CSIT115 Data Management and Security

CSIT882 Data Management Systems

# Data Integrity

Subject Coordinators: Dr Chen Chen, Dr Thanh Le

School of Computing and Information Technology -
University of Wollongong

# Data Integrity

## Outline

# Data integrity ? What is it ?

A term data integrity refers to the overall completeness, accuracy and consistency of data

This can be indicated by the absence of alteration between two instances or between two updates of a data record, meaning data is intact and unchanged

A term data integrity refers to maintaining and assuring the accuracy and consistency of data over its entire life-cycle

Data integrity is a critical aspect to the design, implementation and usage of any system which stores, processes, or retrieves data

Data integrity is the opposite to data corruption, which is a form of data loss

The overall intent of any data integrity technique is the same:
to ensure, that data is recorded exactly as intended and upon later retrieval, to ensure, that data is the same as it was when it was originally recorded

In HTML view press 'p' to see the lecture notes

# Data integrity ? What is it ?

In short, data integrity aims to prevent the unintentional changes to information

# Data Integrity

## Outline

# Consistency constraints

A consistency constraint is a property, that is always valid in a fragment of the real world modelled by a database

A consistency constraint is a condition, that must be satisfied by every persistent state of a database

For example:

- an attribute student-number uniquely identifies each student
- a budget of a small ARC grant cannot exceed 10K
- a value of attribute date-of-birth can be unknown
- an employee is a member of precisely one department
- a salary of full professor is in a range from x to y

# Consistency constraints

## Key constraint: primary and candidate key constraint

Relational schema

```
STUDENT(snum, first-name, last-name, date-of-birth, medicare-num, degree )
primary key= (snum)
candidate key 1 = (first-name, last-name, date-of-birth)
candidate key 2 = (medicare-num)
```

Relational schema

```
SUBJECT(code, title, credits)
primary key = (code)
candidate key = (title)
```

# Consistency constraints

## Referential integrity constraint: foreign key

Relational schema

```
ENROLMENT(snum, code, edate)
primary key = (snum, code, edate)
foreign key 1 = (snum) references STUDENT(snum)
foreign key 2 = (code) references SUBJECT(code)
```

Relational schema

```
COUNTRY(name)
primary key = (name)
CUSTOMER(cnum, first-name, last-name, country-name)
primary key: (cnum)
foreign key country-name references COUNTRY(name)
```

In HTML view press 'p' to see the lecture notes

# Consistency constraints

## NULL/NOT NULL constraint

NULL/NOT NULL constraint

```
STUDENT(snum, first-name, last-name, date-of-birth, medicare-num, degree )
degree    ...    NOT NULL
```

## Attribute type constraint

Attribute type constraint

```
STUDENT(snum, first-name, last-name, date-of-birth, medicare-num, degree )
snum   DECIMAL(7)  ...
```

## Domain constraint

Relational schema

```
SUBJECT(code, title, credits)
credits IN (6, 12)
```

# Consistency constraints

## Other constraints

- Numerical constraint: total number of rows in a table `EMPLOYEE` is less than 1000
- Exclusion constraint: a student cannot be in the same moment undergraduate and postgraduate student
- Distributed (multitable) referential integrity constraint:

Distributed referential integrity constraint

```
UNDERGRADUATE-STUDENT(snum, first-name, last-name, date-of-birth)
POSTGRADUATE-STUDENT(snum, first-name, last-name, date-of-birth)
SCHOLARSHIP(snum, amount)
SCHOLARSHIP.snum references either UNDERGRADUATE-STUDENT.snum or
                               POSTGRADUATE-STUDENT.snum
```

- Subset constraint:

Subset constraint

```
EMPLOYEE( enum, ... , city, ... )
PROJECT(pnum, ... , city, ... )
PROJECT.city is included in EMPLOYEE.city
```

