Lecture 11

Chapter 9: SQL in a Server Environment

John Connor

April 30, 2018

Example web-application architecture.

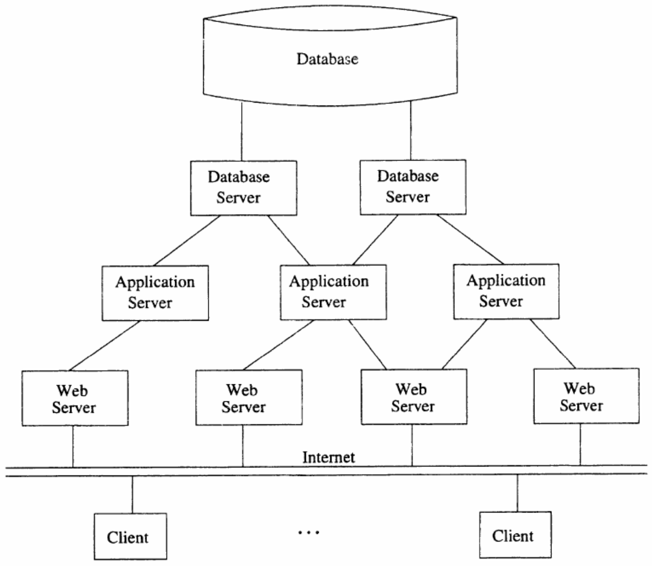

Figure 9.1: The Three-Tier Architecture

Schemas Revisited

Real database schemas have additional information associated with them. For example

1. Character Sets.

Schemas Revisited

Real database schemas have additional information associated with them. For example

1. Character Sets. How to interpret the contents of varchar, char, etc.

Schemas Revisited

Real database schemas have additional information associated with them. For example

1. Character Sets. How to interpret the contents of varchar, char, etc. You almost certainly want to use UTF-8, but many databases use latin1 by default.
2. Collations.

Schemas Revisited

Real database schemas have additional information associated with them. For example

1. Character Sets. How to interpret the contents of varchar, char, etc. You almost certainly want to use UTF-8, but many databases use latin1 by default.
2. Collations. What values do the relations $<, >, =$ take on when given string and character arguments?
3. Privileges.

Schemas Revisited

Real database schemas have additional information associated with them. For example

1. Character Sets. How to interpret the contents of varchar, char, etc. You almost certainly want to use UTF-8, but many databases use latin1 by default.
2. Collations. What values do the relations $<, >, =$ take on when given string and character arguments?
3. Privileges. Information about what operations users can perform on which objects.
4. Stored Procedures.

Schemas Revisited

Real database schemas have additional information associated with them. For example

1. Character Sets. How to interpret the contents of varchar, char, etc. You almost certainly want to use UTF-8, but many databases use latin1 by default.
2. Collations. What values do the relations $<, >, =$ take on when given string and character arguments?
3. Privileges. Information about what operations users can perform on which objects.
4. Stored Procedures. Executable code.

Character Sets and Collations

In mysql the current default character set and collation are stored in “server-level” variables.

```
USE db_name;
SELECT @@character_set_database, @@collation_database
```

Character Sets and Collations

In mysql the current default character set and collation are stored in “server-level” variables.

```
USE db_name;
SELECT @@character_set_database, @@collation_database
```

When creating a table, you have the option of using a different character set and collation.

```
CREATE TABLE t1(
    (columns ...)
) DEFAULT CHARACTER SET utf8 COLLATE utf8_bin;
```

Privileges

In mysql, a user can be created with the following command

```
CREATE USER user_name@system_name
  IDENTIFIED BY 'password';
```

Privileges

In mysql, a user can be created with the following command

```
CREATE USER user_name@system_name
  IDENTIFIED BY 'password';
```

Beware! The system name “localhost” is special! In most cases it is not equivalent to “127.0.0.1”, but rather it refers to connecting to the service through a Unix socket.

Privileges

In mysql, a user can be created with the following command

```
CREATE USER user_name@system_name
 IDENTIFIED BY 'password';
```

Beware! The system name "localhost" is special! In most cases it is not equivalent to "127.0.0.1", but rather it refers to connecting to the service through a Unix socket.

Permissions are granted to a user by commands similar to

```
GRANT type_of_permission
   ON database_name.table_name
   TO 'user_name'@'system_name';
```

Where type_of_permission is one of `ALL PRIVILEGES`, `CREATE`, `DROP`, `DELETE`, `INSERT`, `SELECT`, `UPDATE`, `GRANT OPTION`

Privileges

And of course a user can be removed

```
DROP USER 'user_name'@'system_name';
```

Privileges

And of course a user can be removed

```
DROP USER 'user_name'@'system_name';
```

The Current permissions can be viewed

Privileges

And of course a user can be removed

```
DROP USER 'user_name'@'system_name';
```

The Current permissions can be viewed

```
SHOW GRANTS user_name;
```

Privileges

And of course a user can be removed

```
DROP USER 'user_name'@'system_name';
```

The Current permissions can be viewed

```
SHOW GRANTS user_name;
```

And revoked

```
REVOKE type_of_permission
    ON database_name.table_name
  FROM 'user_name'@'system_name'
```

Defining Stored Procedures and Functions

```
CREATE PROCEDURE <name> (<parameters>)
    <local declarations>
    <procedure body>;

CREATE FUNCTION <name> (<parameters>) RETURNS <type>
    <local declarations>
    <function body>;
```

