value store	107.79	+0.47	+7.14
11.	11.	↑14.	Elasticsearch	Search engine	96.48	+3.99	+24.93
12.	12.	↑13.	Teradata	Relational DBMS	73.06	-0.57	-1.20
13.	13.	↓11.	SAP Adaptive Server	Relational DBMS	69.16	-1.88	-17.36
14.	14.	↓12.	Solr	Search engine	66.96	+1.19	-14.98
15.	15.	15.	HBase	Wide column store	57.81	+2.30	-1.22
16.	16.	↑17.	FileMaker	Relational DBMS	55.35	+0.34	+4.35
17.	17.	↑18.	Splunk	Search engine	51.29	+2.38	+9.06
18.	18.	↓16.	Hive	Relational DBMS	48.82	+1.01	-4.71
19.	19.	19.	SAP HANA	Relational DBMS	43.42	+0.68	+5.22
20.	20.	↑25.	MariaDB	Relational DBMS	38.53	+1.65	+14.31
21.	21.	21.	Neo4j	Graph DBMS	36.37	+0.80	+2.83
22.	↑24.	↑24.	Couchbase	Document store	28.54	+1.14	+2.28
23.	23.	↓22.	Memcached	Key-value store	28.43	+0.74	-3.99
24.	↓22.	↓20.	Informix	Relational DBMS	28.19	-0.86	-9.76
25.	25.	↑28.	Amazon DynamoDB	Document store	27.42	+0.82	+7.43
26.	26.	↓23.	CouchDB	Document store	21.48	+0.42	-5.12
27.	27.	↑30.	Vertica	Relational DBMS	21.06	+0.58	+3.31
28.	↑29.	↓27.	Netezza	Relational DBMS	19.81	+0.34	-1.24

http://db-engines.com/en/ranking/

Database Ranking (Sept 2016)

315 systems in ranking, September 2016

Rank			DBMS	Database Model	Score		
Sep 2016	Aug 2016	Sep 2015			Sep 2016	Aug 2016	Sep 2015
1.	1.	1.	Oracle	Relational DBMS	1425.56	-2.16	-37.81
2.	2.	2.	MySQL	Relational DBMS	1354.03	-3.01	+76.28
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1211.55	+6.51	+113.72
4.	↑ 5.	↑ 5.	PostgreSQL	Relational DBMS	316.35	+1.10	+30.18
5.	↓ 4.	↓ 4.	MongoDB	Document store	316.00	-2.49	+15.43
6.	6.	6.	DB2	Relational DBMS	181.19	-4.70	-27.95
7.	7.	↑ 8.	Cassandra	Wide column store	130.49	+0.26	+2.89
8.	8.	↓ 7.	Microsoft Access	Relational DBMS	123.31	-0.74	-22.68
9.	9.	9.	SQLite	Relational DBMS	108.62	-1.24	+0.97
10.	10.	10.	Redis	Key-value store	107.79	+0.47	+7.14

References

Sadalage, Pramod J., and Martin Fowler. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education, 2012.

Redmond, Eric, and Jim R. Wilson. *Seven databases in seven weeks: a guide to modern databases and the NoSQL movement*. Pragmatic Bookshelf, 2012.

Date, C. J. "Database systems." *Vols. I & II, Narosa Pub* (1986).

