Database Administration: Security and Integrity

University of California, Berkeley

School of Information

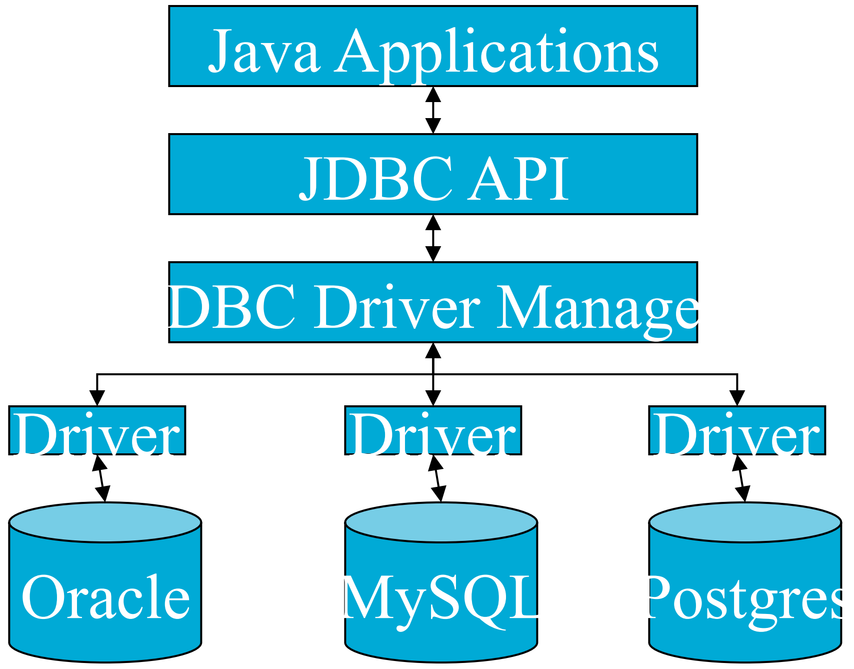
IS 257: Database Management

Security and Integrity Functions in Database Administration

- Review
  - – JDBC and MySQL
  - – Python and MySQL
- Data Integrity
- Security Management
- Backup and Recovery

