2018

Documentation of the database structure

FOR SUBTIWIKI

ZHU BINGYAO

TABLE OF CONTENTS

GENERAL DESCRIPTION

The database structure described in this document is used for *Subti*Wiki and a few other derived biological databases (*Listi*Wiki, *Myco*Wiki, etc). This database structure can be applied to other prokaryotic organisms.

The database structure partially relies on native JSON support of MySQL 5.8+. Hence, MySQL 5.8 or higher is recommended. For other database management system which does not support the "JSON" data type. All "JSON" in the document or the SQL dump file should be replaced by text/ mediumtext/ longtext, based on the size of the data.

Flexible table structure

Development of biological databases are more difficult than other type of databases because of the high complexity of biological data. Hence, a flexible and structured layout of the database is preferred. In this database structure, we tried to add more flexibility to relational database by introducing the JavaScript Object Notation (JSON) format.

The idea is simple. In relational model, a table row (tuple) presents an instance of the entity. Instead of putting different attributes of the tuple into different columns, we encode the tuple into a JSON object and store the JSON object in a single column.

id	title	locus	description	product	essential
12	dnaA	BSU00010	DNA replication	replication initiation protein	true

id	title	data

12	dnaA	<pre>{
		"description": "DNA replication",
		"locus": "BSU00010",
		"product": "replication initiation protein",
		"essential": true
		}</pre>

Figure 1 Encoding a conventional table row to a JSON object

In this way, there is more flexibility: each row in the same table does not require to have exactly the same scheme.

Before August 2015, native JSON support is not available in MySQL. We have to store the JSON objects as text. This brings issues with performance. To fix those issues, we included "index columns".

id	title	data
12	dnaA	{ "description": "DNA replication", "locus": "BSU00010", "product": "replication initiation protein", "essential": true }

id	title	_locus	data
12	dnaA	BSU00010	{ "description": "DNA replication", "locus": "BSU00010", "product": "replication initiation protein", "essential": true }

Figure 2 The "index column", whose name begins with an underscore, stores a copy of the values for performance reasons.

As we included the "index column", the table structures become very complicated with the "index column", "JSON column" and conventional columns. Hence, we implemented the PHP libraries to serve as an abstraction layer. This abstraction layer is implemented with three PHP classes: DBBase, ActiveRecord, and DocumentRecord.

The DBBase class is wrap-around of the PDO class provided by PHP. In theory, by using PDO, the framework of *Subti*Wiki should be compatible with other database management system. However, this compatibility has not been tested.

Virtualization

We call the “JSON column” the “virtual column” because with the help of the abstraction layer, those columns are no longer visible to the developers.

id	title	_locus	data
12	dnaA	BSU00010	{ "description": "DNA replication", "locus": "BSU00010", "product": "replication initiation protein", "essential": true }

```
{
    "id": 12,
    "title": "dnaA",
    "description": "DNA replication",
    "locus": "BSU00010",
    "product": "replication initiation protein",
    "essential": true
}
```

Figure 3 The “real” table structure and the “virtual” table structure.

In the abstraction layer, the “index columns” and “JSON column” are removed. Key-value pairs in the “JSON column” is put directly into the object. The abstraction layer is also responsible for the encoding/decoding the JSON object and the update the “index columns” so that this task is out of the concern of developers.

Native JSON support

Since MySQL 5.8+ (currently latest version 8.0), MySQL has included a native JSON support. This change will not affect most of the *Subti*Wiki framework. However, this function will make data import much easier than before.

