Rule 1: The Information Rule

All information in a relational database is represented explicitly at the logical way in exactly one way – by values in tables.

We cannot store information in anything other than tables. Data is retrieved by matching identifiers in relations, using primary keys:

```
SELECT * from patient WHERE patient_id = 1;
```

Rule 2: The Guaranteed Access Rule

Every atomic value in a relational database is guaranteed by logically accessible by resorting to a combination of table name, primary key value and column name.

Field level data can be accessed via the combination of a table name, primary key and column name.

```
SELECT business_name FROM address WHERE address_id = 11;
```

Rule 3: Systematic Treatment of Null Values

NULL values are supported in a relational DBMS for representing missing information and inapplicable information in a systematic way, independent of data type.

If the cell value is unknown or missing or not applicable, it should always be represented as NULL, regardless of the data type.

```
SELECT doctor_id, by_type FROM appointment;
```

doctor_id	by_type
<null>	dropin
1	dropin
1	phone
1	<null>
4	referral
<null>	post
1	post
4	phone
1	<null>
1	<null>
3	phone
1	phone
<null>	email

Rule 4: Active Online Catalog

The database description is represented at the logical level in the same way as ordinary data, so authorized users can apply the same relational language to its interrogation as they apply to regular data.

A relational database should be self-describing. Its structure is stored in up-to-date data dictionary.

```
SELECT *
FROM information_schema.TABLES
WHERE TABLE_SCHEMA = 'G00411287';
```

Output | information_schema.TABLES

7 rows | Tx: Auto | DDL

	TABLE_SCHEMA	TABLE_NAME	CREATE_TIME	ENGINE	TABLE_COLLATION	UPDATE_TIME
1	G00411287	address	2022-04-21 14:24:02	InnoDB	utf8mb4_general_ci	2022-04-21 14:24:02
2	G00411287	appointment	2022-04-21 14:24:02	InnoDB	utf8mb4_general_ci	2022-04-21 14:41:17
3	G00411287	bill	2022-04-21 14:24:02	InnoDB	utf8mb4_general_ci	2022-04-21 14:24:02
4	G00411287	doctor	2022-04-21 14:24:02	InnoDB	utf8mb4_general_ci	2022-04-21 14:24:02
5	G00411287	patient	2022-04-21 14:24:02	InnoDB	utf8mb4_general_ci	2022-04-21 14:24:02
6	G00411287	payment	2022-04-21 14:24:02	InnoDB	utf8mb4_general_ci	2022-04-21 14:24:42
7	G00411287	treatment	2022-04-21 14:24:02	InnoDB	utf8mb4_general_ci	2022-04-21 14:24:02

Rule 5: Comprehensive Data Sub-Language 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 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).

MySQL uses SQL query language for data definition, data manipulation and transaction management.

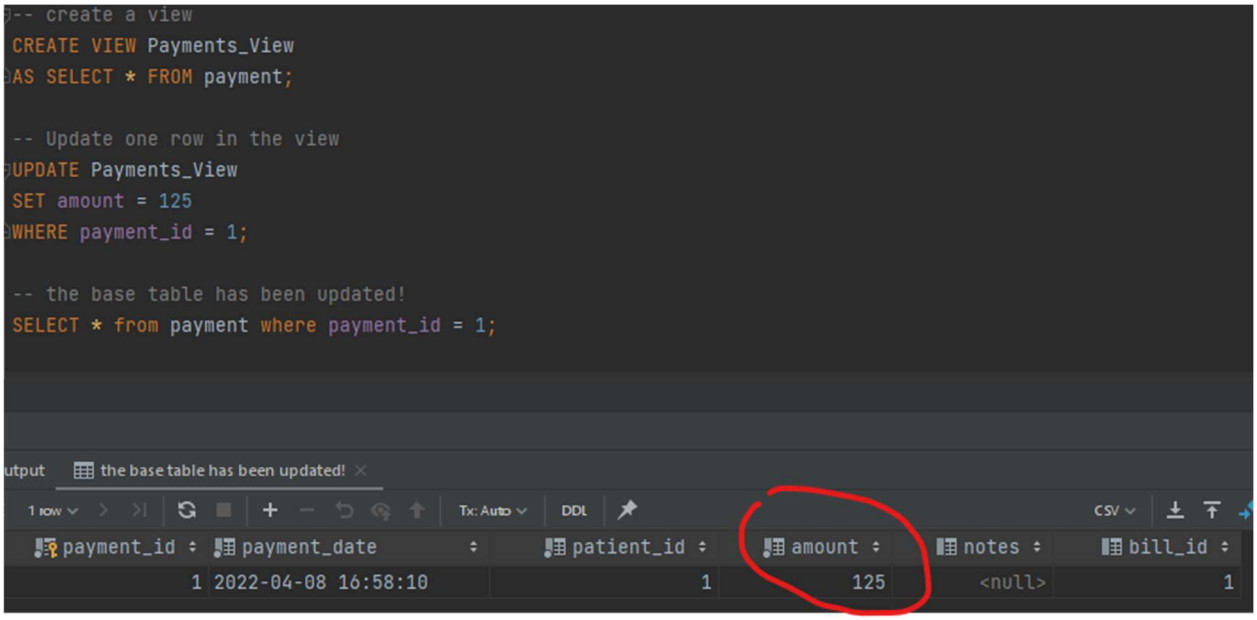
```
CREATE TABLE IF NOT EXISTS example
(
    id          INT AUTO_INCREMENT PRIMARY KEY,
    created_on  DATETIME               NULL,
    description VARCHAR(500)           NULL,
    active      TINYINT(1) DEFAULT 1 NOT NULL
);


INSERT INTO example(created_on, description, active)
VALUES (NOW(), 'Dummy text', 1);

SELECT * FROM example WHERE active = 1;

DROP TABLE IF EXISTS example;
```

