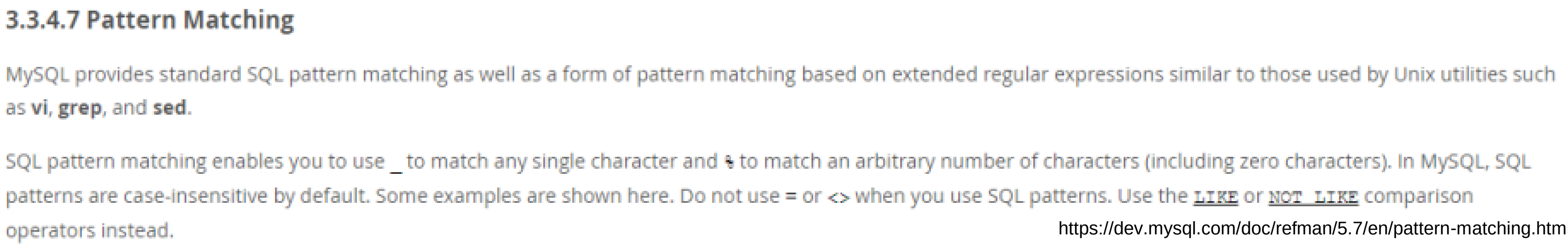
OREGON HEALTH & SCIENCE UNIVERSITY

OHSU

Additional Relational Database Features

BMI 535/635

SQL: Pattern Matching

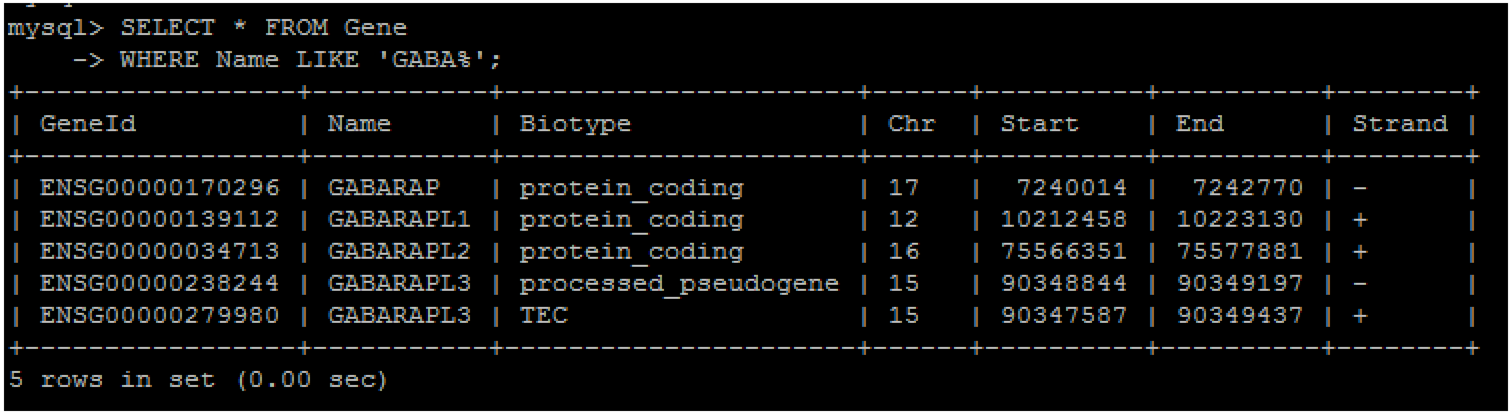


3.3.4.7 Pattern Matching

MySQL provides standard SQL pattern matching as well as a form of pattern matching based on extended regular expressions similar to those used by Unix utilities such as **vi**, **grep**, and **sed**.

SQL pattern matching enables you to use _ to match any single character and % to match an arbitrary number of characters (including zero characters). In MySQL, SQL patterns are case-insensitive by default. Some examples are shown here. Do not use = or <> when you use SQL patterns. Use the LIKE or NOT LIKE comparison operators instead.

https://dev.mysql.com/doc/refman/5.7/en/pattern-matching.html



```
mysql> SELECT * FROM Gene
    -> WHERE Name LIKE 'GABA%';
+-----------------+-----------+----------------------+------+----------+----------+--------+
| GeneId          | Name      | Biotype              | Chr  | Start    | End      | Strand |
+-----------------+-----------+----------------------+------+----------+----------+--------+
| ENSG00000170296 | GABARAP   | protein_coding       | 17   |  7240014 |  7242770 | -      |
| ENSG00000139112 | GABARAPL1 | protein_coding       | 12   | 10212458 | 10223130 | +      |
| ENSG00000034713 | GABARAPL2 | protein_coding       | 16   | 75566351 | 75577881 | +      |
| ENSG00000238244 | GABARAPL3 | processed_pseudogene | 15   | 90348844 | 90349197 | -      |
| ENSG00000279980 | GABARAPL3 | TEC                  | 15   | 90347587 | 90349437 | +      |
+-----------------+-----------+----------------------+------+----------+----------+--------+
5 rows in set (0.00 sec)
```