TABLES

Category

```
CREATE TABLE `Category` (
  `id` varchar(255) NOT NULL,
  `title` varchar(255) DEFAULT NULL,
  `data` json,
  `lastUpdate` timestamp NULL DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP,
  `lastAuthor` varchar(255) NOT NULL DEFAULT 'ghost',
  `count` int(11) DEFAULT NULL,
  `equalTo` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id` (`id`),
  KEY `title` (`title`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

The category table stores a tree-like structure category system used in *Subi*Wiki, *Listi*Wiki and *Myco*Wiki. The content of the table is included in the SQL dump file.

Column	Comment
id	id is in the format as “SW.1.1.2”. The id also indicates the position in the tree
title	We have duplicated categories at different positions of the category tree. Hence, we did not add the unique key on the column. The constraint is maintained on the server-side script level.
equalTo	We have duplicated categories at different positions of the category tree. This column keeps track of the duplicates

Complex

```
CREATE TABLE `Complex` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `title` varchar(255) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `title_UNIQUE` (`title`,`id`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

The Complex table store information about biological complexes. This complex can be protein complex or protein with ligands.

ComplexMember

```
CREATE TABLE `ComplexMember` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `complex` int(11) NOT NULL,
  `member` varchar(255) NOT NULL,
  `description` varchar(255) DEFAULT NULL,
  `coefficient` int(11) NOT NULL DEFAULT '1',
  PRIMARY KEY (`id`),
  KEY `complex` (`complex`),
  CONSTRAINT `fk_ComplexMember_complex` FOREIGN KEY (`complex`) REFERENCES `Complex` (`id`) ON
UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8
```

This table stores information about the composition of a complex.

Column	Comment	foreign key
id	id of the row	
complex	the id of the complex	Complex.id
member	mixed type: could be protein or metabolite An example: {protein| d8ac7fc3d3337863247d96d837ff119a4dd71828} {metabolite|12}	

DataSet

```
CREATE TABLE `DataSet` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `title` varchar(255) NOT NULL,
  `data` text NOT NULL,
  `pubmed` int(10) DEFAULT NULL,
  `type` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

This table is the replacement of the Condition table.

Column	Comments
type	can be **gene-based:** transcriptomics data (fold change) transcriptomics data (raw intensity) proteomics data (copy per cell) proteomics data (existence) **position-based:** tilling array

Gene

```
CREATE TABLE `Gene` (
  `id` char(40) NOT NULL,
  `title` varchar(255) NOT NULL,
  `data` json DEFAULT NULL,
  `_locus` varchar(50) DEFAULT NULL,
  `_function` text,
  `_synonyms` text,
  `_mw` double DEFAULT NULL,
  `_pI` double DEFAULT NULL,
  `_description` text,
  `_essential` varchar(10) DEFAULT NULL,
  `_ec` varchar(30) DEFAULT NULL,
  `_geneLength` int(11) DEFAULT NULL,
  `_proteinLength` int(11) DEFAULT NULL,
  `_strain` varchar(100) NOT NULL,
  `count` int(11) DEFAULT '0',
  `lastUpdate` timestamp NOT NULL DEFAULT '2016-02-24 15:00:00' ON UPDATE CURRENT_TIMESTAMP,
  `lastAuthor` varchar(255) NOT NULL DEFAULT 'ghost',
  PRIMARY KEY (`id`),
  UNIQUE KEY `_locus_3` (`_locus`),
  KEY `_locus` (`_locus`),
  KEY `_mw` (`_mw`),
  KEY `_essential` (`_essential`),
  KEY `_locus_2` (`_locus`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

The Gene table stores information about Genes.

Column	Comments
id	40-character long sha1 hash string
data	virtual column

GeneCategory

```
CREATE TABLE `GeneCategory` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `gene` char(40) DEFAULT NULL,
  `category` varchar(255) DEFAULT NULL,
  `lastAuthor` varchar(255) DEFAULT 'ghost',
  `lastUpdate` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  KEY `idx_category` (`category`),
  KEY `gene` (`gene`),
  KEY `category` (`category`),
  CONSTRAINT `fk_GeneCategory_category` FOREIGN KEY (`category`) REFERENCES `Category` (`id`) ON
UPDATE CASCADE,
  CONSTRAINT `fk_GeneCategory_gene` FOREIGN KEY (`gene`) REFERENCES `Gene` (`id`) ON UPDATE
CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

This table stores all information about the classification of genes in the category system.

Column	Comment	Foreign key
id	id of the row	
gene	id of the gene	Gene.id
category	id of the category	Category.id

GenomicContext

```
CREATE TABLE `GenomicContext` (
  `start` int(11) DEFAULT NULL,
  `stop` int(11) DEFAULT NULL,
  `object` varchar(255) DEFAULT NULL,
  `strand` int(1) DEFAULT NULL,
  `strain` text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

The GenomicContext table stores all information about genomic context, including genes, transcription start sites (TSS), transcription upshit, or transcription downshift. The data stored in this table will be fed to the context browser on gene pages and expression browser page.