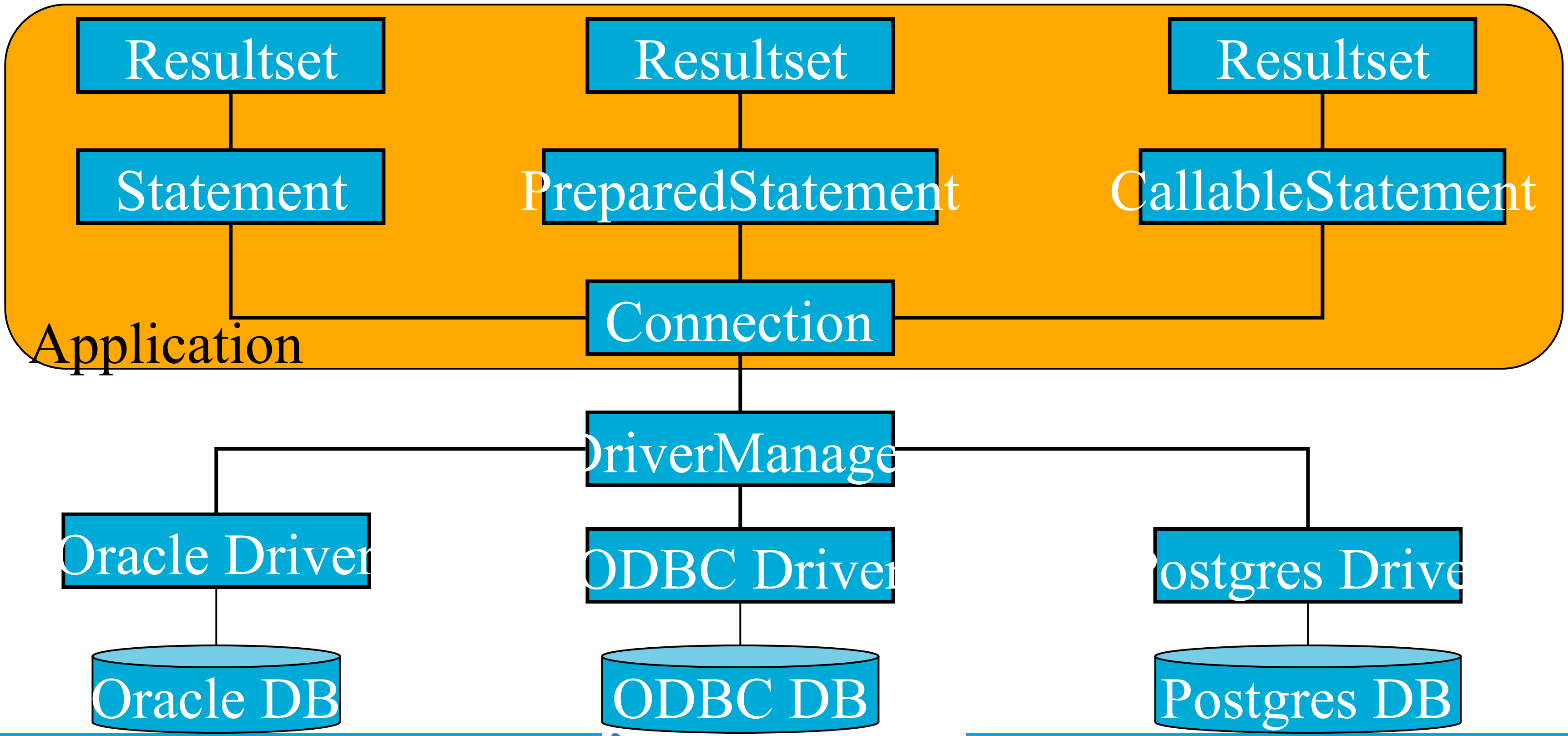
JDBC Architecture

- The goal of JDBC is to be a generic SQL database access framework that works for any database system with no changes to the interface code



JDBC

- Provides a standard set of interfaces for any DBMS with a JDBC driver – using SQL to specify the databases operations.



JDBC Simple Java Implementation

```
import java.sql.*;

public class JDBCTestMysqlHarbinger {

    public static void main(java.lang.String[] args) {
            try {
                // this is where the driver is loaded
                Class.forName("com.mysql.jdbc.Driver").newInstance

            }
            catch (InstantiationException i) {
                System.out.println("Unable to load driver Class");
                return;
            }
            catch (ClassNotFoundException e) {
                System.out.println("Unable to load driver Class");
                return;
            }
```

JDBC Simple Java Impl.

```
        try {
//All DB accees is within the try/catch block...
Connection con = DriverManager.getConnection("jdbc:mysq
                                             /ray?user=ray&password=XXXXXX
// Do an SQL statement...
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT name FROM D
```

JDBC Simple Java Impl.

```
        // show the Results...
        while(rs.next()) {

                System.out.println(rs.getString("Name"));
                System.out.println("");
        }

        // Release the db resources...
        rs.close();
        stmt.close();
        con.close();
    }
    catch (SQLException se) {
        // inform user of errors...
            System.out.println("SQL Exception: " + se.getMe
                se.printStackTrace(System.out);
```

MySQLdb

- MySQLdb is a DB-API for MySQL
- The basic setup is fairly simple…
  - Pip install MySQL-python
  - Conda install mysql-python
- Or, if on harbinger it is already installed
- To use the interface…

MySQLdb

```
#!/usr/bin/python
import MySQLdb
…
cursor = db.cursor()
# Make a string of SQL commands…
sql = "SELECT * FROM DIVECUST"

try:
  # Execute the SQL command in a try/except in case of failure
  cursor.execute(sql)
  # Fetch all the rows in a list of lists.
  results = cursor.fetchall()
  for row in results:
    custno = row[0]
    custname = row[1]
    street = row[2]
    city = row[3]
    state = row[4]
    zip = row[5]
    country = row[6]
    # Now print fetched result
    print "%s : %s, %s, %s, %s %s" % \
         (custname, street, city, state, zip, country)
except:
  print "Error: unable to fetch data"
```

