

**Symbiosis Institute of Technology**

**A DBMS Project Report on**

E-Commerce and Stock Management

Submitted by

**Aayushi Verma-18070124003**

**Isha Patil-18070124031**

**Ravneesh Singh-18070124055**

**Tanvi Pathak-18070124072**

Under the Guidance of

**Prof. Shruti Patil**

# Department of Information Technology

**SYMBIOSIS INSTITUTE OF TECHNOLOGY, PUNE**

# Index

|  |  |
| --- | --- |
| 1. Introduction | 3 |
| 1. Problem Statement | 4 |
| 1. Objectives | 5 |
| 1. Functional Requirements | 6 |
| 1. Entities and their relationships. | 7 |
| 1. E-R /EER diagram | 8 |
| 1. Relational schema | 9 |
| 1. Keys | 11 |
| 1. How Codd’s Rule is applied | 13 |
| 1. Kind of anomalies in relational schema |  |
| 1. Functional dependencies of each relation |  |
| 1. Functional dependency charts of each relation |  |
| 1. Normalization of relational schema |  |
| 1. Database Implementation |  |
| 1. Query Execution |  |
| 1. Functions |  |
| 1. Procedures |  |
| 1. Triggers |  |
| 1. Views |  |

Introduction

A database management system refers to the technology for managing huge amount of data for use as and when convenient by the websites or the user. Management of data involves both defining the structure for storage of information as well as manipulation of data for desired results.

This project is for the subject Database Management system. This project is aimed at computerizing the otherwise manual process of stock management and selling. It is an implementation model of a real life E-Commerce and Stock Management database in MySQL. This model is inspired from e commerce giants like Amazon, Flipkart, Nykka et cetera. This structure explores both sides of the e-commerce and stock database.

It includes the inventory database as well as the user database that will be used to place orders online. We have taken into account the inventory side of the business and implemented a structure viable for the appropriate management of available stocks.

Problem Statement

This project aims to use MySQL to solve the problem of making a completely up to date and automated stock management system which has been previously dealt with in a manual way using excel sheets and hard copy files.

Objective

This E Commerce and Stock Management database can be roughly divided into two segments which aims to make the management of inventory to be completely efficient and have a seamless interconnection to the front end of the website and by extension to the user who logs in.

Functional Requirements

Functionalities:

* User can log in/register to the e-commerce website
* User can view the inventory and choose items
* The inventory is divided into categories to make looking for items easier
* Chosen items can be moved to cart for saving the item’s information
* Items from the cart can then be processed
* User can redeem certain offers
* Money transaction occurs between user and the website

