پوهنتون کابل

پوهنځی کمپیوترساینس

دیپارتمنت سیستم های معلوماتی

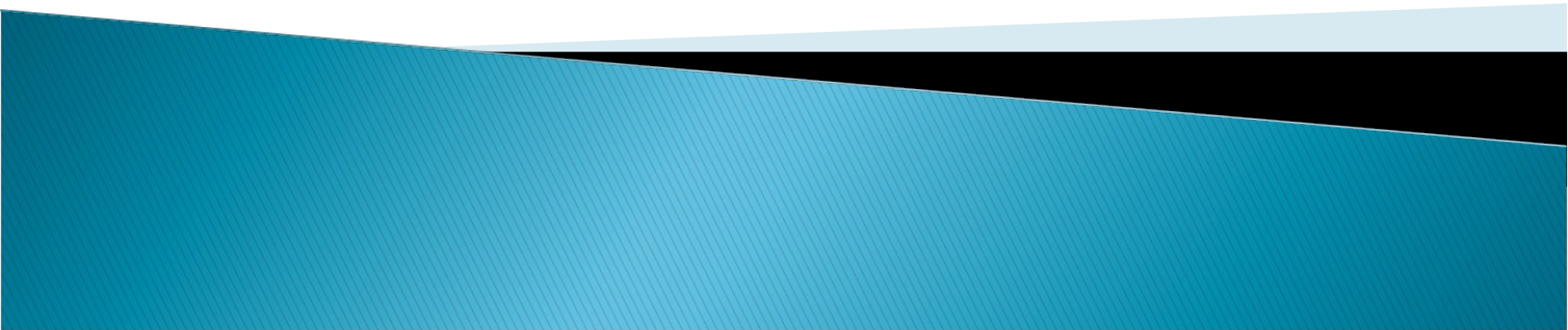
Structured Query Language (SQL) Fundamentals

Lectures 14-15

تهیه کننده : پوهنیار محمد شعیب "زرین خیل"
سال : 1389

Structured Query Language (SQL) 14 + Lab 01

By: M Shuaib Zarinkhail 2010



MySQL Transactional and Locking Statements

- MySQL supports Local Transactions within a given client session
- Local Transactions can set through statements such as
  1. SET autocommit …
  2. START TRANSACTION or BEGIN [WORK]
     - COMMIT
     - ROLLBACK

START TRANSACTION … COMMIT, ROLLBACK – Syntax

- START TRANSACTION *or* BEGIN

  run your commands

- COMMIT *(accept changes)*

  or

- ROLLBACK *(reject changes)*

- The START TRANSACTION or BEGIN statement starts a new transaction
- COMMIT commits the current transaction
- ROLLBACK rolls back the current transaction

START TRANSACTION … COMMIT, ROLLBACK – Autocommit

- SET autocommit = {0 | 1}
- The SET autocommit statement disables or enables the default autocommit mode for the current session
- By default, MySQL runs with autocommit mode enabled

START TRANSACTION … COMMIT, ROLLBACK

- autocommit enabled mode means that as soon as you execute a statement that updates (modifies) a table, MySQL stores the update on disk to make it permanent
- To disable autocommit mode, use the following statement:
  - SET autocommit = 0;

START TRANSACTION … COMMIT, ROLLBACK

- To disable autocommit mode for a single series of statements, use the START TRANSACTION statement:
  - START TRANSACTION;
  - SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
  - UPDATE table2 SET summary=@A WHERE type=1;
  - COMMIT;

START TRANSACTION … COMMIT, ROLLBACK

- With START TRANSACTION, autocommit remains disabled until you end the transaction with COMMIT or ROLLBACK
- The autocommit mode then reverts to its previous state
- START TRANSACTION was added in MySQL 4.0.11

START TRANSACTION … COMMIT, ROLLBACK

- BEGIN and BEGIN WORK are supported as aliases of START TRANSACTION for initiating a transaction
- This is standard SQL syntax and is the recommended way to start an ad-hoc transaction
- BEGIN and BEGIN WORK are available from MySQL 3.23.17 and 3.23.19, respectively

Lab 01 – Movies Database

In this lab you have to:

