1. Information rule

SQL:

```
SELECT firstName, lastName
FROM STAFF
WHERE employeeID=1;
```

Explanation:

The information is stored as values in the 'STAFF' table. This query shows that you can access the data directly using the attribute names (firstName, lastName). No unnecessary redundant, or alternative ways are needed to get this information.

2. Guaranteed access rule

SQL:

```
SELECT salary
FROM STAFF
WHERE employeeID = 3;
```

Explanation:

This query proves that we can access a specific piece of information (salary) by combining table name, primary key and attribute. Thus, every piece of information in the table can be uniquely identified/retrieved.

3. Systematic treatment of NULL values.

SQL:

```
INSERT INTO ANIMAL(name, age, species, breed, type, ownerID)
VALUES("Billy", 2, "dog", "Yorkshire Terrier", "pet", NULL);
```

Explanation:

This query shows the use of NULL to signify missing data (ownerID). NULL values are supported by MySQL and the RDMS provides a systematic way to handle them using the keywords, IS NULL or IS NOT NULL.

4. Dynamic online catalog.

SQL:

```
SELECT TABLE_NAME, COLUMN_NAME, DATA_TYPE
FROM information_scheme.COLUMNS
WHERE TABLE_SCHEMA = GalwayVet;
```

Explanation:

This query retrieves metadata about the columns in the specified database, GalwayVet. Metadata is stored in tables adhering to the relational model, making it possible to query data using SQL.

5. Comprehensive data sublanguage rule.

SQL:

```
UPDATE STAFF
SET salary = salary * 1.2
WHERE employeeID = 2;
```

Explanation:

This query demonstrates updating data in the STAFF table, proving that SQL queries can be used for different purposes, such as defining, querying, and manipulating data, within the scope of the SQL language.

6. View updating rule.

SQL:

```
CREATE VIEW owner_view AS SELECT firstName, role, phone FROM STAFF;
UPDATE VIEW owner_view SET firstName = 'Dr. Joe' WHERE staffID = 1;
```

Explanation:

Query 1 creates a view showing limited data for the STAFF table to be viewed by animal owners so that they can see first names, roles, and phone numbers but they cannot see data such as salary and last name. Query 2 updates this view, and MySQL allows this if it meets certain conditions, such as being derived from one table which does not contain aggregate data.

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

SQL:

```
DELETE FROM APPOINTMENTS
WHERE appointmentID = 2;
```

Explanation:

This query shows a high-level delete operation. With MySQL, inserting, updating, and deleting data is possible without knowing the low-level implementation details.

8. Physical data independence.

SQL:

```
ALTER TABLE PRESCRIPTION
ENGINE = InnoDB;
```

Explanation:

This query changes the storage engine for the PRESCRIPTION table from the common default of mariaDB to InnoDB. It proves that any change to the physical storage of the DB does not affect the underlying logical structure of the PRESCRIPTION table.

9. Logical data independence.

SQL:

```
ALTER TABLE STAFF
ADD COLUMN staffAddress VARCHAR(255);
```

Explanation:

This query adds a column staffAddress to the STAFF table. This update to the logical structure of the table does not affect any existing applications using the database or the logical integrity of the relationships between any other table.

10. Integrity Independence.

```
SQL: ALTER TABLE STAFF

ADD CONSTRAINT min_salary

CHECK (salary > 0);
```

Explanation:

This query adds a check constraint to the salary attribute for the STAFF column, ensuring that it is never 0 or lower than 0. With MySQL, you can use primary key, foreign key, unique and check constraints independent of each other which are stored in the database's metadata.

11. Distribution independence.

The database system should be able to handle distributed databases and hide the details from users.

Third-party tools can be used with MySQL to create a distributed DB system that hides the practical details of the distribution from users.

12. Non subversion rule.

If the system provides a low-level interface, it should not allow users to bypass the integrity constraints defined at higher levels.

This upheld at the database engine level with MySQL, and it is ensured that even if a user were to interact with data at a lower-level interface, they still cannot bypass the constraints defined by the database designer and/or administrator, thus preserving data integrity.