Can run any SQL…

```
#!/usr/bin/python
import MySQLdb
# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Drop table if it already exist using execute() method.
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")

# Create table as per requirement
sql = """CREATE TABLE EMPLOYEE (
         FIRST_NAME  CHAR(20) NOT NULL,
         LAST_NAME  CHAR(20),
         AGE INT,
         SEX CHAR(1),
         INCOME FLOAT )"””
cursor.execute(sql)

# disconnect from server
db.close()
```

MySQLdb

```
#!/usr/bin/python

import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to INSERT a record into the database.
sql = """INSERT INTO EMPLOYEE(FIRST_NAME,
         LAST_NAME, AGE, SEX, INCOME)
         VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""
try:
   # Execute the SQL command
   cursor.execute(sql)
   # Commit your changes in the database
   db.commit()
except:
   # Rollback in case there is any error
   db.rollback()

# disconnect from server
db.close()
```

MySQLdb

```
#!/usr/bin/python

import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to UPDATE required records
sql = "UPDATE EMPLOYEE SET AGE = AGE + 1
                          WHERE SEX = '%c'" % ('M')
try:
   # Execute the SQL command
   cursor.execute(sql)
   # Commit your changes in the database
   db.commit()
except:
   # Rollback in case there is any error
   db.rollback()

# disconnect from server
db.close()
```

Security and Integrity Functions in Database Administration

- Data Integrity (review)
- Security Management
- Backup and Recovery

Data Integrity

- Intrarecord integrity (enforcing constraints on contents of fields, etc.)
- Referential Integrity (enforcing the validity of references between records in the database)
- Concurrency control (ensuring the validity of database updates in a shared multiuser environment)

Integrity Constraints (review)

- The constraints we wish to impose in order to protect the database from becoming inconsistent.
- Five types
  - Required data
  - attribute domain constraints
  - entity integrity
  - referential integrity
  - enterprise constraints

Required Data

- Some attributes must always contain a value -- they cannot have a NULL value
- For example:
  - Every employee must have a job title.
  - Every diveshop diveitem must have an order number and an item number

Attribute Domain Constraints

- Every attribute has a *domain*, that is a set of values that are legal for it to use
- For example:
  - The domain of *sex* in the employee relation is “M” or “F”
- Domain ranges can be used to validate input to the database

Entity Integrity

- The primary key of any entity:
  - Must be Unique
  - Cannot be NULL

Referential Integrity

- A "**foreign key**" links each occurrence in a relation representing a child entity to the occurrence of the parent entity containing the matching candidate (usually primary) key
- Referential Integrity means that if the foreign key contains a value, that value must refer to an existing occurrence in the parent entity
- For example:
  - Since the Order ID in the diveitem relation refers to a particular diveords item, that item must exist for referential integrity to be satisfied.

Referential Integrity

- Referential integrity options are declared when tables are defined (in most systems)
- There are many issues having to do with how particular referential integrity constraints are to be implemented to deal with insertions and deletions of data from the parent and child tables.

Insertion rules

- A row should not be inserted in the referencing (child) table unless there already exists a matching entry in the referenced table
- Inserting into the parent table should not cause referential integrity problems
- Sometimes a special NULL value may be used to create child entries without a parent or with a "dummy" parent

Deletion rules

- A row should not be deleted from the referenced table (parent) if there are matching rows in the referencing table (child)
- Three ways to handle this
  - Restrict -- disallow the delete
  - Nullify -- reset the foreign keys in the child to some NULL or dummy value
  - Cascade -- Delete all rows in the child where there is a foreign key matching the key in the parent row being deleted

Referential Integrity

- This can be implemented using external programs that access the database
- newer databases implement executable rules or built-in integrity constraints (e.g. Access and Oracle)

Enterprise Constraints

- These are business rule that may affect the database and the data in it
  - for example, if a manager is only permitted to manage 10 employees then it would violate an enterprise constraint to manage more