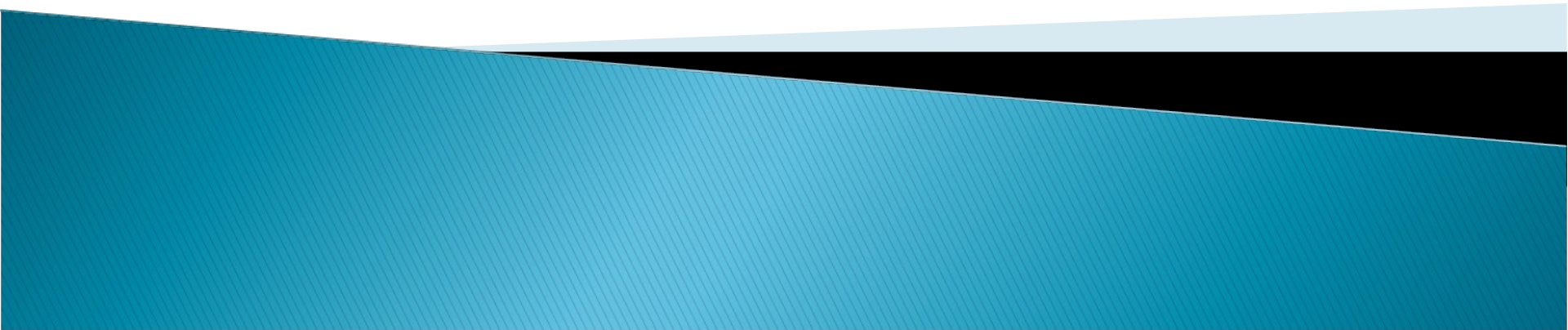
- Create a database in MySQL Server
- Create tables in that database
- Do data entry to the tables

At the end of the lab hour:

- Record your answers and turn them to the lab instructor
- Keep the database for future labs (lab02 and lab03)

Structured Query Language (SQL) 15

By: M Shuaib Zarinkhail 2010



Statement that can not be Rolled Back

- When using TRANSACTIONs, some statements cannot be rolled back
- In general, they include data definition language (DDL) statements, such as
  - CREATE / DROP DATABASEs
  - CREATE / DROP TABLEs
  - ALTER TABLEs
- TRUNCATE TABLE also could no be rolled back

Statements that can not be Rolled Back

- You should design your transactions not to include such statements
- If you issue a statement early in a transaction that cannot be rolled back, and then another statement later fails
  - By ROLLBACK, the full effect of the transaction cannot be rolled back

SAVEPOINT and ROLLBACK TO SAVEPOINT

- You can split a transaction in groups
- Each group can be named
- This is called savepoint
- Savepoints are set by users in sessions
- ROLLBACK TO SAVEPOINT rolls back only commands implemented after the named savepoint

SAVEPOINT and ROLLBACK TO SAVEPOINT – Syntax

- SAVEPOINT identifier
- ROLLBACK TO SAVEPOINT identifier
- RELEASE SAVEPOINT identifier

- Starting from MySQL 4.0.14 and 4.1.1, InnoDB supports the SQL statements SAVEPOINT and ROLLBACK TO SAVEPOINT

SAVEPOINT and ROLLBACK TO SAVEPOINT – Syntax

- The SAVEPOINT statement sets a named transaction savepoint with a name of identifier
- If the current transaction has a savepoint with the same name, the old savepoint is deleted and the new one is set instead

SAVEPOINT and ROLLBACK TO SAVEPOINT

- The ROLLBACK TO SAVEPOINT statement rolls back a transaction to the named savepoint
- The TRANSACTION does not terminate with the" rollback to savepoint" command
- Modifications that the current transaction made to rows after the savepoint was set are undone in the rollback

SAVEPOINT and ROLLBACK TO SAVEPOINT – Syntax

- If the ROLLBACK TO SAVEPOINT statement returns the following error, it means that no savepoint with the specified name exists:
  - ERROR 1181: Got error 153 during ROLLBACK
- All savepoints of the current transaction are deleted if you execute a COMMIT, or a ROLLBACK that does not name a savepoint

Lock and Unlock Tables

- LOCK TABLES command locks tables according to limitations set by users
  - e.g. read, write
- All tables in a DB are locked except those are declared in the command line with the mentioned rights
- You can stop locking tables by releasing the UNLOCK TABLES command