Column	Comment
start	start position on the genome
stop	stop postion on the genome. start should always be smaller than the stop
strand	stranf of the object, 1 for plus strand and 0 for the minus strand
object	the object, which could be genes/TSS/upshifts/downshifts for genes, a markup is used such as {gene|the_id_of_the_gene} An example: {gene|cd6a26b15ba063d40a274b4db7ee3cb4a1a4f9ca} TSS
strain	If multiple information about multiple strains exists in the system.

Screen shot (from *Subti*Wiki)

start	stop	object	strand
3545889	3545964	{gene|D6AC23EBAA03F4B905B0525F44E8F0DD6F82E9C7}	0
3173722	3173797	{gene|E8D13401B55A07EDB8E446175BA040AEB00ACB48}	0
2331330	2331715	{gene|57A79AAF00243B5E058FA901D43AA7B55CA33F63}	0
2814490	2814682	{gene|A0E4AC7E74B5374198D047589629DA96A681CAF4}	0
2095927	2096113	{gene|E85341621B3835F63F07C506E08B73A92E588B1F}	0
3450710	3451071	{gene|5C12BB19B975F7FEBCFA119DAC66338C20FCECFA}	0
150	150	upshift	1
297	297	upshift	0
1066	1066	upshift	0
3169	3169	upshift	1

History

```
CREATE TABLE `History` (
  `commit` char(16) NOT NULL,
  `origin` varchar(255) DEFAULT NULL,
  `identifier` varchar(255) DEFAULT NULL,
  `record` json DEFAULT NULL,
  `user` varchar(255) DEFAULT NULL,
  `lastOperation` varchar(255) DEFAULT NULL,
  `time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`commit`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

This table stores different versions of the record from all database.

Column	Comment
commit	a 16-character long random url-safe string
origin	the table where the record come from. The first letter is changed to lower case.
identifier	the id of the record in the origin table
record	the full record in JSON format
user	user name
lastOperation	“add”, “remove”, “update”
time	time of the record change

Interaction

```
CREATE TABLE `Interaction` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `prot1` varchar(255) NOT NULL,
  `prot2` varchar(255) NOT NULL,
  `data` JSON,
  `lastUpdate` timestamp NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  `lastAuthor` varchar(255) NOT NULL DEFAULT 'ghost',
  PRIMARY KEY (`id`),
  UNIQUE KEY `mark` (`id`),
  UNIQUE KEY `duplication_constraint` (`prot1`,`prot2`),
  KEY `fk_Interaction_prot2` (`prot2`),
  CONSTRAINT `fk_Interaction_prot1` FOREIGN KEY (`prot1`) REFERENCES `Gene` (`id`) ON UPDATE
CASCADE,
  CONSTRAINT `fk_Interaction_prot2` FOREIGN KEY (`prot2`) REFERENCES `Gene` (`id`) ON UPDATE
CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8
```

This table stores all information about protein-protein interaction

Column	Comment	Foreign key
id	id of the row	
prot1	id of the first protein	Gene.id
prot2	id of the second protein	Gene.id
data	virtual column	

Important: prot1 should always be smaller than prot2. This is for the unique key constraint to work on symmetric relationships.

MaterialViewGeneRegulation

```
CREATE TABLE `MaterialViewGeneRegulation` (
  `gene` char(40) NOT NULL,
  `regulation` int(11) NOT NULL DEFAULT '0',
  UNIQUE KEY `gene` (`gene`,`regulation`),
  KEY `fk_MaterialViewGeneRegulation_regulation` (`regulation`),
  CONSTRAINT `fk_MaterialViewGeneRegulation_gene` FOREIGN KEY (`gene`) REFERENCES `Gene` (`id`)
ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `fk_MaterialViewGeneRegulation_regulation` FOREIGN KEY (`regulation`) REFERENCES
`Regulation` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

In *Subti*Wiki, regulation is modelled to two types

1. transcriptional regulation, in which a regulator can affect a whole operon
2. other regulation: in which a regulator can only affect a gene

This table exists to improve the performance of the database, as recursive query is not supported in MySQL and even if it is supported, the performance is poor.