Rule 6: View Updating Rule

All the views of a database, which can theoretically be updated, must also be updatable by the system.

Initial value is 1:

Although not recommended, this rule allows to update the records in the view and thus propagate the updates back to the base table:



Rule 7: High-Level Insert, Update, and Delete Rule

A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

```
-- insert multiple rows
INSERT INTO payment (patient_id, bill_id, payment_date, payment_amount,
payment_type, cancellation_fee, notes)
VALUES (1, 1, NOW(), 100.00, 'card', NULL, NULL),
       (2, 2, NOW(), 32.00, 'cheque', 0.0, 'Dummy Notes');

-- hike the 2022 fees (the Water Tiger is roaring!)
UPDATE treatment
SET fees = fees * 1.5
WHERE YEAR(treatment_date) = 2022;
```

Rule 8: Physical Data Independence

Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representation or access methods

External applications or users should not be affected by moving data to another server, storage medium or different OS (depends on the agreed downtime). Moving data to another filegroup/disks, partitioning tables, modifying indexes or the storage engines should not affect the end user.

This query creates a uniqueness index on the email attribute, but the end user is not affected (as long as the table is kept online at the index creation time):

```
CREATE UNIQUE INDEX uq_patient_email ON patient(email);
```

Rule 9: Logical Data Independence

Application programs and terminal activities remain logically unimpaired when information preserving changes of any kind that theoretically permit unimpairment are made to the base tables.

If we add another table or extra columns, it should not impact other tables and existing queries:

```
CREATE TABLE dummytable
(
    id             INT PRIMARY KEY,
    useless_field INT NULL
);

ALTER TABLE payment
    ADD COLUMN no_effect_rule_9 VARCHAR(100) NULL;

-- this query still works as before
SELECT patient_id, payment_amount, payment_date FROM payment;
```

Rule 10: Integrity Independence

A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.

The database should maintain referential integrity and not rely on external applications for integrity constraints. Use non null primary keys, foreign keys, check constraints etc.

Bill ID=100 does not exist, so the operation fails:

```
dbproject> INSERT INTO payment (payment_date, patient_id, amount, notes, bill_id)
           VALUES (NOW(), 1, 123.00, NULL, 100)
[2022-04-08 19:01:45] [23000][1452] Cannot add or update a child row: a foreign key constraint fails (`dbproject`
 .`payment`, CONSTRAINT `payment_ibfk_1` FOREIGN KEY (`bill_id`) REFERENCES `bill` (`bill_id`))
[2022-04-08 19:01:45] [23000][1452] Cannot add or update a child row: a foreign key constraint fails (`dbproject`
 .`payment`, CONSTRAINT `payment_ibfk_1` FOREIGN KEY (`bill_id`) REFERENCES `bill` (`bill_id`))
```

Rule 11: Distribution Independence

A relational DBMS has distribution independence.

The end user should not be aware that the data is distributed over various locations.

For example: we do not know how Azure or AWS store their distributed databases as long as they work.

Queries should work the same whether our data is split/replicated or not. The users get the data as if the records are stored locally.

Rule 12: Non-Subversion Rule

If a relational system has a low-level (single-record-at-a-time) language, that low-level language cannot be used to subvert or bypass the integrity or constraints expressed in the higher-level relational language (multiple-records-at-a-time)

Only SQL code should be able to change data.

Operating system calls and/or low-level languages should not be able to write directly into tables to change data. This is insecure and may break the integrity of the database.