Lock and Unlock Tables – Syntax

- LOCK TABLES tbl_name [[AS] alias] lock_type [, tbl_name [[AS] alias] lock_type]…
- lock_type: READ [LOCAL] | [LOW_PRIORITY] WRITE
- UNLOCK TABLES

Lock and Unlock Tables – Example

- LOCK TABLES tOne AS TableOne WRITE, tTwo AS T2 READ, tThree READ, tFour READ;

- UNLOCK TABLES;

Lock and Unlock Tables

- A session can acquire or release locks only for itself
- One session cannot acquire locks for another session or release locks held by another session
- Locks may be used to emulate transactions or to get more speed when updating tables

Lock and Unlock Tables

- As of MySQL 4.0.2, to use LOCK TABLES you must have the LOCK TABLES privilege, and the SELECT privilege for each table to be locked
- In MySQL 3.23, you must have SELECT, INSERT, DELETE, and UPDATE privileges for all tables in a DB
- UNLOCK TABLES explicitly releases any table locks held by the current session

Lock and Unlock Tables

- The LOCK command applies only to non-TEMPORARY tables
  - LOCK TABLES is allowed (but ignored) for a TEMPORARY table
- If you use ALTER TABLE on a locked table, it may become unlocked
  - LOCK TABLE tOne WRITE;
  - alter table tOne add column colFour int;

Lock and Unlock Tables

Table locks are released implicitly under these conditions:

2. Beginning a transaction (for example, with START TRANSACTION) implicitly performs an UNLOCK TABLES
3. If a client connection drops, the server releases table locks held by the client

Lock and Unlock Tables

- A table lock protects only against inappropriate reads or writes by other clients
- The client holding the lock, even a read lock, can perform table-level operations such as DROP TABLE
- Truncate operations are not transaction-safe, so an error occurs if the client attempts the TRUNCATE command during an active transaction or while holding a table lock

Lock and Unlock Tables

- A session that requires locks must acquire all the locks that it needs in a single LOCK TABLES statement
- While the locks are held, the session can access only the locked tables
- For example, in the following sequence of statements, an error occurs for the attempt to access t2 because it was not locked in the LOCK TABLES statement: *NEXT SLIDE*

Lock and Unlock Tables

- mysql> LOCK TABLES t1 READ;
- mysql> SELECT COUNT(*) FROM t1;

```
+---------------+
| COUNT(*) |
+---------------+
| 3                |
+---------------+
```

- mysql> SELECT COUNT(*) FROM t2;
- ERROR 1100 (HY000): Table 't2' was not locked with LOCK TABLES

Lock and Unlock Tables

- You cannot refer to a locked table multiple times in a single query using the same name
- Use aliases instead, and obtain a separate lock for the table and each alias:
  - mysql> LOCK TABLE t WRITE, t AS t1 READ;
  - mysql> INSERT INTO t SELECT * FROM t;
  - ERROR 1100: Table 't' was not locked with LOCK TABLES
  - mysql> INSERT INTO t SELECT * FROM t AS t1;

Lock and Unlock Tables

In the previous slide:

- The error occurs for the first INSERT because there are two references to the same name for a locked table
- The second INSERT succeeds because the references to the table use different names

Lock and Unlock Tables

- If your statements refer to a table by means of an alias, you must lock the table using that same alias
- It does not work to lock the table without specifying the alias:
  - mysql> LOCK TABLE t READ;
  - mysql> SELECT * FROM t AS myalias;
  - ERROR 1100: Table 'myalias' was not locked with LOCK TABLES

Lock and Unlock Tables

- Conversely, if you lock a table using an alias, you must refer to it in your statements using that alias:
  - mysql> LOCK TABLE t AS myalias READ;
  - mysql> SELECT * FROM t;
  - ERROR 1100: Table 't' was not locked with LOCK TABLES
  - mysql> SELECT * FROM t AS myalias;

Lock and Unlock Tables

- LOCK TABLES acquires locks as follows:
2. Sort all tables to be locked in an internally defined order
3. If a table is to be locked with a read and a write lock, put the write lock request before the read lock request