- ... and many, many other constraints

In HTML view press 'p' to see the lecture notes

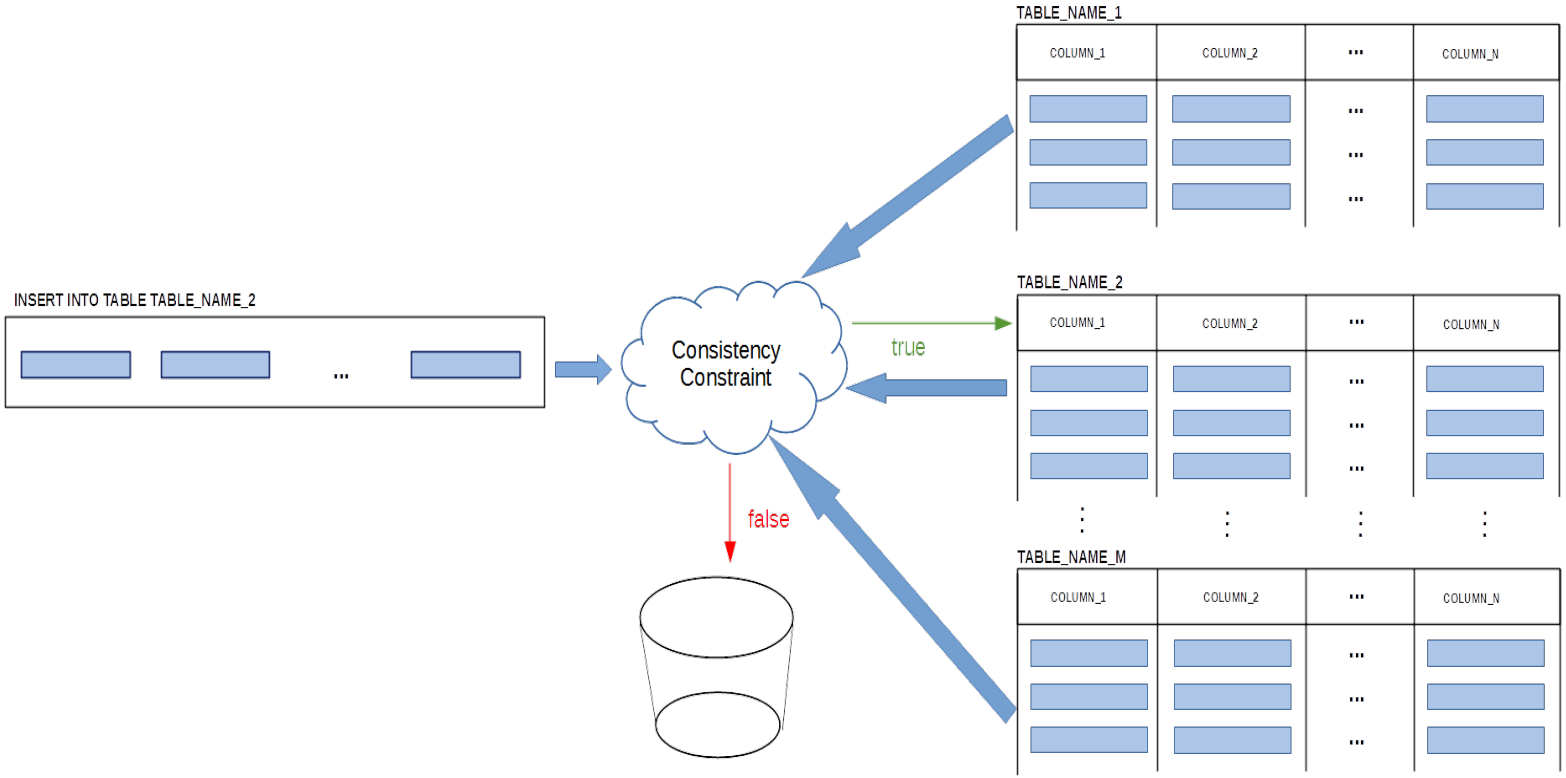
# Data Integrity

## Outline

# Consistency constraints

How do we enforce consistency constraints ?

