Constraints and Triggers

When creating a table in SQL the keys are identified by declaring one of them to be the PRIMARY KEY and the others to be UNIQUE.

When attributes in one table are meant to refer to a PRIMARY KEY or UNIQUE key in another table, they should be declared to be a FOREIGN KEY. Use REFERENCES clauses:

<attri.> <domain type> REFERENCES <table>(<attr.>)

[above form not recognized by InnoDB]

FOREIGN KEY (<attributes>) REFERENCES <table>(<attributes>) [list after <table> can be omitted if it is the primary key of <table>.]

Maintaining Referential Integrity

Put ON DELETE <option> and ON UPDATE <option> on the end of a REFERENCES clause to specify the policy to be followed to enforce referential integrity. The possible <option> keywords are:

NO ACTION - (the default policy) Do not delete or update a row that is referenced by this foreign key.

CASCADE – If this foreign key references a row and that row gets deleted or updated, delete or update this row too. (Only the attributes in the foreign key get updated.)

SET NULL – If this foreign key references a row and that row gets deleted or updated, set the attributes in this foreign key to NULL. (Error if they can't be NULL.)

Not Null Constraints

When a table is created, an attribute can be declared with the NOT NULL phrase to disallow NULL values.

Modifying Constraints

A constraint can be modified, but only if it has been given a name. CONSTRAINT <name> can be put in front of constraint clauses (PRIMARY KEY, UNIQUE, FOREIGN KEY).

ALTER TABLE can be used to DROP a constraint or ADD a constraint.

ALTER TABLE emp DROP CONSTRAINT fkmanager;

ALTER TABLE emp ADD CONSTRAINT fkproject FOREIGN KEY (project) REFERENCES projects(project_ID);

Triggers

In SQL, triggers are chunks of code that automatically get executed when certain events happen. The code can be run just before the event happens or just after the event happens. The kinds of event that can activate triggers are inserts, deletes and updates.

In MySQL, the delimiter must be changed if more than one semi-colon is needed to define a trigger.

Textbook example in MySQL

```
DELIMITER //
CREATE TRIGGER NetWorthTrigger
AFTER UPDATE ON MovieExec
FOR EACH ROW
IF OLD.netWorth > NEW.netWorth THEN
  SET NEW.netWorth = OLD.netWorth;
END IF; //
DELIMITER ;
```

[No FOR EACH STATEMENT in MySQL.]

3rd Textbook example in MySQL

```
DELIMITER //
CREATE TRIGGER FixYearTrigger
BEFORE INSERT ON Movies
FOR EACH ROW
IF NEW.year IS NULL THEN
   SET NEW.year = 1915;
END IF;//
DELIMITER ;
```


General Trigger Format

CREATE TRIGGER <trigger name>

{ BEFORE | AFTER }

{ INSERT | UPDATE | DELETE }

ON <table name>

FOR EACH ROW

<triggered SQL statement>

Key words allowed in SQL statement: BEGIN, END, IF, THEN, CASE, WHEN, WHILE, LOOP, REPEAT, LEAVE, ITERATE, DECLARE, SET

Programming Statements

These statements are allowed in the body of a trigger or stored program.

BEGIN ... END;

DECLARE var type value; (default) value is optional, must be inside a BEGIN END block

SET var=expr, var2=expr2, ...;

SELECT col1, col2, ... INTO var1, var2, ... FROM ... LIMIT 1; like SELECT statement with INTO phrase added. LIMIT 1 part is optional, but it is an error if SELECT returns more than one row.

Programming Statements cont.

IF ... THEN ... ELSEIF ... THEN ... ELSE ... END IF;

CASE value

   WHEN ... THEN ...

   WHEN ... THEN ...

   ELSE ...

END CASE; The value is optional, but then the WHEN clauses are boolean expressions. The THEN and ELSE clauses can be lists of statements.

Programming Statements cont. 2

label: LOOP ... END LOOP label;

LEAVE label; (exit loop)

ITERATE label; (go back to beginning of loop)

label: REPEAT ... UNTIL ... END REPEAT label;

label: WHILE ... DO ... END WHILE label;

LEAVE and ITERATE work in REPEAT and WHILE too.

Label is optional if LEAVE and ITERATE are not used.

INSERT, DELETE and UPDATE statements are allowed in the body of a trigger.

[triggerdemo.txt, undotriggerdemo.txt]

Cursors

Cursors provide a way to step through a bag of tuples returned by a SELECT statement. The special statements for cursors are

DECLARE cursor_name CURSOR FOR select_statement

OPEN cursor_name

FETCH cursor_name INTO var1, var2, ...

CLOSE cursor_name

(See example in MySQL documentation, Section 12.8.5 Cursors.)