**Error message**

MySQL error: #1452 - Cannot add or update a child row: a foreign key constraint fails (`database\_name`.`constraint\_name`, CONSTRAINT `constraint\_name` FOREIGN KEY (`column\_name`) REFERENCES `parent\_table` (`column\_name`)).

**Cause**

MySQL is unable to create the Foreign Key as it would violate a referential integrity constraint. This is due to a row existing in the child table which is not in the domain of the parent table.

This error will never occur if you create the Foreign Keys before inserting data.

## Index the columns

Recall that any columns in the child table which are not a Primary Key must be indexed in order to participate in a Foreign Key relationship

(Steps to create a foreign key:

1. Table structure – index the columns
2. Relation view – create the foreign key)

## Modify a parent row in a Foreign Key constraint

Navigate to the astronaut table of the ISS\_2017 database and attempt to edit and delete any row.

Note that you will be unable to perform the action as the Foreign Key ON DELETE and ON UPDATE action are set to RESTRICT, MySQL will throw the error:   
#1451 - Cannot delete or update a parent row: a foreign key constraint fails.

## Setting Foreign Key actions

Whenever a row is updated or deleted in the parent table of a Foreign Key relationship, the DBMS will check the Foreign Key actions. The action decides what will happen to values in the child table which reference the parent table. There are four actions that can be set:

* CASCADE: Delete or update the row from the parent table, and automatically delete or update the matching rows in the child table.
* SET NULL: Delete or update the row from the parent table, and set the foreign key column or columns in the child table to NULL.
* NO ACTION: Equivalent to RESTRICT
* RESTRICT: Rejects the delete or update operation for the parent table.

## Viewing a table definition

The table definition contains general information about the table such as keys, constraints and the domains of columns.

## Step 1: Setting the correct context

Open the database which contains the table you wish to view.

## Step 2: Viewing the definition

The table definition in PHPMyAdmin is referred to as the *Structure*

## Viewing all table definitions

View the table definitions of all tables using the Designer view.

## About SQL query

The SQL menu is available at all context levels. There are two main context levels to consider when opening the SQL menu: the database, and server context level.

## Step 1: Creating a new database

Using SQL, create a new database called protein.

The CREATE DATABASE command creates a new database given the specified name. It is best practice to use a database name with no spaces.

### Syntax:

CREATE DATABASE [database\_name]

### Example:

The following example creates a database called protein:

CREATE DATABASE protein

## Step 2: Creating new tables

Ensure that you are in the context of the protein database prior to executing any SQL statements.

Using SQL, create three new tables from the definitions below.

The CREATE TABLE command creates a new table given in the current database (check the context level). It is best practice to use a table name with no spaces.

### Syntax:

CREATE TABLE [table\_name]   
(   
[column\_name] [data\_type] {NULL | NOT NULL},   
...   
PRIMARY KEY ([column\_name], [column\_name...])   
);

### Example:

The following statement creates a new table called gene\_product

.

CREATE TABLE `gene\_product`   
(   
`protein\_id` VARCHAR(100) NOT NULL ,   
`organism` VARCHAR(255) NOT NULL ,   
`gene` VARCHAR(10) NOT NULL ,   
`protein` VARCHAR(255) NOT NULL ,   
`date\_created` DATETIME NOT NULL ,   
`gene\_function` TEXT NULL ,   
PRIMARY KEY (`protein\_id`)   
);

## Step 3: Altering a table

Using the ALTER TABLE statement, perform the following tasks.

1. Alter the gene\_product table and remove the date\_created column.
2. Alter the ontology table and set the definition column to allow NULL values.

### Syntax:

Adding a column:

ALTER TABLE [table\_name]   
ADD [column\_name] [data\_type] {NULL | NOT NULL};

Dropping a column:

ALTER TABLE [table\_name]   
DROP [column\_name];

Altering a column:

ALTER TABLE [table\_name]   
MODIFY [column\_name] [data\_type] {NULL | NOT NULL};

### Example:

The following example adds the column taxon to the gene\_productdatabase.

ALTER TABLE gene\_product   
ADD taxon VARCHAR(255) NOT NULL;

The following example drops the axon table from the gene\_productdatabase.

ALTER TABLE gene\_product   
DROP taxon;

The following example modifies the gene column of the gene\_producttable, sets the data type to VARCHAR(50) and allows NULL values.

ALTER TABLE gene\_product   
MODIFY gene VARCHAR(50) NULL;

## Step 4: Set the Foreign Keys

Create referential integrity constraints (Foreign Keys) from the definitions below.

Foreign Keys can be defined when issuing a CREATE TABLE statement or appended after the table has been created using the ALTER TABLEstatement.

### Syntax:

ALTER TABLE [table\_name]   
ADD CONSTRAINT [foreign\_key\_name]   
FOREIGN KEY ([child\_column], [...]) REFERENCES [parent\_table] ([parent\_column])   
ON DELETE { RESTRICT | CASCADE | SET NULL } ON UPDATE { RESTRICT | CASCADE | SET NULL };

### Example:

The following example creates a Foreign Key titled FK\_protein\_id on the child table ( gene\_ontology) for the column protein\_id. The Foreign Key references the parent table ( gene\_product). If a row is deleted in the parent table the action will be RESTRICTed and if a row is updated it will CASCADEto the child table.

