Learning Management System: MS3

Rabeet Fatmi, Rakesh Meka, Daniel Teichman (Lead)

Advanced Database Development

Spring 2016

Contents

[Project Description 3](#_Toc445735554)

[Problem Statement 3](#_Toc445735555)

[Expected Outcome 3](#_Toc445735556)

[Schema 3](#_Toc445735557)

[Code 3](#_Toc445735558)

# Project Description

We will demonstrate a distributed database architecture for a scalable Learning Management System (LMS) using a relational and document-store or key-value clusters. The distributed database architecture will employ clustering and replication techniques to demonstrate safeguarding of data located in geographically disparate data centers. We will proceed by gathering LMS requirements, creating a high-level design of an application that meets those requirements, designing a database to support the application, and then creating application code to demonstrate some features of an ideal LMS application.

# Problem Statement

As of now, Blackboard and Canvas are the two major competing LMS applications for universities. Both products are commonly hosted by the vendor and these products are costly and not within the institution's direct control. We seek to reduce the cost of using a high-quality LMS by creating a free open-source (FOSS) LMS application that can operate with organizationally-controlled dedicated server hardware while maintaining safekeeping of documents in a distributed database with geographically disperse cloud servers. We seek to reduce the per-student cost of providing LMS services to universities globally with the hope that universities will benefit from a scalable FOSS LMS as much as other organizations have benefited from other FOSS projects such as Linux, a preeminent server operating system.

# Expected Outcome

We will deliver a distributed database design and architecture to support an LMS. Our design will feature replication to guarantee security of the data in the event of a hardware failure of a dedicated application server. The software application, in the scope of this project, will contain a set of features to allow for user management tasks (add/edit/remove users), course management tasks (grading and add/edit/remove assignments and course materials), and assignment management tasks (create assignment submissions). The scope of features will be a minimum to demonstrate a suitable database architecture in support of a useful LMS application. Finally, scalability testing will consist of a benchmark to simulate a large number of students simultaneously downloading course materials and posting assignments to determine that the software is ready to serve the needs of a large university.

# Schema

# C:\Users\Daniel\Downloads\prj.png

# Code (for PostgreSQL):

CREATE schema audit;

REVOKE CREATE ON schema audit FROM public;

CREATE TABLE audit.logged\_actions (

schema\_name text NOT NULL,

TABLE\_NAME text NOT NULL,

user\_name text,

action\_tstamp TIMESTAMP WITH TIME zone NOT NULL DEFAULT CURRENT\_TIMESTAMP,

action TEXT NOT NULL CHECK (action IN ('I','D','U')),

original\_data text,

new\_data text,

query text

) WITH (fillfactor=100);

REVOKE ALL ON audit.logged\_actions FROM public;

GRANT SELECT ON audit.logged\_actions TO public;

CREATE INDEX logged\_actions\_schema\_table\_idx

ON audit.logged\_actions(((schema\_name||'.'||TABLE\_NAME)::TEXT));

CREATE INDEX logged\_actions\_action\_tstamp\_idx

ON audit.logged\_actions(action\_tstamp);

CREATE INDEX logged\_actions\_action\_idx

ON audit.logged\_actions(action);

CREATE OR REPLACE FUNCTION audit.if\_modified\_func() RETURNS TRIGGER AS $body$

DECLARE

v\_old\_data TEXT;

v\_new\_data TEXT;

BEGIN

IF (TG\_OP = 'UPDATE') THEN

v\_old\_data := ROW(OLD.\*);

v\_new\_data := ROW(NEW.\*);

INSERT INTO audit.logged\_actions (schema\_name,table\_name,user\_name,action,original\_data,new\_data,query)

VALUES (TG\_TABLE\_SCHEMA::TEXT,TG\_TABLE\_NAME::TEXT,session\_user::TEXT,substring(TG\_OP,1,1),v\_old\_data,v\_new\_data, current\_query());

RETURN NEW;

ELSIF (TG\_OP = 'DELETE') THEN

v\_old\_data := ROW(OLD.\*);

INSERT INTO audit.logged\_actions (schema\_name,table\_name,user\_name,action,original\_data,query)

VALUES (TG\_TABLE\_SCHEMA::TEXT,TG\_TABLE\_NAME::TEXT,session\_user::TEXT,substring(TG\_OP,1,1),v\_old\_data, current\_query());

RETURN OLD;

ELSIF (TG\_OP = 'INSERT') THEN

v\_new\_data := ROW(NEW.\*);