Data and Domain Integrity

- This is now increasing handled by the database. In Oracle or MySQL, for example, when defining a table you can specify:
- CREATE TABLE table-name (

attr2 attr-type NOT NULL, *forbids NULL values*

attrN attr-type CHECK (attrN = UPPER(attrN)) *verifies that the data meets certain criteria*

attrO attr-type DEFAULT default_value); *Supplies default values*

Remember that not all of these work in all MySQL engines

Referential Integrity

- Ensures that dependent relationships in the data are maintained. In Oracle or MySQL, for example:
- CREATE TABLE table-name (

  attr1 attr-type PRIMARY KEY,

  attr2 attr-type NOT NULL,

…, attrM attr-type REFERENCES tablename(attrname) *ON DELETE CASCADE*, …

These have many additional options…

Concurrency Control

- The goal is to support access by multiple users to the same data, at the same time
- It must assure that the transactions are *serializable* and that they are *isolated*
- It is intended to handle several problems in an uncontrolled system
- Specifically:
  - Lost updates
  - Inconsistent data states during access
  - Uncompleted (or committed) changes to data

No Concurrency Control: Lost updates

John

- Read account balance (balance = $1000)
- Withdraw $200 (balance = $800)
- Write account balance (balance = $800)

Marsha

- Read account balance (balance = $1000)
- Withdraw $300 (balance = $700)
- Write account balance (balance = $700)

ERROR!

Concurrency Control: Locking

- Locking levels
  - Database
  - Table
  - Block or page
  - Record
  - Field
- Types
  - Shared (S locks)
  - Exclusive (X locks)

Concurrency Control: Updates with X locking

John

- Lock account balance
- Read account balance (balance = $1000)
- Withdraw $200 (balance = $800)
- Write account balance (balance = $800)
- Unlock account balance

Marsha

- Read account balance (DENIED)
- Lock account balance
- Read account balance (balance = $800)
- etc...

Concurrency Control: Deadlocks

John

- Place S lock
- Read account balance (balance = $1000)
- Request X lock (denied)
- wait ...

Marsha

- Place S lock
- Read account balance (balance = $1000)
- Request X lock (denied)
- wait...

Deadlock!

Concurrency Control

- Avoiding deadlocks by maintaining tables of potential deadlocks and "backing out" one side of a conflicting transaction
- Normally strict Two-Phase locking (TPL or 2PL) is used. It has the characteristics that
  - Strict 2PL prevents transactions from reading uncommitted data, overwriting uncommitted data, and unrepeatable reads
  - It prevents cascading rollbacks (i.e. having to roll back multiple transactions), since eXclusive locks (for write privileges) must be held until a transaction commits

Lock compatibility table

Lock type	read-lock	write-lock
read-lock		X
write-lock	X	X

Transaction Control in ORACLE

- Transactions are sequences of SQL statements that ORACLE treats as a unit
  - From the user's point of view a private copy of the database is created for the duration of the transaction
- Transactions are started with SET TRANSACTION, followed by the SQL statements
- Any changes made by the SQL are made permanent by COMMIT
- Part or all of a transaction can be undone using ROLLBACK

Transactions in ORACLE

- COMMIT; *(I.e., confirm previous transaction)*
- SET TRANSACTION READ ONLY;
- SELECT NAME, ADDRESS FROM WORKERS;
- SELECT MANAGER, ADDRESS FROM PLACES;
- COMMIT;
- *Freezes the data for the user in both tables before either select retrieves any rows, so that changes that occur concurrently will not show up*
- *Commits before and after ensure any uncompleted transactions are finish, and then release the frozen data when done*

Transactions in ORACLE

- Savepoints are places in a transaction that you may ROLLBACK to (called checkpoints in other DBMS)
  - SET TRANACTION…;
  - SAVEPOINT ALPHA;
  - *SQL STATEMENTS…*
  - IF (CONDITION) THEN ROLLBACK TO SAVEPOINT ALPHA;
  - SAVEPOINT BETA;
  - *SQL STATEMENTS…*
  - IF …;
  - COMMIT;