- Define a consistency constraint in `CREATE TABLE` statement
- Then, the verification is performed by a database server

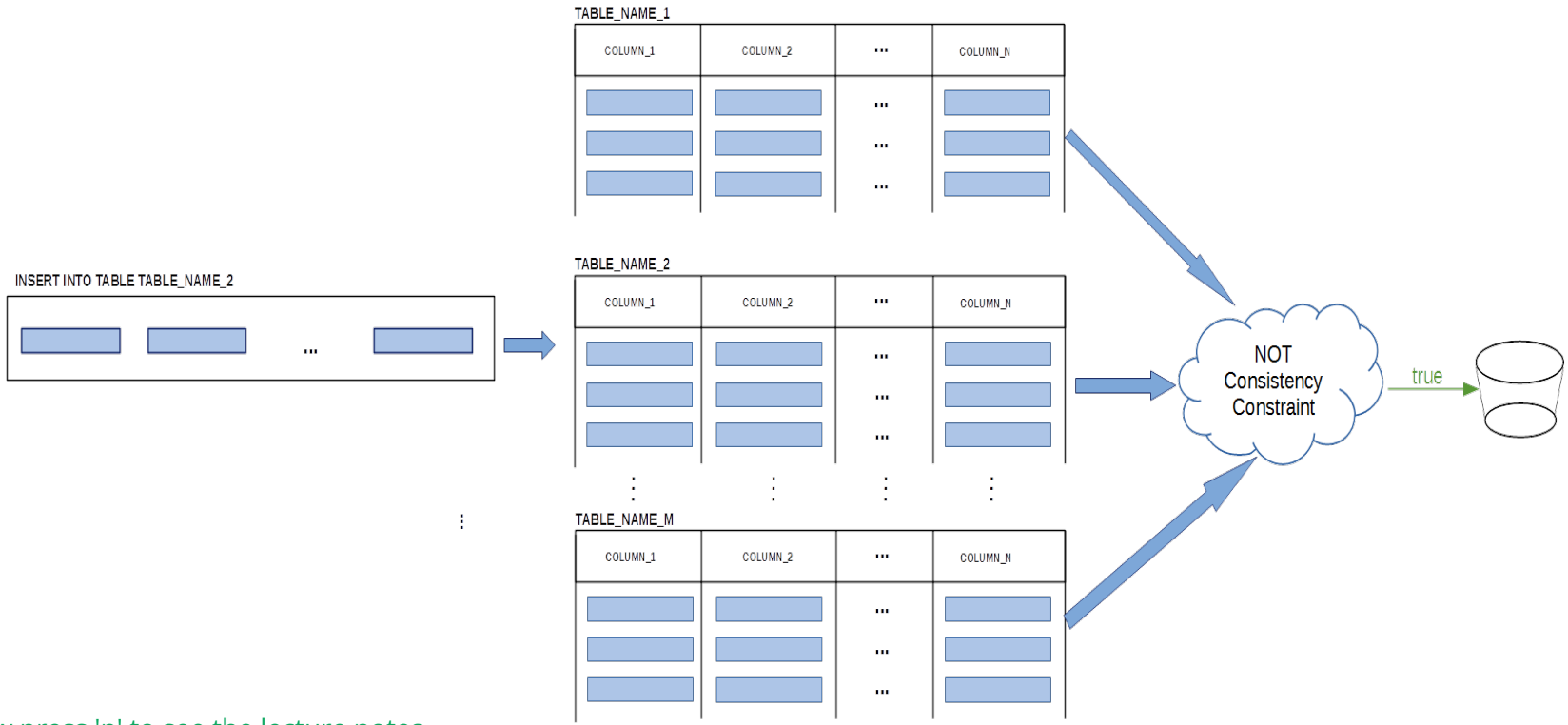



In HTML view press 'p' to see the lecture notes

# Consistency constraints

How do we enforce consistency constraints ?

- Implement verification of a consistency constraint as SQL script reporting violation of consistency constraints and process the script from time to time
- Then, the verification is performed by a database server




In HTML view press 'p' to see the lecture notes

# Consistency constraints

How do we enforce consistency constraints ?