INSERT INTO audit.logged\_actions (schema\_name,table\_name,user\_name,action,new\_data,query)

VALUES (TG\_TABLE\_SCHEMA::TEXT,TG\_TABLE\_NAME::TEXT,session\_user::TEXT,substring(TG\_OP,1,1),v\_new\_data, current\_query());

RETURN NEW;

ELSE

RAISE WARNING '[AUDIT.IF\_MODIFIED\_FUNC] - Other action occurred: %, at %',TG\_OP,now();

RETURN NULL;

END IF;

EXCEPTION

WHEN data\_exception THEN

RAISE WARNING '[AUDIT.IF\_MODIFIED\_FUNC] - UDF ERROR [DATA EXCEPTION] - SQLSTATE: %, SQLERRM: %',SQLSTATE,SQLERRM;

RETURN NULL;

WHEN unique\_violation THEN

RAISE WARNING '[AUDIT.IF\_MODIFIED\_FUNC] - UDF ERROR [UNIQUE] - SQLSTATE: %, SQLERRM: %',SQLSTATE,SQLERRM;

RETURN NULL;

WHEN OTHERS THEN

RAISE WARNING '[AUDIT.IF\_MODIFIED\_FUNC] - UDF ERROR [OTHER] - SQLSTATE: %, SQLERRM: %',SQLSTATE,SQLERRM;

RETURN NULL;

END;

$body$

LANGUAGE plpgsql

SECURITY DEFINER

SET search\_path = pg\_catalog, audit;

-- DB

create schema db;

create table db.Role (rolename varchar PRIMARY KEY);

create table db.Users (username varchar PRIMARY KEY, password varchar NOT NULL, email varchar);

create table db.Semester (year integer, term varchar, PRIMARY KEY (year, term));

create table db.course (courseid integer PRIMARY KEY, coursename text, section text, semester\_year integer, semester\_term varchar, FOREIGN KEY (semester\_year, semester\_term) references db.semester (year, term));

create table db.course\_memberships (courseid integer references course (courseid), username varchar references db.users (username), rolename varchar references db.role (rolename), PRIMARY KEY (courseid, username));

create table db.course\_material (materialid integer PRIMARY KEY, courseid integer references db.course (courseid), path text, filename text, username varchar references db.users (username));

create table db.assignment (assignmentid integer PRIMARY KEY, courseid integer references db.course (courseid), username varchar references db.users (username), assignmentname text, assignmenttext text);

create table db.submission (submissionid integer PRIMARY KEY, assignmentid integer references db.assignment (assignmentid), username varchar references db.users (username), timestamp timestamp DEFAULT current\_timestamp);

CREATE INDEX course\_id ON db.Assignment USING btree (courseid);

CREATE INDEX assignment\_id ON db.Submission USING btree (assignmentid);

-- Triggers

CREATE TRIGGER role\_audit AFTER INSERT OR UPDATE OR DELETE ON db.role FOR EACH ROW EXECUTE PROCEDURE audit.if\_modified\_func();

CREATE TRIGGER users\_audit AFTER INSERT OR UPDATE OR DELETE ON db.users FOR EACH ROW EXECUTE PROCEDURE audit.if\_modified\_func();

CREATE TRIGGER semester\_audit AFTER INSERT OR UPDATE OR DELETE ON db.semester FOR EACH ROW EXECUTE PROCEDURE audit.if\_modified\_func();

CREATE TRIGGER course\_audit AFTER INSERT OR UPDATE OR DELETE ON db.course FOR EACH ROW EXECUTE PROCEDURE audit.if\_modified\_func();

CREATE TRIGGER course\_memberships\_audit AFTER INSERT OR UPDATE OR DELETE ON db.course\_memberships FOR EACH ROW EXECUTE PROCEDURE audit.if\_modified\_func();

CREATE TRIGGER course\_material\_audit AFTER INSERT OR UPDATE OR DELETE ON db.course\_material FOR EACH ROW EXECUTE PROCEDURE audit.if\_modified\_func();

CREATE TRIGGER assignment\_audit AFTER INSERT OR UPDATE OR DELETE ON db.assignment FOR EACH ROW EXECUTE PROCEDURE audit.if\_modified\_func();

CREATE TRIGGER submission\_audit AFTER INSERT OR UPDATE OR DELETE ON db.submission FOR EACH ROW EXECUTE PROCEDURE audit.if\_modified\_func();