1. Can be by Cash
2. Can be by Credit Card

* Alterations to the stock

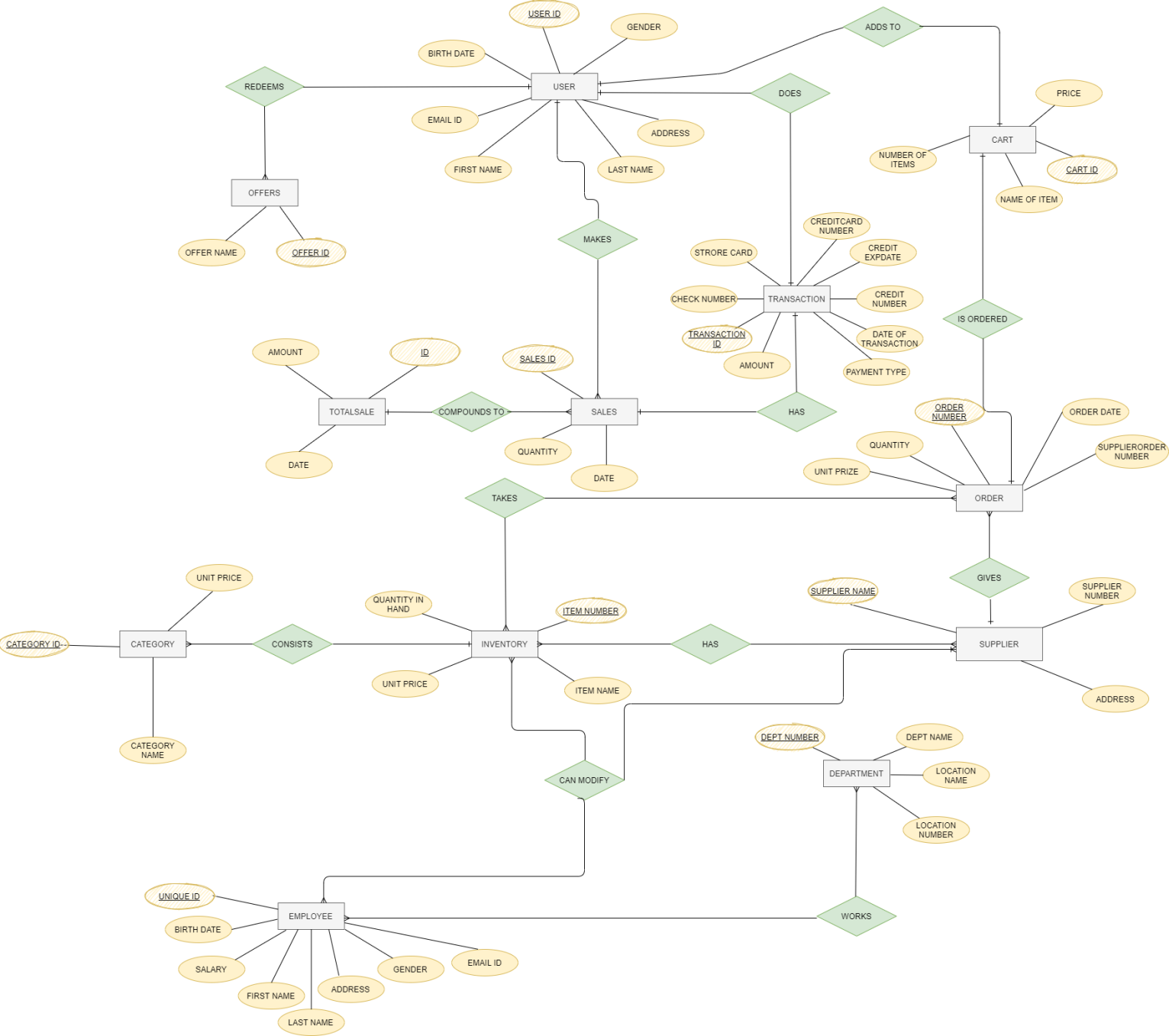
1. New Items can be added
2. Items can be deleted
3. Quantities can be monitored
4. Prices can be adjusted

* Employees can make adjustments to the inventory
* Supplier can add or remove the items from the inventory
* Employee is responsible for monitoring the suppliers
* All sales made are stored in a database for future reference.

Entities and their Relationships

* Employee *works* in Department
* Employee *can modify* Inventory
* Inventory *has* Supplier
* Employee *can modify* Supplier
* Inventory *takes* Order
* Category *consists of* Inventory
* Supplier *can modify* Inventory
* Supplier *gives* Order
* Inventory *takes* Order
* Items are *Ordered by Cart*
* User *adds to* Cart
* User *does* Transaction
* Users *redeems* Order
* User *makes* Sales
* Sales *has* Transaction
* Sales *compounds to* Total Sales

E R diagram



Relational Schema

**TABLE NAME: USER**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| USER ID | GENDER | ADDRESS | LAST NAME | FIRST NAME | EMAIL ID | BIRTH DATE |

**TABLE NAME: CART**

|  |  |  |  |
| --- | --- | --- | --- |
| CART ID | PRICE | NAME OF ITEM | NUMBER OF ITEMS |

**TABLE NAME: TRANSACTION**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| TRANSACTION ID | CREDITCARD NUMBER | CREDIT NUMBER | DATE OF TRANSACTION | PAYMENT TYPE | AMOUNT | STORE CARD | CHECK NUMBER |

**TABLE NAME: ORDER**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ORDER NUMBER | ORDER DATE | SUPPLIERORDER NUMBER | ***SUPPLIER NAME*** | ***ITEM NUMBER*** | UNIT PRIZE | QUANTITY |

**TABLE NAME: SALES**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SALES ID | ***TRANSACTION ID*** | ***ITEM NUMBER*** | DATE | QUANTITY |

**TABLE NAME: OFFERS**

|  |  |
| --- | --- |
| OFFER ID | OFFER NAME |

**TABLE NAME: TOTAL SALE**

|  |  |  |
| --- | --- | --- |
| ID | DATE | AMOUNT |

**TABLE NAME: CATEGORY**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CATEGORY ID | ***ITEM NUMBER*** | UNIT PRICE | CATEGORY NAME | QUANTITY IN HAND |