- Implement verification of a consistency constraint within a database application, for example Java application accessing a relational database through Java DataBase Connectivity (JDBC)
- Then, the verification is performed by a database application
- Implement verification of consistency constraints within a stored procedure or stored function
- Then, the verification is performed by a database server
- Implement verification of consistency constraints as a servlet, php code, etc
- Then, the verification is performed by a Web server
- Implement verification of consistency constraints as a database trigger
- Then, the verification is performed by a database server

# Consistency constraints

Verification of a consistency constraint through SQL script

Assume that a relational table `EMPLOYEE` contains information common to all employees and relational table `DRIVER` and `ADMIN` contains information specific to drivers and administration people

We would like to enforce a multitable constraint saying, that a table `EMPLOYEE` must contain only information about the drivers and admin people, it means, that if a person is recorded in `EMPLOYEE` table then the person must be recorded in `ADMIN` or `DRIVER` table or in both tables

The following `SELECT` statement included in SQL script verifies the constraint and lists all rows in `EMPLOYEE` table that violate the constraint

SELECT statement

```
SELECT 'Multitable constraint failed, employee' AS "Constraint",
       enum, 'is not included in either DRIVER or ADMIN tables' AS "Condition"
FROM EMPLOYEE
WHERE enum NOT IN (SELECT enum
                   FROM DRIVER)
and
enum NOT IN (SELECT enum
                   FROM ADMIN);
```

In HTML view press 'p' to see the lecture notes

# Consistency constraints

The following `SELECT` statement

SELECT statement that verifies a consistency constraint

```
SELECT 'Multitable constraint failed, employee' AS "Constraint",
       enum, 'is not included in either DRIVER or ADMIN tables' AS "Condition"
FROM EMPLOYEE
WHERE enum NOT IN (SELECT enum
                   FROM DRIVER)
and
enum NOT IN (SELECT enum
                   FROM ADMIN);
```

may return the following results

Sample results from SELECT statement above

```
+---------------------------------------+------+-----------------------------------------------------+
| Constraint                            | enum | Condition                                           |
+---------------------------------------+------+-----------------------------------------------------+
| Multitable constraint failed, employee |   20 | is not included in either DRIVER or ADMIN tables |
+---------------------------------------+------+-----------------------------------------------------+
```

In HTML view press 'p' to see the lecture notes

# Consistency constraints

Verification of a consistency constraint through stored routines (functions and procedures)

A stored routine is a piece of code whose logic is usually implemented in a general purpose procedural language (host language) including embedded SQL statements to communicate with a database server

A stored routine is stored in a data dictionary (data repository) of a database management system, for example information_system database on MySQL

A stored procedure `country_hos`

A stored procedure

```
CREATE PROCEDURE country_hos
(IN con CHAR(20))
BEGIN
  SELECT Name, HeadOfState FROM Country
  WHERE Continent = con;
END;
```

# Consistency constraints

A stored procedure `country_hos` is invoked using `CALL` statement in the following way

Calling a stored procedure

```
CALL country_hos('Europe');
```

The following call to a stored procedure `insert_employee` can be used instead of `INSERT` statement to verify the consistency constraints within the procedure

Calling a stored procedure

```
CALL insert_employee(123456, 'James', 'Bond', '1960-12-12', 'MI6');
```

A stored procedure does not have to return a value and it can modify its parameters for a later inspection by a caller

A stored procedure can also generate the result sets to be returned to the client program through one of its parameters

In HTML view press 'p' to see the lecture notes

# Consistency constraints

A stored function can be used much like a built-in row function

When a stored function is invoked in an expression, then it returns a value during the evaluation of the expression

Stored function

```
CREATE FUNCTION CustomerLevel(p_creditLimit double) RETURNS VARCHAR(10)
  DETERMINISTIC
BEGIN
  DECLARE lvlvarchar(10);
    IF p_creditLimit> 50000 THEN
        SET lvl = 'PLATINUM';
    ELSEIF (p_creditLimit<= 50000 AND p_creditLimit>= 10000) THEN
        SET lvl = 'GOLD';
    ELSEIF p_creditLimit< 10000 THEN
        SET lvl = 'SILVER';
    END IF;
RETURN (lvl);
END;
```