Defining Stored Procedures and Functions

```
CREATE PROCEDURE <name> (<parameters>)
    <local declarations>
    <procedure body>;
```

```
CREATE FUNCTION <name> (<parameters>) RETURNS <type>
    <local declarations>
    <function body>;
```

To call a stored procedure, use the `CALL` statement.

```
CALL <name> (<argument list>);
```

Defining Stored Procedures and Functions

```
CREATE PROCEDURE <name> (<parameters>)
    <local declarations>
    <procedure body>;
```

```
CREATE FUNCTION <name> (<parameters>) RETURNS <type>
    <local declarations>
    <function body>;
```

To call a stored procedure, use the `CALL` statement.

```
CALL <name> (<argument list>);
```

To invoke a user defined function `foo`, you use the c-like syntax `foo(1,2,3)` wherever an expression is allowed.

```
SELECT foo(R.A, R.B, R.C), R.D
  FROM R
 WHERE bar(R.A) < bar(R.B);
```

Example Stored Procedure

There are many special SQL statements that can placed in function and stored procedure bodies. For more details, see §9.4.

```
CREATE PROCEDURE Move (
      IN oldAddr VARCHAR(255),
      IN newAddr VARCHAR(255)
)
UPDATE MovieStar
    SET address = newAddr
 WHERE address = oldAddr;
```

Example Stored Procedure

There are many special SQL statements that can placed in function and stored procedure bodies. For more details, see §9.4.

```
CREATE PROCEDURE Move (
      IN oldAddr VARCHAR(255),
      IN newAddr VARCHAR(255)
)
UPDATE MovieStar
   SET address = newAddr
 WHERE address = oldAddr;

CALL Move('123␣Foo␣Lane', '456␣Bar␣Drive');
```

Example Stored Procedure

There are many special SQL statements that can placed in function and stored procedure bodies. For more details, see §9.4.

```
CREATE PROCEDURE Move (
    IN oldAddr VARCHAR(255),
    IN newAddr VARCHAR(255)
)
UPDATE MovieStar
   SET address = newAddr
 WHERE address = oldAddr;

CALL Move('123␣Foo␣Lane', '456␣Bar␣Drive');
```

Pop Quiz! Why is this stored procedure (probably) a horrible idea?

Example Function (Figure 9.13)

This function takes as arguments a studio name and a year, and returns true if and only if the named studio produced at least one comedy in the given year, or if the named studio produced no movies at all during the given year.

Example Function (Figure 9.13)

This function takes as arguments a studio name and a year, and returns true if and only if the named studio produced at least one comedy in the given year, or if the named studio produced no movies at all during the given year.

```
CREATE FUNCTION BandW(s CHAR(15), y INT) RETURNS BOOLEAN
IF NOT EXISTS (
    SELECT * FROM Moves
     WHERE year = y AND studioName = s)
THEN RETURN TRUE;
ELSEIF 1 <= (
    SELECT COUNT(*)
      FROM Movies
     WHERE year = y AND studioName = s
                AND genre = 'comedy')
THEN RETURN TRUE;
ELSE RETURN FALSE;
END IF;
```

Statement Details

Local variables can be declared

```
DECLARE <name> <type>;
```

Statement Details

Local variables can be declared

DECLARE $<\mathtt{name}>$ $<\mathtt{type}>$;

and assigned values

SET $<\mathtt{name}> = <\mathtt{expression}>$;

where `expression` can be a literal, a function, or a query which returns a scaler.

Statement Details

Local variables can be declared

```
DECLARE <name> <type >;
```

and assigned values

```
SET <name> = <expression >;
```

where `expression` can be a literal, a function, or a query which returns a scaler.
Branching statements have the form

```
IF <condition > THEN
  <statements>
ELSE IF <condition > THEN
  <statements>
ELSE IF <condition > THEN
  ...
ELSE
  <statements>
END IF ;
```

Statement Details

Loops can be declared

```
<label>: LOOP
  <statements>
END LOOP;
```

Statement Details

Loops can be declared

```
<label>: LOOP
  <statements>
END LOOP;
```

to escape a loop,

```
<label>: LOOP
  ...
  IF <condition> THEN
    LEAVE <label>;
  END IF;
  ...
END LOOP;
```

These other loop statements are also supported

```
WHILE <condition> DO
  <statements>
END WHILE;

REPEAT
  <statements>
UNTIL <condition>
END REPEAT;
```

Cursors

The last type of loop is a for-loop, which uses a *cursor* to access the results of a query.

```
FOR <loop name> AS <cursor name> CURSOR FOR
    <query>
DO
    <statements>
END IF;
```

Cursor Details

This is a procedure (Figure 9.15) which uses a cursor to compute the mean and variance of movie lengths by studio.

Cursor Details

```
CREATE PROCEDURE MeanVar(
  IN s CHAR(15),
  OUT mean REAL,
  OUT variance REAL
)
DECLARE movieCount INTEGER;
  SELECT length FROM Movies WHERE studioName = s;
BEGIN
  SET mean = 0.0;
  SET variance = 0.0;
  SET movieCount = 0;
    FOR movieLoop AS MovieCursor CURSOR FOR
      SELECT length FROM Movies WHERE studioName = 's';
    DO
      SET movieCount = movieCount + 1;
      SET mean = mean + 1;
      SET variance = variance + length * length;
    END FOR;
    SET mean = mean / movieCount;
    SET variance = variance / movieCount - mean * mean;
END;
```