**TABLE NAME: INVENTORY**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ITEM NUMBER | QUANTITY IN HAND | UNIT PRICE | ***DEPT NUMBER*** | ITEM NAME |

**TABLE NAME: SUPPLIER**

|  |  |  |  |
| --- | --- | --- | --- |
| SUPPLIER NAME | SUPPLIER NUMBER | ADDRESS | ***DEPT NUMBER*** |

**TABLE NAME: DEPARTMENT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DEPT NUMBER | DEPT NAME | LOCATION NAME | LOCATION NUMBER | ***UNQUE ID*** |

**TABLE NAME: EMPLOYEE**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| UNIQUE ID | BIRTH DATE | SALARY | FIRST NAME | LAST NAME | ADDRESS | GENDER | EMAIL ID |

Keys

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TABLE NAME** | **PRIMARY KEY** | **FOREIGN KEY** | **CANDIDATE KEY** | **ALTERNATE KEY** |
| 1. USER | USER\_ID | NULL | 1) USER\_ID,  2) ADDRESS  3) EMAIL | ADDRESS,EMAIL |
| 1. CART | CART\_ID | ITEM\_NUMBER  (TABLE:  INVENTORY) | 1) NAME OF ITEM  2) CART\_ID | NAME OF ITEM |
| 1. OFFERS | OFFER\_ID | NULL | 1) OFFER\_ID  2) OFFER\_NAME | OFFER\_NAME |
| 1. TRANSACTION | TRANSACTION\_ID | NULL | 1)CREDIT\_NUMBER  2)TRANSACTION\_ID | CREDIT\_NUMBER |
| 1. SALES | SALES\_ID | 1)  TRANSACTION\_ID  (TABLE:  TRANSACTION)  2)ITEM\_NUMBER  (TABLE:  INVENTORY) | 1)SALES\_ID  2)DATE | DATE |
| 1. TOTAL SALES | ID | NULL | 1) DATE  2) ID | DATE |
| 1. ORDERS | ORDER NUMBER | SUPPLIER NAME  (TABLE:  SUPPLIER) | 1) ITEM\_NUMBER  2) SUPPLIER\_  NUMBER  3) ORDER\_NUMBER | 1) ITEM\_NUMBER  2) SUPPLIER\_NUMBER |
| 1. INVENTORY | ITEM\_NUMBER | DEPT\_NUMBER  (TABLE:  DEPARTMENT) | 1) ITEM\_NAME  2) ITEM\_NUMBER | ITEM\_NAME |
| 1. CATEGORY | CATEGORY\_ID | ITEM\_NUMBER  (TABLE: INVENTORY) | 1) CATEGORY NAME  2) CATEGORY ID | CATEGORY NAME |
| 1. SUPPLIER | SUPPLIER\_NAME | DEPT\_NUMBER  (TABLE:  DEPARTMENT) | 1) SUPPLIER NUMBER  2) SUPPLIER\_NAME | SUPPLIER NUMBER |
| 1. DEPARTMENT | DEPT\_NUMBER | UNIQUE\_ID  (TABLE:  EMPLOYEE) | 1) LOCATION NUMBER  2)DEPT\_NUMBER  3)DEPT\_NAME | 1)LOCATION NUMBER  2)DEPT\_NAME |
| 1. EMPLOYEE | UNIQUE\_ID | NULL | 1) FIRST\_NAME  2) LAST NAME  3) ADDRESS  4) EMAIL  5)UNIQUE\_ID | 1)FIRST NAME  2)LAST NAME  3)ADDRESS  4)EMAIL |

HOW CODD’s RULES WERE APPLIED:

RULE 1: THE INFORMATION RULE

*All information in the database is to be represented in one and only one way, namely by values in column positions within rows of tables*

This rule indicates that every piece of data that we permanently store in a database is stored in a table. In general, MySQL Server fulfills this rule, because we cannot store any information in anything other than a table.

RULE 2:  THE GUARANTEED ACCESS RULE

*All data must be accessible. This rule is essentially a restatement of the fundamental requirement for* [*primary keys*](http://en.wikipedia.org/wiki/Unique_key)

This rule states that all the data should be accessible and also stresses the importance of primary keys for locating data in the database. The table name locates the correct table, the column name finds the correct column, and the primary key value finds the row containing an individual data item of interest. In other words, each piece of data is accessible by the combination of table name, primary key value, and column name.

Using MySQL, we can search for the primary key value, and once we have the row, the data is accessed via the column name. We can also access data by any of the columns in the table, though we aren’t always guaranteed to receive a single row back.