Transactions in MySQL

- START TRANSACTION or BEGIN starts a transaction block (disables *autocommit*)
- COMMIT or ROLLBACK will commit the transaction block ***or*** return to state before the block was started
- MySQL may use different underlying database engines – the **InnoDB** engine also supports SAVEPOINT and ROLLBACK TO SAVEPOINT

- NOTE: This *syntax* can be used in any of MySQL's database engines - but it only ***WORKS*** when using the **InnoDB engine** (which can be set up when the tables are created)

Transactions in MySQL (5.0+)

- START TRANSACTION [WITH CONSISTENT SNAPSHOT] | BEGIN [WORK]
- COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
- ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
- SET AUTOCOMMIT = {0 | 1}

- The START TRANSACTION and BEGIN statement begin a new transaction. COMMIT commits the current transaction, making its changes permanent. ROLLBACK rolls back the current transaction, canceling its changes. *The SET AUTOCOMMIT statement disables or enables the default autocommit mode for the current connection*

MySQL: Explicit locking of tables

- LOCK TABLES
  tbl_name [[AS] alias] lock_type
  [, tbl_name [[AS] alias] lock_type] ...

- lock_type:
  READ [LOCAL] | [LOW_PRIORITY] WRITE

- UNLOCK TABLES

- MySQL enables client sessions to acquire table locks explicitly for the purpose of cooperating with other sessions for access to tables, or to prevent other sessions from modifying tables during periods when a session requires exclusive access to them. 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.

Security and Integrity Functions in Database Administration

- Data Integrity
- Security Management
- Backup and Recovery

Database Security

- Views or restricted subschemas
- Authorization rules to identify users and the actions they can perform
- User-defined procedures (with rule systems or triggers) to define additional constraints or limitations in using the database
- Encryption to encode sensitive data
- Authentication schemes to positively identify a person attempting to gain access to the database

Views

- A subset of the database presented to some set of users
  - SQL:

    CREATE VIEW viewname AS SELECT field1, field2, field3,…, FROM table1, table2 WHERE <where clause>;
  - Note: “queries” in Access function as views

Restricted Views

- Main relation has the form:

Name	C_name	Dept	C_dept	Prof	C_prof	*TC*
J Smith	S	Dept1	S	Cryptography	TS	TS
M Doe	U	Dept2	S	IT Security	S	S
R Jones	U	Dept3	U	Secretary	U	U

U = unclassified : S = Secret : TS = Top Secret

Restricted Views

S-view of the data

NAME	Dept	Prof
J Smith	Dept1	---
M Doe	Dept2	IT Security
R Jones	Dept3	Secretary

U-view of the data

NAME	Dept	Prof
M Doe	---	---
R Jones	Dept3	Secretary

Authorization Rules

- Most current DBMS permit the DBA to define "access permissions" on a table by table basis (at least) using the **GRANT** and **REVOKE** SQL commands
- Some systems permit finer grained authorization, such as by column (most use GRANT and REVOKE on variant views

Grant and Revoke

- When Database Administrators (DBA) set up a database, they can specify not only the users and their logins, but also what privileges they have for each table (and/or column) in the database
- Thus, you could, for example, grant SELECT on the Salary table for employees, but not INSERT or UPDATE

In MySQL

- GRANT *priv_type* [(*column_list*)] [, *priv_type* [(*column_list*)]] ... ON [*object_type*] *priv_level* TO *user_specification* [, *user_specification*] ... [REQUIRE {NONE | *ssl_option* [[AND] *ssl_option*] ...}] [WITH {GRANT OPTION | *resource_option*} ...]
- E.g. : GRANT SELECT (Name, Street) ON DIVECUST TO ‘yliu’@’localhost’;

In MySQL

- REVOKE *priv_type* [(*column_list*)] [, *priv_type* [(*column_list*)]] ... ON [*object_type*] *priv_level* FROM *user* [, *user*] ...
- REVOKE ALL PRIVILEGES, GRANT OPTION FROM *user* [, *user*] ...
- E.g.: REVOKE INSERT ON DIVECUST FROM ‘user’@’server’;