This is table maintained by triggers. Creating SQL is:

```
INSERT INTO `MaterialViewGeneRegulon`
  SELECT `Gene`.`id`, `Regulation`,`id` FROM `Gene`
    JOIN `ViewGeneOperon` on `ViewGeneOperon`.`gene` = `Gene`.`id`
    JOIN `Regulation` on `Regulation`.`regulated` = CONCAT("{operon|", `ViewGeneOperon`.`operon`, "}")
  WHERE `Regulation`.`regulated` like "{operon|%}";
INSERT INTO `MaterialViewGeneRegulation`
  SELECT SUBSTR(`regulated`, 7, 40), `id` FROM `Regulation` WHERE `Regulated` like "{gene|%}";
```

Triggers for maintenance:

```
DELIMITER $$

-- after operon deleted, update the MaterialView by deleting relevant rows
CREATE TRIGGER `Operon_after_delete` AFTER DELETE ON `Operon`
 FOR EACH ROW
  BEGIN
   if @triggerEnabled THEN
    delete MaterialViewGeneRegulation from MaterialViewGeneRegulation join Regulation on Regulation.id =
MaterialViewGeneRegulation.regulation where Regulation.regulated like concat("{operon|", old.id, "}");
   end IF
  END$$

-- after operon update (genes could be changed), update the MaterialView by deleting the relavant rows and re-
insert the updated ones
CREATE TRIGGER `Operon_after_update` AFTER UPDATE ON `Operon`
 FOR EACH ROW begin
  if @triggerEnabled then
   delete MaterialViewGeneRegulation from MaterialViewGeneRegulation join Regulation on Regulation.id =
MaterialViewGeneRegulation.regulation where Regulation.regulated = concat("{operon|", new.id, "}");
   insert ignore into MaterialViewGeneRegulation (gene, regulation) select gene, id from ViewGeneOperon join
Regulation on Regulation.regulated = concat("{operon|", ViewGeneOperon.operon, "}") where operon = new.id;
  end if;
end$$

-- after insertion of a new regulation, insert the relevant rows
CREATE TRIGGER `Regulation_after_insert` AFTER INSERT ON `Regulation`
 FOR EACH ROW begin
  if new.regulated like "{operon|%}" and @triggerEnabled then
   insert into MaterialViewGeneRegulation (gene, regulation) select gene, new.id from ViewGeneOperon where
new.regulated like concat("{operon|", operon, "}");
  end if;
  if new.regulated like "{gene|%}" and @triggerEnabled then
   insert into MaterialViewGeneRegulation (gene, regulation) select substr(regulated, 7, 40), id from Regulation;
END$$

DELIMITER ;
```

MetaData

```
CREATE TABLE `MetaData` (
  `className` varchar(255) NOT NULL,
  `scheme` json NOT NULL,
  PRIMARY KEY (`className`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

In *Subti*Wiki database, some of the tables has a “flex table structure”, which means the table has a virtual column (named “data” and has “JSON” datatype).

In MySQL, the JSON datatype is unordered and MySQL hashes the keys for better performance. This is a problem for us. *Subti*Wiki is designed to be “you see what you input” and the orders of the keys are important. Hence, this table exists to collect all the keys and preserve the key orders of objects of different tables.

Metabolite

```
CREATE TABLE `Metabolite` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `title` varchar(255) CHARACTER SET latin1 DEFAULT NULL,
  `synonym` varchar(255) CHARACTER SET latin1 DEFAULT NULL,
  `pubchem` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `title_UNIQUE` (`title`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