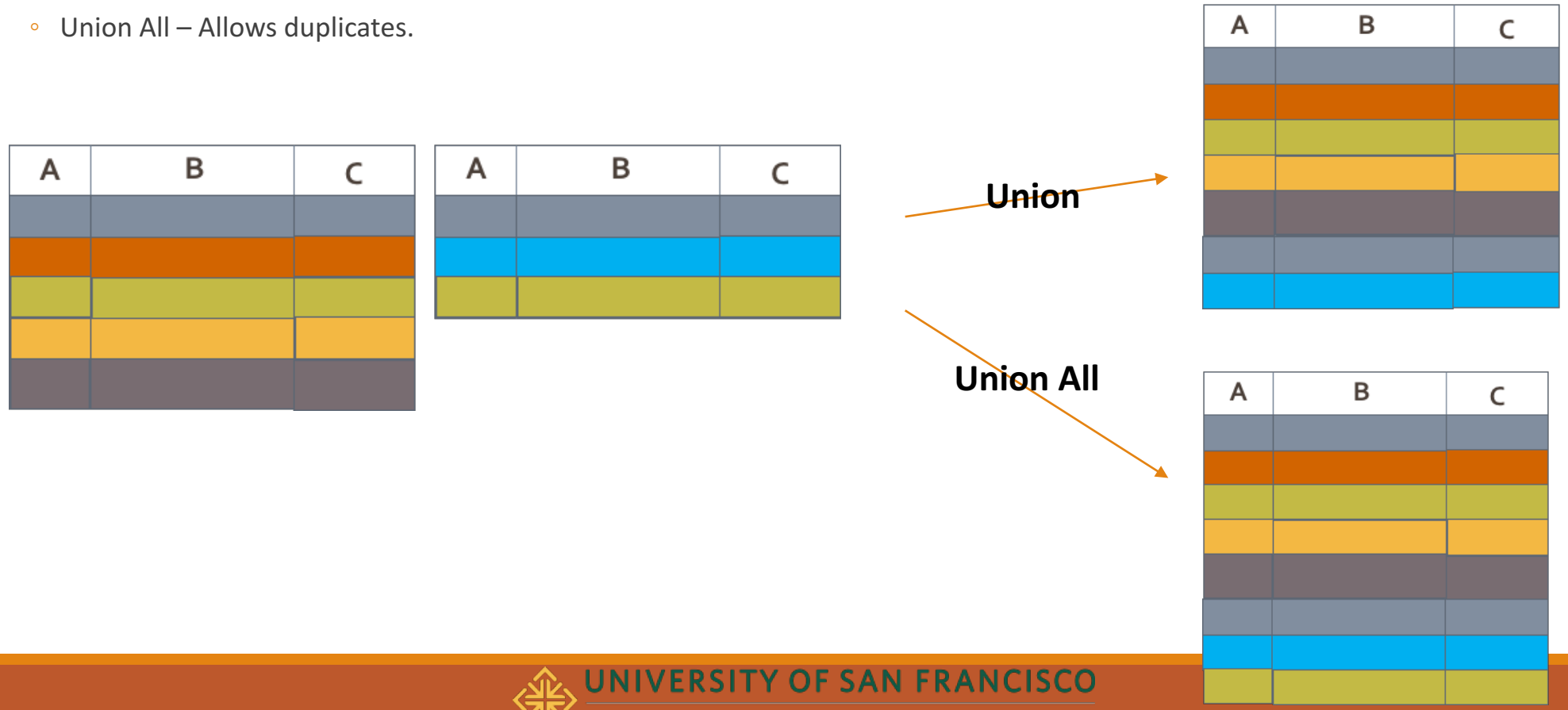
SQL Interview Questions

Important RDBMS concepts to remember

- Create
- Insert
- Select
- Update
- Join (Inner, Outer, Left, Right, etc.)*
- Union All/Union/ Union Distinct
- Minus
- Intersect
- Difference WHERE vs. Having
- Normalization (1NF, 2NF and 3NF)*
- Transaction(Concurrency Control) – ACID*
- Indexing* - R tree, Hash
- Truncate vs Delete
- Execution Plan
- Rank, Dense_rank, Rownum

Relational Databases

- Union : Combine result sets from 2 or more SELECT queries.
  - Union – Returns distinct values by default.
  - Union All – Allows duplicates.



Relational Databases

- Delete vs. Truncate

Truncate	Delete
We can't Rollback after performing Truncate. **Example:** BEGIN TRAN TRUNCATE TABLE tranTest SELECT * FROM tranTest ROLLBACK SELECT * FROM tranTest	We can Rollback after delete. **Example:** BEGIN TRAN DELETE FROM tranTest SELECT * FROM tranTest ROLLBACK SELECT * FROM tranTest
Truncate reset identity of table.	Truncate reset identity of table.
It locks the entire table.	It locks the table row.
Its DDL(Data Definition Language) command.	Its DML(Data Manipulation Language) command.
We can't use WHERE clause with it.	We can use WHERE to filter data to delete.
Trigger is not fired while truncate.	Trigger is fired.
Syntax : **1)** TRUNCATE TABLE table_name	**Syntax :** **1)** DELETE FROM table_name **2)** DELETE FROM table_name WHERE example_column_id IN (1,2,3)

http://stackoverflow.com/questions/139630/whats-the-difference-between-truncate-and-delete-in-sql

Relational Databases

WHERE vs. HAVING

- WHERE : Conditions on *individual rows.*
- HAVING : Conditions on *aggregations.*
  - i.e. results of a single result produced from *multiple* rows using aggregate functions such as count, average, min, max, sum, etc .
- As a rule of thumb, use WHERE before GROUP BY and HAVING after GROUP BY.