MySQL Access Privileges

Privilege	Column	Context
CREATE	Create_priv	databases, tables, or indexes
DROP	Drop_priv	databases, tables, or views
GRANT OPTION	Grant_priv	databases, tables, or stored routines
LOCK TABLES	Lock_tables_priv	databases
REFERENCES	References_priv	databases or tables
EVENT	Event_priv	databases
ALTER	Alter_priv	tables
DELETE	Delete_priv	tables
INDEX	Index_priv	tables
INSERT	Insert_priv	tables or columns
SELECT	Select_priv	tables or columns

MySQL Access Privileges

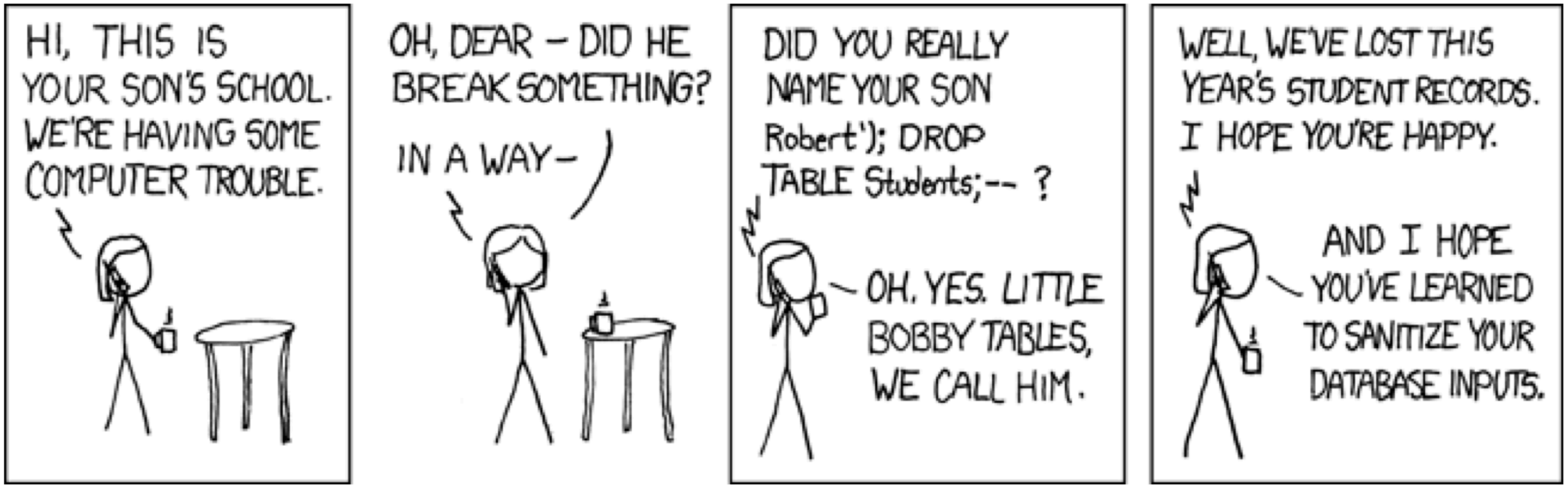
Privilege	Column	Context
UPDATE	Update_priv	tables or columns
CREATE TEMPORARY TABLES	Create_tmp_table_priv	tables
TRIGGER	Trigger_priv	tables
CREATE VIEW	Create_view_priv	views
SHOW VIEW	Show_view_priv	views
ALTER ROUTINE	Alter_routine_priv	stored routines
CREATE ROUTINE	Create_routine_priv	stored routines
EXECUTE	Execute_priv	stored routines
FILE	File_priv	file access on server host
CREATE TABLESPACE	Create_tablespace_priv	server administration
CREATE USER	Create_user_priv	server administration

MySQL Access Privileges

Privilege	Column	Context
PROCESS	Process_priv	server administration
PROXY	see proxies_priv table	server administration
RELOAD	Reload_priv	server administration
REPLICATION CLIENT	Repl_client_priv	server administration
REPLICATION SLAVE	Repl_slave_priv	server administration
SHOW DATABASES	Show_db_priv	server administration
SHUTDOWN	Shutdown_priv	server administration
SUPER	Super_priv	server administration
ALL [PRIVILEGES]		server administration
USAGE		server administration

Security in DB Applications

- The most common form of security violation on the web involves an SQL injection attack on a DB-driven web site


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Security in DB Applications

- An SQL Injection attack *only* works when data provided by users (such as the contents of a form) is inserted directly into SQL and submitted to the database system
- To avoid this, any input should be cleaned by
  - Removing any SQL reserved characters (like “'”, “;”, “)”, etc.
  - And possibly reserved words like “SELECT”, “DROP”, “TABLES”, etc.