This table collects all data about metabolites

Column	Comment
id	id of the metabolite
title	name of the metabolite
synonym	synonym of the metabolite
pubchem	pubchem id of the metabolite, the chemical formula images shown in the pathway are from PubChem

Operon

```
CREATE TABLE `Operon` (
  `id` char(40) NOT NULL,
  `title` text,
  `data` json DEFAULT NULL,
  `_genes` longtext,
  `lastUpdate` timestamp NULL DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP,
  `lastAuthor` varchar(255) NOT NULL DEFAULT 'ghost',
  `count` int(11) NOT NULL DEFAULT '0',
  `hash` char(40) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `hash` (`hash`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

The Operon table stores all the information about operons.

Column	Comments
id	a 40-character long sha1 string
data	virtual column
_genes	corresponds to the key “genes” in the JSON object stored in data column An example: [[gene|d8ac7fc3d3337863247d96d837ff119a4dd71828]]-[[gene|b2b2809ab3b7d23a1781355364b24c642b6cf5f2]]
hash	A sha1 hash value of the _genes column. A hash is used here to keep the _genes column unique. However, a unique constraint cannot be added to columns with the datatype “longtext”.

OmicsData_gene

```
CREATE TABLE `OmicsData_gene` (
  `gene` char(40) NOT NULL,
  `dataSet` int(11) NOT NULL,
  `value` double DEFAULT NULL,
  UNIQUE KEY `gene_2` (`gene`,`dataSet`),
  KEY `gene` (`gene`),
  KEY `dataSet` (`dataSet`),
  CONSTRAINT `fk_omics_dataset` FOREIGN KEY (`dataSet`) REFERENCES `DataSet` (`id`) ON UPDATE
CASCADE,
  CONSTRAINT `fk_omics_gene` FOREIGN KEY (`gene`) REFERENCES `Gene` (`id`) ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

This table stores gene-based expression data (transcriptomics and proteomics)

Column	Comments	Foreign key
gene	id of the gene	Gene.id
dataset	id of the data set	DataSet.id

OmicsData_position

```
CREATE TABLE `OmicsData_position`(
   `position` int(11) NOT NULL,
   `strand` int(1) NOT NULL,
   `dataSet` int(11) NOT NULL,
   `value` double DEFAULT NULL,
   UNIQUE KEY `gene_2` (`position`,`dataSet`, `strand`),
   KEY `strand` (`strand`),
   KEY `position` (`position`),
   KEY `dataSet` (`dataSet`),
   CONSTRAINT `fk_omics_pos_dataset` FOREIGN KEY (`dataSet`) REFERENCES `DataSet` (`id`) ON
UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

This table stores position-based expression data.

Column	Comments	Foreign key
position	position on the genome	
strand	1 for plus, 0 for minus	
dataSet	id of the data set	DataSet.id

ParalogousProtein

```
CREATE TABLE `ParalogousProtein` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `prot1` char(40) DEFAULT NULL,
  `prot2` char(40) DEFAULT NULL,
  `data` mediumtext,
  `strain` text,
  `lastUpdate` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  `lastAuthor` varchar(255) NOT NULL DEFAULT 'ghost',
  PRIMARY KEY (`id`),
  UNIQUE KEY `prot1_2` (`prot1`,`prot2`),
  KEY `index_prot1` (`prot1`),
  KEY `index_prot2` (`prot2`),
  KEY `prot1` (`prot1`,`prot2`) USING BTREE,
  CONSTRAINT `fk_ParalogousProtein_prot1` FOREIGN KEY (`prot1`) REFERENCES `Gene` (`id`) ON
UPDATE CASCADE,
  CONSTRAINT `fk_ParalogousProtein_prot2` FOREIGN KEY (`prot2`) REFERENCES `Gene` (`id`) ON
UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8
```

This table collects information about paralogous proteins in the organism.

Column	Comment	Foreign key
id	id of the row	
prot1	id of the first protein	Gene.id
prot2	id of the second protein	Gene.id
data	virtual column	

Important: prot1 should always be smaller than prot2. This is for the unique key constraint to work on symmetric relationships.

Pathway

```
CREATE TABLE `Pathway` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `title` varchar(255) NOT NULL,
  `map` longtext,
  PRIMARY KEY (`id`),
  UNIQUE KEY `title` (`title`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

This table stores all the information about different pathways.