Then it can be used in `SELECT` statement in the following way

Calling a stored function

```
SELECT customerName, CustomerLevel(creditLimit)
FROM customers
GROUP BY customerName;
```

In HTML view press 'p' to see the lecture notes

# Consistency constraints

Verification of a consistency constraint through the database triggers

A database trigger is a named database object associated with a table and such that it is automatically activated when a particular event occurs for the table

Some uses for database triggers are to evaluate consistency constraints after the modifications or to perform the calculations of the values of derived attributes

The database rriggers can also be used to enforce sophisticated database security constraints and/or to audit suspicious database activities, like for example an update to a column `SALARY` performed on Sunday

A database trigger is created through `CREATE TRIGGER` statement

A database trigger is activated when a statement inserts, updates, or deletes the rows in the associated table

# Consistency constraints

For example, the rows can be inserted by `INSERT` statements, and then insert trigger is automatically activated for each inserted row

A database trigger can be also activated either before or after the trigger event

We can activate a database trigger before each row is inserted into a table or after each row that is updated.

For example:

CREATE TRIGGER statement

```
CREATE TRIGGER upd_check BEFORE UPDATE ON account
FOR EACH ROW
BEGIN
  IF NEW.amount < 0 THEN
    ROLLBACK;
  END IF;
END;
```

# Consistency constraints

In another application a database trigger is used to audit the updates on a relational table `EMPLOYEE`

CREATE TRIGGER statement

```
CREATE TRIGGER before_employee_update
BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
  INSERT INTO employees_audit( SETaction='update',
                               employeeNumber=OLD.employeeNumber,
                               lastname=OLD.lastname, changedat=NOW() );
END;
```

# Data Integrity

## Outline

# `ROLLBACK` and `COMMIT` statements

Database systems allow for the immediate reversals of the recent modifications with `ROLLBACK` statement

On the other side, `COMMIT` statement makes all modifications performed since the beginning of a session or since the latest processing of `COMMIT` statement, permanent in a database and it also makes the reversal of such modification with `ROLLBACK` statement impossible

All modifications performed since the latest `COMMIT` statement or the beginning of a sesson can be reversed with `ROLLBACK` statement

# `ROLLBACK` and `COMMIT` statements

A system variable `AUTOCOMMIT` can be used to control making the database modifications permanent

If `AUTOCOMMIT = 'OFF'` then all modifications to a database are committed by either `COMMIT` statement or any data definition statement like `CREATE`, `ALTER`, `DROP`

If `AUTOCOMMIT = 'ON'` then all modifications to a database are immediately and automatically committed at the end of processing of the data manipulation statement

`EXIT` statement at the end of a session does not commit the modifications

By default, a system variable `AUTOCOMMIT` is set to `'ON'` at the beginning of a session

A value of `AUTOCOMMIT` variable can be changed in the same way as the values of other system variables

In HTML view press 'p' to see the lecture notes

# ROLLBACK and COMMIT statements

We use a sample database that consists of the following tables

CREATE TABLE statement

```
CREATE TABLE DEPARTMENT(
name                VARCHAR(50)        NOT NULL,
code                CHAR(5)            NOT NULL,
total_staff_number  DECIMAL(2)         NOT NULL,
chair               VARCHAR(50)            NULL,
budget              DECIMAL(9,1)       NOT NULL,
  CONSTRAINT dept_pkey PRIMARY KEY(name),
  CONSTRAINT dept_ckey1 UNIQUE(code),
  CONSTRAINT dept_ckey2 UNIQUE(chair),
  CONSTRAINT dept_check1 CHECK (total_staff_number BETWEEN 1 AND 50) );
```