RULE 3:  SYSTEMATIC TREATMENT OF NULL VALUES

*The DBMS must allow each field to remain null (or empty). Specifically, it must support a representation of "missing information and inapplicable information", distinct from all regular values, and independent of* [*data type*](http://en.wikipedia.org/wiki/Data_type)*. It is also implied that such representations must be manipulated by the DBMS in a systematic way.​*

This rule requires that the RDBMS support a distinct NULL placeholder, regardless of data type. NULLs are distinct from an empty character string or any other number, and they are always to be considered as unknown values. NULLs must propagate through mathematical operations as well as string operations. NULL + = NULL, the logic being that NULL means “unknown.”

MySQL provides several useful functions that handle NULL effectively: [IFNULL](https://www.mysqltutorial.org/mysql-ifnull/), [COALESCE](https://www.mysqltutorial.org/mysql-coalesce/), and [NULLIF](https://www.mysqltutorial.org/mysql-nullif/).

The IFNULL function accepts two parameters. The IFNULL function returns the first argument if it is not NULL, otherwise, it returns the second argument.

RULE 4:  Active [online](http://en.wikipedia.org/wiki/Online) [catalog](http://en.wikipedia.org/wiki/Database_catalog) based on the relational model:​

*The system must support an online, inline, relational* [*catalog*](http://en.wikipedia.org/wiki/Database_catalog) *that is accessible to authorized users by means of their regular* [*query language*](http://en.wikipedia.org/wiki/Query_language)*. That is, users must be able to access the database's structure (catalog) using the same query language that they use to access the database's data.​*

This rule requires that a relational database be self-describing. In other words, the database must contain certain system tables whose columns describe the structure of the database itself, or alternatively, the database description is contained in user-accessible tables.

This is implemented through The INFORMATION\_SCHEMA.  The INFORMATION\_SCHEMA is a schema that has a set of views to look at much of the metadata for the tables, the relationships, the constraints, and even the code in the database.

RULE 5:  THE COMPREHENSIVE DATA SUBLANGUAGE RULE

*A relational system may support several languages and various modes of terminal use. However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and whose ability to support all of the following is comprehensible: a. data definition b. view definition c. data manipulation (interactive and by program) d. integrity constraints e. authorization f. transaction boundaries (begin, commit, and rollback).*

This is implemented using MySQL. This rule mandates the existence of a relational database language, such as MySQL, to manipulate data. MySQL as such isn’t specifically required. The language must be able to support all the central functions of a DBMS: creating a database, retrieving and entering data, implementing database security, and so on.

RULE 6:  THE VIEW UPDATING RULE

*All views that are theoretically updatable must be updatable by the system.​*

This rule deals with views, which are virtual tables used to give various users of a database different views of its structure.

In MySQL, views are not only query-able but also updatable. It means that you can use the [INSERT](https://www.mysqltutorial.org/mysql-insert-statement.aspx) or [UPDATE](https://www.mysqltutorial.org/mysql-update-data.aspx) statement to insert or update rows of the base table through the updatable view. In addition, we can use a DELETE statement to remove rows of the underlying table through the view.

However, to create an updatable [view](https://www.mysqltutorial.org/introduction-sql-views.aspx), the [SELECT statement](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) that defines the view must not contain any of the following elements:

* [Aggregate functions](https://www.mysqltutorial.org/mysql-aggregate-functions.aspx) such as [MIN](https://www.mysqltutorial.org/mysql-min/), [MAX](https://www.mysqltutorial.org/mysql-max-function/), [SUM](https://www.mysqltutorial.org/mysql-sum/), [AVG](https://www.mysqltutorial.org/mysql-avg/), and [COUNT](https://www.mysqltutorial.org/mysql-count/).
* [DISTINCT](https://www.mysqltutorial.org/mysql-distinct.aspx)
* [GROUP BY](https://www.mysqltutorial.org/mysql-group-by.aspx) clause.
* [HAVING](https://www.mysqltutorial.org/mysql-having.aspx) clause.
* [UNION](https://www.mysqltutorial.org/sql-union-mysql.aspx) or UNION ALL clause.
* [Left join](https://www.mysqltutorial.org/mysql-left-join.aspx) or outer join.
* [Subquery](https://www.mysqltutorial.org/mysql-subquery/) in the [SELECT](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) clause or in the [WHERE](https://www.mysqltutorial.org/mysql-where/) clause that refers to the table appeared in the FROM clause.
* Reference to non-updatable view in the FROM clause.
* Reference only to literal values.
* Multiple references to any column of the base table.