Column	Comment
id	id of the pathway
title	name of the pathway
map	the svg file for pathway presentation

Pubmed

```
CREATE TABLE `Pubmed` (
  `id` int(11) NOT NULL DEFAULT '0',
  `report` text,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

This table stores the cache of pubmed citations for papers used in the website as references.

Reaction

```
CREATE TABLE `Reaction` (
  `pathway` int(11) DEFAULT NULL,
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `reversible` int(1) DEFAULT NULL,
  `equation` text,
  `comment` text,
  `lastAuthor` varchar(255) NOT NULL DEFAULT 'ghost',
  `lastUpdate` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `KEGG` varchar(20) DEFAULT NULL,
  `EC` varchar(255) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `pathway` (`pathway`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

This table stores information about the biochemical reactions.

Column	Comment
pathway	id of the pathway
id	id of the reaction
reversible	whether the reaction is reversible or not
equation	the equation of the reaction. In *Subti*Wiki, this column is automatically updated.
KEGG	id of the reaction in KEGG database
EC	the EC number of the reaction

ReactionCatalyst

```
CREATE TABLE `ReactionCatalyst` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `reaction` int(11) NOT NULL,
  `catalyst` varchar(255) DEFAULT NULL,
  `modification` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `ReactionCatalystUnique` (`reaction`,`catalyst`,`modification`,`position`) USING BTREE,
  KEY `fk_Reaction_enzyme_1_idx` (`reaction`),
  CONSTRAINT `fk_ReactionCatalyst_reaction` FOREIGN KEY (`reaction`) REFERENCES `Reaction` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8
```

This table stores information of the catalysts of biochemical reactions.

Column	Comment	Foreign key
id	id of the row	
reaction	id of the reaction	Reaction.id
catalyst	mixed type, can be a protein or a complex An example: {protein| d8ac7fc3d3337863247d96d837ff119a4dd71828} {complex|12}	
modification	phosphorylation, etc	

ReactionMetabolite

```
CREATE TABLE `ReactionMetabolite` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `reaction` int(11) NOT NULL,
  `coefficient` int(11) NOT NULL,
  `metabolite` int(11) NOT NULL,
  `side` varchar(1) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `ReactionChemicalUnique` (`reaction`,`metabolite`,`side`),
  KEY `idx_1` (`reaction`),
  KEY `idx2` (`metabolite`),
  CONSTRAINT `fk_ReactionMetabolite_1` FOREIGN KEY (`metabolite`) REFERENCES `Metabolite` (`id`) ON
UPDATE CASCADE,
  CONSTRAINT `fk_ReactionMetabolite_2` FOREIGN KEY (`reaction`) REFERENCES `Reaction` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

This table stores information about metabolites in the biochemical reactions.

Column	Comment	Foreign key
id	id of the row	
reaction	id of the reaction	Reaction.id
coefficient	coefficient of the metabolite in the reaction	
metabolite	id of the metabolite	Metabolite.id
side	L or R	

ReactionPathway

```
CREATE TABLE `ReactionPathway` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `reaction` int(11) NOT NULL,
  `pathway` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `reaction` (`reaction`),
  KEY `pathway` (`pathway`),
  CONSTRAINT `fk_ReactionPathway_reaction` FOREIGN KEY (`reaction`) REFERENCES `Reaction` (`id`) ON
UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT `fk_ReactionPathway_pathway` FOREIGN KEY (`pathway`) REFERENCES `Pathway` (`id`) ON
UPDATE CASCADE ON DELETE CASCADE,
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

