Data Definition, Index & Server Side Logic

Business Data Management and Analytics

Contributions by Arthur Adamopoulos and Vince Bruno

Defining Tables

```
CREATE TABLE customer (
 CustomerID     INT(10) NOT NULL
                        PRIMARY KEY,
 Surname        VARCHAR(40),
 Given          VARCHAR(40),
 DOB            DATETIME,
 Sex            CHAR(1),
 Phone          VARCHAR(20),
 Address        VARCHAR(40),
 Suburb         VARCHAR(40),
 State          VARCHAR(40),
 Postcode       VARCHAR(10)
);
```

```
CREATE TABLE mobile (
 MobileID       INT(10) NOT NULL
                        PRIMARY KEY,
 PhoneNumber    VARCHAR(20),
 BrandName      VARCHAR(40),
 Joined         DATETIME,
 Cancelled      DATETIME,
 PlanName       VARCHAR(20),
 PhoneColour    VARCHAR(20),
 CustomerID     INT(10),
 StaffID                INT(10)
);
```

MySQL Field Types

- Sample list of specific native types:
  - TINYINT, SMALLINT, MEDIUMINT, INT (INTEGER), BIGINT
  - DATETIME, DATE, TIMESTAMP, TIME, YEAR
  - FLOAT, REAL, DOUBLE PRECISION
  - DEC (DECIMAL), NUMERIC
  - CHAR, VARCHAR, BINARY and VARBINARY
  - BLOB and TEXT

ANSI Field Types (new)

- Set of types defined as a standard.
- MySQL maps these to native types
  - CHAR(<length>)
  - VARCHAR(<length>)
  - DATE
  - NUMERIC(<precision>,<scale>)
  - DECIMAL (<precision>,<scale>),
  - INT, FLOAT

Working with Tables

- Deleting a Table:

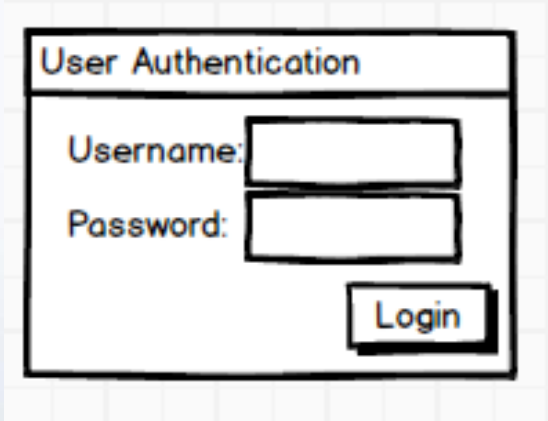
```
DROP TABLE customer;
```

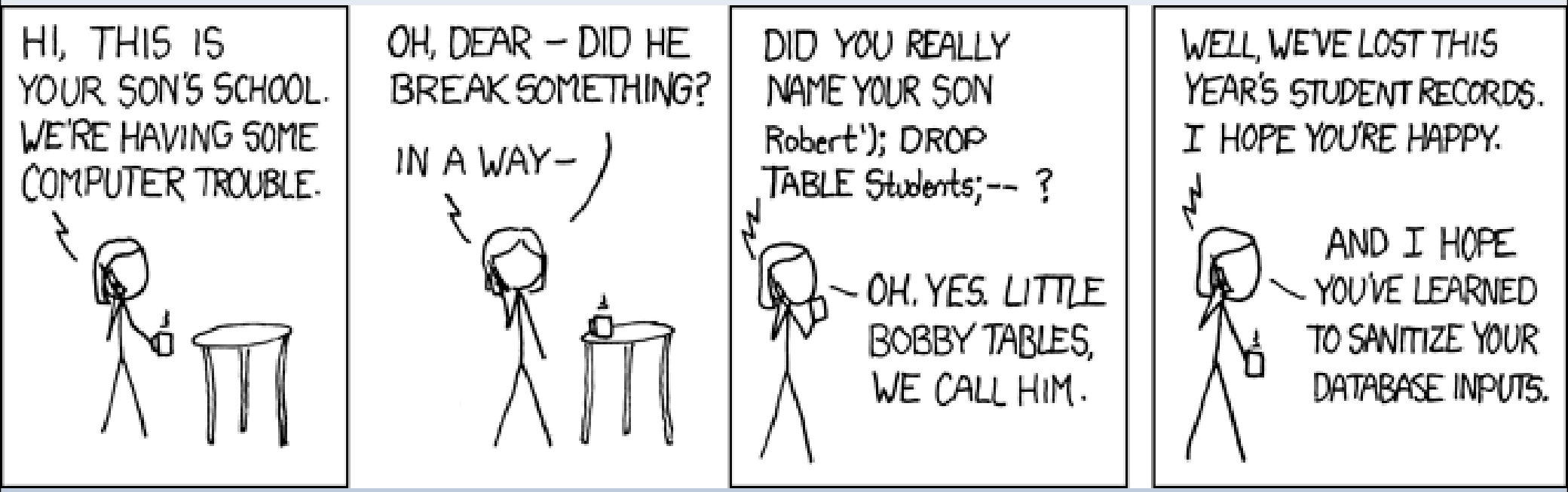
- Copying a table - creates a brand new table

```
CREATE TABLE SMITHCUST AS
SELECT *
FROM customer
WHERE SURNAME = 'SMITH';
```