CREATE TABLE statement

```
CREATE TABLE COURSE(
cnum                CHAR(7)            NOT NULL,
title               VARCHAR(200)       NOT NULL,
credits             DECIMAL(2)         NOT NULL,
offered_by          VARCHAR(50)            NULL,
  CONSTRAINT course_pkey PRIMARY KEY(cnum),
  CONSTRAINT course_check1 CHECK (credits IN (6, 12)),
  CONSTRAINT course_fkey1 FOREIGN KEY(offered_by)
                        REFERENCES DEPARTMENT(name)  );
```

In HTML view press 'p' to see the lecture notes

# ROLLBACK and COMMIT statements

Assume, that we would like to delete the Department of Arts and `AUTOCOMMIT = 'ON'`

First, we delete all courses offered by the Department of Arts

DELETE statement

```
DELETE FROM COURSES WHERE offered_by = 'Arts';
```

With `AUTOCOMMIT = 'ON'` such statement is immediately committed and the results of the deletion are available to the other user of the same database

Assume, that at this point, another user processes `SELECT` statements, that find the total number of departments and the total number of courses offered by each department

SELECT statement

```
SELECT COUNT(*) FROM DEPARTMENT;
```

SELECT statement

```
SELECT offered_by, count(*) FROM COURSE GROUP BY offered_by;
```

The user querying a database gets incorrect information, that the Department of Arts still exists and it offers no courses !

In HTML view press 'p' to see the lecture notes

# ROLLBACK and COMMIT statements

Now, assume, that we would like to delete the Department of Arts and `AUTOCOMMIT = 'OFF'`

First we delete all courses offered by the Department of Arts

```
DELETE FROM COURSES WHERE offered_by = 'Arts';
```
DELETE statement

With `AUTOCOMMIT = 'OFF'` such statement is not immediately committed and the results of the deletion are not available to other users of the same database

Assume, that at this point, another user processes `SELECT` statements, that find the total number of departments and the total number of courses offered by each department

```
SELECT COUNT(*) FROM DEPARTMENT
```
SELECT statement

```
SELECT offered_by, count(*) FROM COURSE GROUP BY offered_by
```
SELECT statement

Another user gets correct information because the deletions have not been committed yet and another user does not see the deletions

In HTML view press 'p' to see the lecture notes

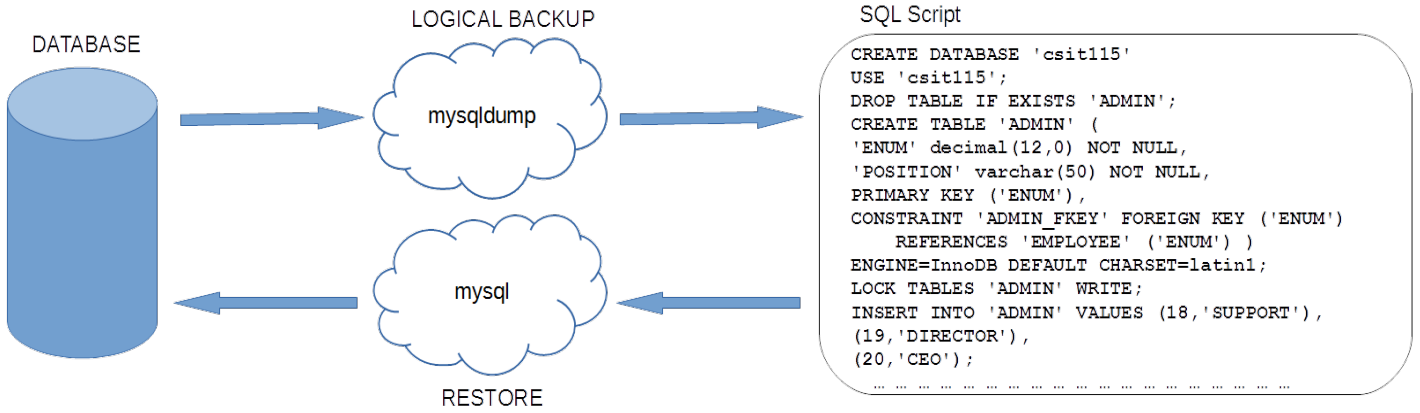
# Data Integrity

## Outline

# Backup and recovery

## Physical backup versus Logical backup

- A physical backup consists of the raw copies of the folders and files, that store database contents
- A physical backup is suitable for the large, important databases, that need to be recovered quickly when the problems occur
- A logical backup saves information represented as a logical database structure (`CREATE DATABASE, CREATE TABLE` statements) and the contents (`INSERT` statements or delimited-text files)