ALTER TABLE gene\_ontology   
ADD CONSTRAINT FK\_protein\_id   
FOREIGN KEY (protein\_id) REFERENCES gene\_product (protein\_id)   
ON DELETE RESTRICT ON UPDATE CASCADE;

## Step 5: Inserting data

1. To pre-populate the tables, copy and paste the contents of [**protein\_insert.sql**](https://learn.uq.edu.au/bbcswebdav/courses/INFS1200S_6860_60602/practicals/queries_and_views/executing_queries/protein_insert.sql)into the SQL query editor.
2. Manually insert new rows into the gene\_ontology table from the definition below.

|  |  |
| --- | --- |
| **protein\_id** | **ontology\_id** |
| Q92008 | GO:0048663 |
| Q15465 | GO:0048864 |

The INSERT statement requires a destination table and one or more tuples to be inserted. Multiple tuples can be inserted in the same statement by joining them with a comma.

### Syntax:

INSERT INTO [table\_name]   
([column\_1\_name], [column\_2\_name])   
VALUES   
([column\_1\_value], [column\_2\_value]) ,   
([column\_1\_value], [column\_2\_value]);

### Example:

The following example inserts one new tuple into the ontology table.

INSERT INTO `ontology`   
(`ontology\_id`, `role`, `definition`)   
VALUES   
('GO:0090210', 'regulation of blood-brain barrier', 'Any process that modulates the rate...');

## Step 6: Selecting data

Using a SQL SELECT statement, view the contents of the gene\_product, ontology, and gene\_ontology tables.

The basic SELECT statement loads all data from a given table name.

### Syntax:

SELECT \*   
FROM [table\_name];

### Example:

The following example loads all data from the gene\_product table.

SELECT \* FROM gene\_product;

## Step 7: Updating a row

1. Update the ontology table and set the ontology\_id to GO:TEST for the ontology with ontology\_id = GO:0002039.
2. Execute a SELECT statement to view the contents of gene\_ontology, ontology, and ontology\_counts.

#### Explanation:

1. Recall that the Foreign Key action is set to CASCADE any update operation in the parent tables ( ontology and gene\_product) where a row contains a reference to the child table ( gene\_ontology).
2. The ontology\_id value has been updated across all tables and views that reference it.

Note that the UPDATE statement is very similar in structure to a SELECTstatement. An UPDATE statement will modify the value of an existing row, given a filter. If no filter is given the action will be applied to all rows.

### Syntax:

UPDATE [table] SET   
[column\_1\_name] = [column\_1\_value],   
[column\_2\_name] = [column\_2\_value]   
WHERE [column\_name] = [filter\_value];

### Example:

The following UPDATE statement will set the gene to SHHa where the protein\_id matches Q15465.

UPDATE gene\_product SET   
gene = 'SHHa'   
WHERE protein\_id = 'Q15465'

**Views**

## Step 1: Creating views

1. Create a new view called ontology\_counts to save the following query. This view will display the number of times each ontology has been referenced to a protein.
2. Browse the ontology\_counts view by executing a SQL query to return all rows.

SELECT ontology\_id, COUNT(\*) AS ontology\_count   
FROM gene\_ontology   
GROUP BY ontology\_id

The CREATE VIEW statement converts a query into a virtual table. The query used to create the view is essentially stored as a table which returns the same results as the query which was used to create it.

### Syntax:

CREATE VIEW [view\_name] AS   
(   
[SQL Query]   
);

### Example:

The following example creates a new view called orphaned\_ontology which will display all ontologies which are not referenced to a protein.

CREATE VIEW orphaned\_ontology AS (   
SELECT \* FROM ontology   
WHERE ontology\_id NOT IN   
(SELECT ontology\_id FROM gene\_ontology));

## Step 2: Working around MySQL view limitations

There exist limitations with MySQL views which you may encounter. The most likely issue is that subqueries cannot be used in the FROM clause. If it is not possible to rewrite the statement to not use a subquery, then the workaround is to create a view in place of the subquery.

Suppose that you wish to find the average number of times all ontologies are used (i.e. referenced to a protein). Instinctively you may want to try SELECT AVG(COUNT(\*)) however, this functionality does not exist. A SQL query to this solution is as shown below:

SELECT AVG(subquery.ontology\_count)   
FROM   
(SELECT ontology\_id, COUNT(\*) AS ontology\_count   
FROM gene\_ontology   
GROUP BY ontology\_id) AS subquery;

Then click create view, enter the view name 🡪 to create a view

1. Modify the statement shown above with the CREATE VIEW command to create a new view called average\_ontology\_counts.
2. Resolve the issue by converting the subquery into a view. Note, you will have already created the view ontology\_counts.

#### Explanation:

1. You will have received an error, this is a limitation with MySQL views ( #1349 - View's SELECT contains a subquery in the FROM clause).
2. A view can be used in place of a subquery in all SELECT queries. It is essential to working around the view limitation.

CREATE VIEW average\_ontology\_counts AS (   
SELECT AVG(ontology\_count)   
FROM ontology\_counts);

## Step 1: Creating two new databases

Create two new databases called coralbleach and property. If you already have an existing coralbleach database, export and delete it. Otherwise, name the coralbleach database something different.

The CREATE DATABASE command creates a new database. It is best practice to use a database name with no spaces.

### Syntax:

CREATE DATABASE [database\_name];

### Example:

The following statement creates a new database called coralbleach

.

CREATE DATABASE `coralbleach`;

**s**