Drop Table Story…

- USERNAME: vince'); DROP TABLE student; --
  PASSWORD: hello





Inserting Simple rows / records

```
INSERT INTO customer
    VALUES ('9001234J','Jones','Fred','01/01/70');
```

- Insert uses the order of fields on create to place values:

```
INSERT INTO customer(surname, dob, customerID, given)
    VALUES ('Jones', '01/01/70', '30001', 'Fred');
```

- Or you can specify the exact field order to insert into.
- Any column not listed is given a NULL value.
- If it is a NOT NULL column, insert statement will fail.

Inserting Records from other tables

```
INSERT INTO pink_mobiles
SELECT customerID, surname, given, phonenumber, joined
FROM customer c, mobile m
WHERE c.customerID = m. customerID
AND phonecolour = 'Pink';
```

- This command does not create a new table. The table must already exist.

Deleting Records

- By default, delete deletes <u>all</u> rows:

```
DELETE FROM mobile;
```

- To delete only selected rows, specify a where clause, which can contain all usual criteria:

```
DELETE FROM customer
WHERE customerID = 20002;
```

Updating Records

- Update is performed on every row in the table, unless constrained in a where clause.
- SET clause used to change values of fields.
- SET can contain calculations etc.
- Updates can also have nested queries, both in the where clause and the set clause:

```
UPDATE mobile
SET Cancelled = SYSDATE()
WHERE mobileID = 10023;
```

INDEXES

- An index speeds up searching and joining operations
- Indexes slow down updates, however.
- Index is simply created.
- System handles all updates to the index.
- The system decides if the index will be used.
- A command cannot specify the use of an index.

Indexes

```
CREATE INDEX CUSTSURN
    ON customer(surname);

CREATE UNIQUE INDEX mobilephonenum
    ON mobile(phonenumber);
```

Indexes

- Create Indexes on columns used for:
  - Primary keys (unique)
  - Foreign keys
  - search fields
  - ordering fields
- Do NOT create indexes on:
  - Fields with few different values
  - Small tables
  - NULL values

Indexes Example

```
CREATE INDEX planname_index
  ON mobile(planname);
```

MobileI	PhoneNumber	BrandName	Joined	Cancelled	PlanName	PhoneColour	Custome	Sta
4500	413448970	Samsung	2007-03-12	2007-11-02	Yes10	Blue	20002	10
4501	413941923	Nokia	2006-07-15	2007-07-29	Yes30	Transparent	20002	9
4502	410717359	NEC	2007-06-03	2009-01-24	Yes10	Red	20004	9
4503	412256126	Ericson	2006-04-05	2006-11-24	Yes10	Brown	20006	6
4504	410079801	Ericson	2007-05-02	(NULL)	FreeStyle	Grey	20006	7
4505	410589454	NEC	2006-08-14	(NULL)	Yes20	Rainbow	20008	5
4506	411229676	Nokia	2007-04-23	2008-01-22	Yes30	Pink	20008	8
4507	413980788	Nokia	2007-06-25	(NULL)	Yes10	Transparent	20010	7
4508	410783126	Ericson	2006-09-14	(NULL)	Yes20	Pink	20010	3
4509	410329528	Philips	2006-06-17	(NULL)	Yes20	Yellow	20012	9
4510	411607013	Nokia	2007-03-26	(NULL)	Yes40	Green	20012	7
4511	412093772	Nokia	2005-09-29	(NULL)	Yes30	Silver	20018	6
4512	413881812	Ericson	2006-02-17	(NULL)	FreeStyle	Brown	20024	4

Indexes Example

- Index is created and handled automatically in database, looks like this:

planname	mobileID
FreeStyle	4512,4504
Yes10	4507,4500,4503,4502
Yes20	4505,4508,4509
Yes30	4501,4511,4506
Yes40	4510

SELECT * FROM mobile
WHERE planname = 'Yes20'

- Without Index – 13 mobile records read
- Index used – 1 index record read + 3 mobile records read

Indexes Example

- If two new records are added the index is automatically updated:

4513	413612678	Nokia	2007-04-08	(NULL)	FreeStyle	Grey	20028	1
4514	411655779	Philips	2006-12-17	(NULL)	Yes20	Gold	20028	9

- Index now looks like this:

planname	mobileID
FreeStyle	4513,4512,4504
Yes10	4500,4507,4502,4503
Yes20	4505,4514,4509,4508
Yes30	4511,4506,4501
Yes40	4510

- For each INSERT, UPDATE or DELETE of the base table will initiate change to indexes

Server Side Logic

- Most DBMS's provide a procedural language that can be used to enforce complex business rules and run business logic on the server.
  - MySQL provides a simple language based on SQL
  - Oracle has a language called PL/SQL.
  - MS SQLServer has TransactSQL.
- Server side logic can be coded via:
  - Stored Procedures
  - User defined Functions
  - Triggers

Server Side Logic

- Most are procedural programming language roughly based on SQL.
- It has similar constructs to any other programming language such as variables, IF statement and Loops
- It also has special constructs such as cursors to allow looping through a table one row at a time.
- Results of SQL statements such as SELECT are not displayed to the user, but instead put into variables.

Stored Procedures

- A Stored procedure is a named block of procedural code which is compiled and stored on the server, in the schema of the user who created it.
- It is the same, conceptually, as a subroutine in any other programming language – like VB.NET & Java.
- It can be passed parameters.
- It can be called with the CALL command from another procedure.

Stored Procedure Example

```
CREATE PROCEDURE addNewMobile (v_customerID INTEGER)
BEGIN
    INSERT INTO mobile(customerID, joined)
    VALUES(v_customerID, SYSDATE() );
END ;
```

- To call this procedure from the SQLPlus prompt, from another procedure, or from VB or Java:

```
CALL addNewMobile( 21088 ) ;
```

User Defined Functions

- Functions are similar to Stored Procedures. They are named and stored on the server in the schema of the person who created them.
- They can be made available to other users.
- Syntax rules are exactly the same, main difference is that a function returns a value.
- Once compiled, a function can be used as if it was a native DBMS function – i.e. In a normal SELECT statement

User Defined Function Example

```
DELIMITER $$
CREATE FUNCTION calc_age( v_dob DATE ) RETURNS INT
BEGIN
  DECLARE v_age INT;
  SET v_age = truncate( datediff( sysdate(), v_dob) / 365.25 , 0 );
  RETURN  v_age;
END$$
DELIMITER ;
```

User Defined Function Example

- Users with execute privilege on function theYear could then use it in a SQL statement.

```
SELECT calc_age( DOB) FROM staff;

SELECT calc_age(SYSDATE) + 20;
```

calc_age(dob)
48
35
23
49
31
34
48
36
28
23

User Defined Function Example

```
CREATE FUNCTION hello ( v_name CHAR(20) )
    RETURNS CHAR(50) DETERMINISTIC NO SQL
BEGIN
    DECLARE v_message CHAR(50);
    SET v_message = CONCAT('Hello, ',v_name,'!');
    RETURN v_message;
END;
```

- Execute examples:

```
SELECT hello('Vince');
SELECT hello(surname) FROM customer;
```

hello('Vince')
Hello, Vince!

hello(surname)
Hello, RAJOO!
Hello, PHONGWATCHARARUK!
Hello, CHOW!
Hello, LIVERIADIS!
Hello, SAMARAWICKRAMA!
Hello, GILTRAP!
Hello, BINDEVIS!
Hello, QUAH!

Triggers

- Triggers are similar to Stored Procedures and Functions.
- Unlike Stored Procedures and Functions, triggers are not called explicitly by a user, procedure, function or program.
- When triggers are defined, they are attached to a particular table, for particular events such as INSERT, UPDATE or DELETE.

Triggers

- Triggers are fired when the corresponding triggering event happens on the table.
  - Eg: a user issues an INSERT command
- They are also often used to implement complex auditing, input validation and updating:
  - Eg: to update a stock on hand value when a sale occurs

Trigger Example

```
CREATE OR REPLACE TRIGGER atleast15
AFTER INSERT ON customer FOR EACH ROW
BEGIN
    DECLARE thedob DATE ;
    SET thedob =  new.DOB;
    IF ( datediff( SYSDATE(), thedob ) < ( 15 * 365.25) ) THEN
        #Customer must be at least 15 years old;
        ** MySQL doesn't currently support a 'RAISE ERROR' operation;
    END IF;
END ;
```

Trigger Example

- Insert that worked:

Insert into customer(customerID, dob) values (1, '1990-10-01');

- Insert that failed:

Insert into customer(customerID, dob) values (2, '2009-10-01');