In HTML view press 'p' to see the lecture notes

# Backup and recovery

## Physical backup versus Logical backup

- Physical backup is more compact than logical backup
- Application of physical backup is faster than logical backup because it involves only copying files without any conversion
- Physical backup can be performed while the MySQL server is not running
- If a database server is running, then it is necessary to perform the appropriate locking, so that the server does not change the database contents during the backup
- Logical backup is done by querying MySQL server to obtain the database structures and the database contents
- Logical backup is slower than physical backup because the server must access database information and convert it into a logical format
- Output from logical backup is larger than from physical backup, particularly when saved in a text format
- Granularity of logical backup and restore is available at the server level (all databases), at a database level (all tables in a particular database), or at a table level

# Backup and recovery

## Physical backup versus Logical backup

- Logical backup is performed with the MySQL server running and the server does not need to be taken offline
- Logical backup is suitable for the smaller amounts of data where we might edit the data values or the table structures, or recreate the data on a different machine architecture

# Backup and recovery

## Logical backup with `mysqldump` program

`mysqldump` program produces two types of output, depending on whether `--tab` option is used

- Without `--tab` option `mysqldump` writes SQL statements to the standard output
- The output consists of `CREATE` statements, that create the dumped objects (databases, tables, stored routines, and so forth), and `INSERT` statements, that load data into the tables
- The output can be saved in a file and reloaded later using `mysql` command line interface to recreate the dumped objects
- Options are available to modify the format of the SQL statements, and to control which objects are dumped

# Backup and recovery

Logical backup with `mysqldump` program

`mysqldump` program produces two types of output, depending on whether the `--tab` option is used

- With `--tab` option `mysqldump` produces two output files for each dumped table
- The server writes one file as tab-delimited text, one line per table row
- This file is named `tbl_name.txt` in the current folder
- The server also sends a `CREATE TABLE` statements to `mysqldump`, which writes it as a file named `tbl_name.sql` in the current folder.

# Backup and recovery

The examples of logical backup created with `mysqldump` program without `--tab` option, it means, that the outputs are directly written to a single SQL script file

By default, `mysqldump` writes information as SQL statements to the standard output that can be saved in a file with `file-name`

Starting 'mysqldump' program

```
mysqldump [arguments] > file-name
```

Example 1: `mysqldump` connects as a user `root`, prompts about a password, uses a `verbose` mode, `locks all` dumped tables, to prevent data inconsistencies, takes a backup of `all databases` and saves it in a file `dump.sql`

Starting 'mysqldump' program with parameters

```
mysqldump --user root --password --verbose --lock_tables
          --all-databases > dump.sql
```

# Backup and recovery

Example 2: `mysqldump` connects as a user `root`, prompts about a password, uses a `verbose` mode, `locks all` dumped tables to prevent data inconsistencies, takes a backup of `csit115` database and saves it in a file `csit115dump.sql`

Starting 'mysqldump' program with parameters

```
mysqldump --user root --password --verbose --lock_tables
          --databases csit115 > csit115dump.sql
```