Security in DB Applications

- Some Web Application Servers, like PHP, include functions to “sanitize” inputs
  - For example
    - mysql_real_escape_string($cname)
    - This basically just escapes quotes
    - Input string 'Louis'; DROP TABLE NEWCUST;'
    - Converted query ‘SELECT * FROM DIVECUST D where D.Name like '%Louis\'; DROP TABLE NEWCUST;%' ;

SQL Injection

- Also, as discussed last week, parameterized PHP can be used to avoid executing what *should be* data

Security and Integrity Functions in Database Administration

- Data Integrity
- Security Management
- Backup and Recovery – introduction (more next time)

Database Backup and Recovery

- Backups
- Journaling (audit trail)
- Checkpoint facility
- Recovery manager

- Info on Backups, etc. from MySQL docs http://dev.mysql.com/doc/refman/5.1/en/backup-and-recovery.html

MySQL Backup Types

- Physical (Raw) Versus Logical Backups
  - Physical (or Raw) Backups
    - Physical backups consist of raw copies of the directories and files that store database contents. This type of backup is suitable for large, important databases that need to be recovered quickly when problems occur.
  - Logical Backups
    - Logical backups save information represented as logical database structure (CREATE DATABASE, CREATE TABLE statements) and content (INSERT statements or delimited-text files). This type of backup is suitable for smaller amounts of data where you might edit the data values or table structure, or recreate the data on a different machine architecture.

From: http://dev.mysql.com/doc/refman/5.1/en/backup-types.html

Logical Backup

- The backup is done by querying the MySQL server to obtain database structure and content information.
- Backup is slower than physical methods because the server must access database information and convert it to logical format.
- Output is larger than for physical backup, particularly when saved in text format.
- Backup and restore granularity is available at the server level (all databases), database level (all tables in a particular database), or table level. This is true regardless of storage engine.
- The backup does not include log or configuration files, or other database-related files that are not part of databases.
- Backups stored in logical format are machine independent and highly portable.
- Logical backups are performed with the MySQL server running.

Logical Backups

- Logical backup tools include the **mysqldump** program and the **SELECT ... INTO OUTFILE** statement. These work for any storage engine, even MEMORY.
- To restore logical backups, SQL-format dump files can be processed using the **mysql** client. To load delimited-text files, use the **LOAD DATA INFILE** statement or the **mysqlimport** client.

Logical Backups

- Mysqldump –p [-X] *databasename tablename(s)*
- *Demo of normal and XML output*

Physical Backups

- The backup consists of exact copies of database directories and files. Typically this is a copy of all or part of the MySQL data directory.
- Physical backup methods are faster than logical because they involve only file copying without conversion.
- Output is more compact than for logical backup.
- Backup and restore granularity ranges from the level of the entire data directory down to the level of individual files.
- In addition to databases, the backup can include any related files such as log or configuration files.
- Backups are portable only to other machines that have identical or similar hardware characteristics.
- Backups can be performed while the MySQL server is not running. If the server is running, it is necessary to perform appropriate locking so that the server does not change database contents during the backup.

Physical Backups

- Physical backup tools include file system-level commands (such as cp, scp, tar, rsync), **mysqlhotcopy** for MyISAM tables, **ibbackup** for InnoDB tables, or **START BACKUP** for NDB tables.
- For restore, files copied at the file system level or with **mysqlhotcopy** can be copied back to their original locations with file system commands; **ibbackup** restores InnoDB tables, and **ndb_restore** restores NDB tables.