Regulation

```
CREATE TABLE `Regulation` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `regulator` varchar(255) DEFAULT NULL,
  `regulated` varchar(255) DEFAULT NULL,
  `mode` varchar(255) DEFAULT NULL,
  `description` text,
  `lastAuthor` varchar(255) NOT NULL DEFAULT 'ghost',
  `lastUpdate` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  UNIQUE KEY `duplication_constraint` (`regulator`,`regulated`),
  KEY `index2` (`regulator`,`regulated`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

This table stores information about gene/operon regulations.

In *Subti*Wiki, regulation is modelled to two types

1. transcriptional regulation, in which a regulator can affect a whole operon
2. other regulation: in which a regulator can only affect a gene

Column	Comment
id	id of the regulation
regulator	mixed type, can be a protein or riboswitch An example: {protein| d8ac7fc3d3337863247d96d837ff119a4dd71828} {riboswitch|T-box}
regulated	mixed type, can be a gene or an operon An example: {operon | d8ac7fc3d3337863247d96d837ff119a4dd71828} {operon|45B867D63B0D40557C698DD1870A35EA13FC36C6}

Regulon

```
CREATE TABLE `Regulon` (
  `id` varchar(255) NOT NULL,
  `data` json,
  `lastUpdate` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  `lastAuthor` varchar(255) NOT NULL DEFAULT 'ghost',
  `count` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

This table stores additional information about the regulon.

Column Name	Comment

id	In the format as type:regulator_id
	An example:
	`| protein:FCDE900167AE36377979E0CF7BC33C7F351B95D3 |`
	`| protein:FD4E988BB73F6476CCEA4C521F343D6F8CBA457F |`
	`| riboswitch:A-box |`
	`| riboswitch:B12 riboswitch |`
	`| riboswitch:EAR riboswitch |`
	`| riboswitch:FMN-box |`

data	virtual column

Sequence

```
CREATE TABLE `Sequence` (
  `gene` char(40) NOT NULL,
  `dna` mediumtext,
  `aminos` mediumtext,
  `strain` text NOT NULL,
  PRIMARY KEY (`gene`),
  UNIQUE KEY `gene` (`gene`),
  CONSTRAINT `fk_Sequence_gene` FOREIGN KEY (`gene`) REFERENCES `Gene` (`id`) ON DELETE
CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

This table stores the sequences of genes/proteins.

Statistics

```
CREATE TABLE `Statistics` (
  `item` varchar(255) NOT NULL,
  `count` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`item`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

This table stores statistics of the website usage. Default items include:

Item
categoryExport
expressionBrowser
geneCategoryExport
geneExport
genomeBrowser
index
interactionBrowser
interactionExport
operonExport
pathwayBrowser
regualtionBrowser
regulationExport
statistics

User

```
CREATE TABLE `User` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varbinary(255) NOT NULL,
  `realName` varchar(255) DEFAULT NULL,
  `password` varbinary(255) NOT NULL,
  `email` varchar(255) DEFAULT NULL,
  `registration` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `privilege` int(11) NOT NULL DEFAULT '1',
  `token` binary(32) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `name` (`name`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

This table stores user account information.

Column	Comment
password	hashed password This column uses the encryption from MediaWiki engine. see https://www.mediawiki.org/wiki/Manual:User_table
token	user token, for Cookie-based register reserved
privilege	could be 1,2,3

Privilege	Functions
1	edit content
2	edit/delete content (except for Gene) invite user change user privileges
3	edit/delete content (including Gene, requires extra password) edit templates import data invite user

ViewGeneOperon

```
CREATE VIEW `ViewGeneOperon`
  SELECT `Gene`.`id` as `gene`, `Operon`.`id` as `operon` WHERE `Operon`.`_genes` like CONCAT("%",
`Gene`.`id`, "%");
```

Wiki

```
CREATE TABLE `Wiki` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `title` varchar(255) NOT NULL,
  `article` longtext,
  `lastAuthor` varchar(255) NOT NULL DEFAULT 'ghost',
  `lastUpdate` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  UNIQUE KEY `title` (`title`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

This table is a supplement function of old wiki functions in *Subti*Wiki.