# Backup and recovery

## Sample contents of a file `csit115dump.sql`

Sample contents of logical backup

```
CREATE DATABASE /*!32312 IF NOT EXISTS*/ 'csit115' /*!40100 DEFAULT CHARACTER SET latin1 */;
USE 'csit115';
DROP TABLE IF EXISTS 'ADMIN';
CREATE TABLE 'ADMIN' (
  'ENUM' decimal(12,0) NOT NULL,
  'POSITION' varchar(50) NOT NULL,
  PRIMARY KEY ('ENUM'),
  CONSTRAINT 'ADMIN_FKEY' FOREIGN KEY ('ENUM') REFERENCES 'EMPLOYEE' ('ENUM')
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
LOCK TABLES 'ADMIN' WRITE;
INSERT INTO 'ADMIN' VALUES (18,'SUPPORT'),(19,'DIRECTOR'),(20,'CEO');
UNLOCK TABLES;
DROP TABLE IF EXISTS 'DRIVER';
CREATE TABLE 'DRIVER' (
  'ENUM' decimal(12,0) NOT NULL,
  'LNUM' decimal(8,0) NOT NULL,
  'STATUS' varchar(10) NOT NULL,
  PRIMARY KEY ('ENUM'),
  UNIQUE KEY 'DRIVER_UNIQUE' ('LNUM'),
  CONSTRAINT 'DRIVER_FKEY' FOREIGN KEY ('ENUM') REFERENCES 'EMPLOYEE' ('ENUM')
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
LOCK TABLES 'DRIVER' WRITE;
INSERT INTO 'DRIVER' VALUES (1,10001,'AVAILABLE'),(2,10008,'ON LEAVE'),(3,10002,'AVAILABLE'),
(4,10004,'AVAILABLE'),(5,10003,'ON LEAVE'),(6,10012,'AVAILABLE'),(7,20002,'BUSY'),
(8,20003,'BUSY'),(9,30005,'BUSY'),(10,40002,'BUSY'),(11,20045,'AVAILABLE');
UNLOCK TABLES;
... ... ...
```

In HTML view press 'p' to see the lecture notes

# Backup and recovery

To restore a database `csit115`: connect as a user `csit115`

Connecting as 'csit115' user

```
mysql -u csit115 -p -v
```

Drop a database `csit115` and exit `mysql`

Dropping a database

```
DROP DATABASE csit115;
exit;
```

Connect as a user `csit115` and restore a database `csit115`

Restoring a database from a logical backup

```
mysql -u csit115 -p -v < csit115dump.sql
```

# Backup and recovery

Example 3: `mysqldump` connects as a user `root`, prompts about a password, uses a `verbose` mode, `locks all` dumped tables to prevent data inconsistencies, takes a backup of `EMPLOYEE` and `DRIVER` tables located in a `csit115` database and saves it in a file `empdriv.sql`

```
                                                Taking a logical backup of selected relational tables
mysqldump csit115 EMPLOYEE DRIVER --user root --password
                                  --verbose --lock_tables > empdriv.sql
```

To restore the tables `EMPLOYEE` and `DRIVER`: connect as a user `csit115`

```
                                                Connecting as 'csit115' user
mysql -u csit115 -p -v
```

Drop the tables `EMPLOYEE` and `DRIVER` and exit `mysql`

```
                                                Dropping foreign key constraints and relational tables
ALTER TABLE TRIP DROP FOREIGN KEY trip_fkey1;
ALTER TABLE ADMIN DROP FOREIGN KEY admin_fkey;
DROP TABLE DRIVER;
DROP TABLE EMPLOYEE;
exit;
```

# Backup and recovery

Recreate and restore tables `DRIVER` and `EMPLOYEE` from a backup file `empdriv.sql`

Restoring the relational tables from a logical backup

```
mysql csit115 –u root –p <  empdriv.sql
```

Recreate referential integrity constraints

Recreating foreign key constraints

```
ALTER TABLE ADMIN ADD CONSTRAINT admin_fkey
                      FOREIGN KEY(ENUM) REFERENCES EMPLOYEE(ENUM);
ALTER TABLE TRIP ADD CONSTRAINT trip_fkey1
                     FOREIGN KEY (LNUM) REFERENCES DRIVER(LNUM);
```

# References

MySQL 8.0 Reference Manual, 13.3 Transactional and Locking Statements

MySQL 8.0 Reference Manual, 7.4 Using mysqldump for Backup