RULE 7:   HIGH-LEVEL INSERT, UPDATE AND DELETE:​

*The system must support set-at-a-time insert, update, and delete operators. This means that data can be retrieved from a relational database in sets constructed of data from multiple rows and/or multiple tables. This rule states that insert, update, and delete operations should be supported for any retrievable set rather than just for a single row in a single table.​*

This rule stresses the set-oriented nature of a relational database. It requires that rows be treated as sets in insert, delete, and update operations. The rule is designed to prohibit implementations that support only row-at-a-time, navigational modification of the database. The MySQL language covers this via the INSERT, UPDATE, and DELETE statements

RULE 8: PHYSICAL DATA INDEPENDENCE

*Changes to the physical level (how the data is stored, whether in arrays or linked lists etc.) must not require a change to an application based on the structure.*​

Applications must still work using the same syntax, even when changes are made to the way in which the database internally implements data storage and access methods. This rule implies that the way the data is stored physically must be independent of the logical manner in which it’s accessed. This is saying that users shouldn’t be concerned about how the data is stored or how it’s accessed. In fact, users of the data need only be able to get the basic definition of the data they need. Other things that shouldn’t affect the user’s view of the data are as follows:

• Adding indexes: Indexes determine how the data is stored, yet the user, through MySQL, will never know that indexes are being used.

 • Changing the filegroup of an object: Just moving a table to a new filegroup will not affect the application. You access the data in the same way no matter where it is physically located.

 • Using partitioning: Beyond moving entire tables around to different filegroups, you can move parts of a table around by using partitioning technologies to spread access around to different independent subsystems to enhance performance.

• Modifying the storage engine: From time to time, Oracle has to modify how MySQL Server operates (especially in major version upgrades). However, MySQL statements must appear to access the data in the same manner as they did in any previous version, only (we hope) faster.

RULE 9: LOGICAL DATA INDEPENDENCE

*Changes to the logical level (tables, columns, rows, and so on) must not require a change to an application based on the structure. Logical data independence is more difficult to achieve than physical data independence.*

Along with rule 8, this rule insulates the user or application program from the low-level implementation of the database. Together, they specify that specific access or storage techniques used by the RDBMS—and even changes to the structure of the tables in the database—shouldn’t affect the user’s ability to work with the data

This can be implemented in MySQL using views

RULE 10: INTEGRITY INDEPENDENCE

[*Integrity constraints*](http://en.wikipedia.org/wiki/Integrity_constraints) *must be specified separately from application programs and stored in the* [*catalog*](http://en.wikipedia.org/wiki/Database_catalog)*. It must be possible to change such constraints as and when appropriate without unnecessarily affecting existing applications.​*

The database must support a minimum of the following two integrity constraints:

• Entity integrity: No component of a primary key is allowed to have a NULL value.

 • Referential integrity: For each distinct non-NULL foreign key value in a relational database, there must exist a matching primary key value from the same domain.

This rule says that the database language should support integrity constraints that restrict the data that can be entered into the database and the database modifications that can be made. In other words, the RDBMS must internally support the definition and enforcement of entity integrity (primary keys) and referential integrity (foreign keys). It requires that the database be able to implement constraints to protect the data from invalid values and that careful database design is needed to ensure that referential integrity is achieved.

MySQL Server does a great job of providing the tools to make this rule a reality. We can protect our data from invalid values for most any possible case using constraints and triggers.

RULE 11: DISTRIBUTION INDEPENDENCE

*The data manipulation sublanguage of a relational DBMS must enable application programs and terminal activities to remain logically unimpaired whether and whenever data are physically centralized or distributed.*

 This rule says that the database language must be able to manipulate data located on other computer systems. In essence, we should be able to split the data on the RDBMS out onto multiple physical systems without the user realizing it. MySQL Server supports distributed transactions among MySQL Server sources, as well as other types of sources.

RULE 12: THE NONSUBVERSION RULE

*If the system provides a low-level (record-at-a-time) interface, then that interface cannot be used to subvert the system, for example, bypassing a relational security or integrity constraint.​*

This rule requires that alternate methods of accessing the data are not able to bypass integrity constraints, which means that users can’t violate the rules of the database in any way. For most MySQL servers this rule is followed, because there are no methods of getting to the raw data and changing values other than by the methods prescribed by the database.