SQL: Comparison Operators

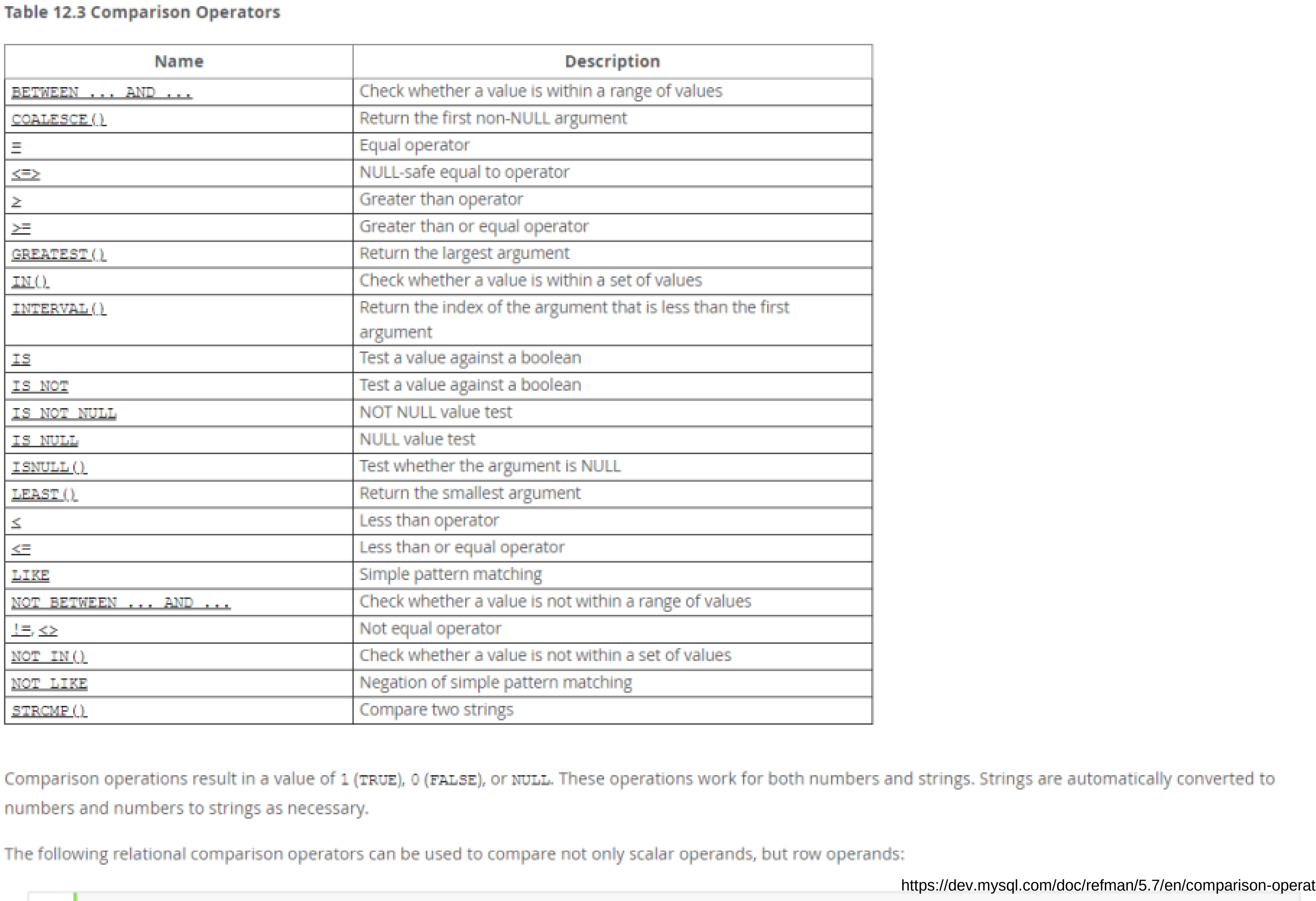


Table 12.3 Comparison Operators

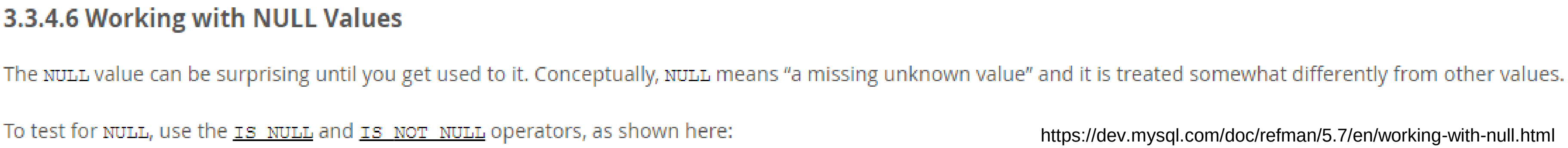
Name	Description
`BETWEEN ... AND ...`	Check whether a value is within a range of values
`COALESCE()`	Return the first non-NULL argument
`=`	Equal operator
`<=>`	NULL-safe equal to operator
`>`	Greater than operator
`>=`	Greater than or equal operator
`GREATEST()`	Return the largest argument
`IN()`	Check whether a value is within a set of values
`INTERVAL()`	Return the index of the argument that is less than the first argument
`IS`	Test a value against a boolean
`IS NOT`	Test a value against a boolean
`IS NOT NULL`	NOT NULL value test
`IS NULL`	NULL value test
`ISNULL()`	Test whether the argument is NULL
`LEAST()`	Return the smallest argument
`<`	Less than operator
`<=`	Less than or equal operator
`LIKE`	Simple pattern matching
`NOT BETWEEN ... AND ...`	Check whether a value is not within a range of values
`!=`, `<>`	Not equal operator
`NOT IN()`	Check whether a value is not within a set of values
`NOT LIKE`	Negation of simple pattern matching
`STRCMP()`	Compare two strings

Comparison operations result in a value of `1` (`TRUE`), `0` (`FALSE`), or `NULL`. These operations work for both numbers and strings. Strings are automatically converted to numbers and numbers to strings as necessary.

The following relational comparison operators can be used to compare not only scalar operands, but row operands:

https://dev.mysql.com/doc/refman/5.7/en/comparison-operators.html

SQL: NULL Values

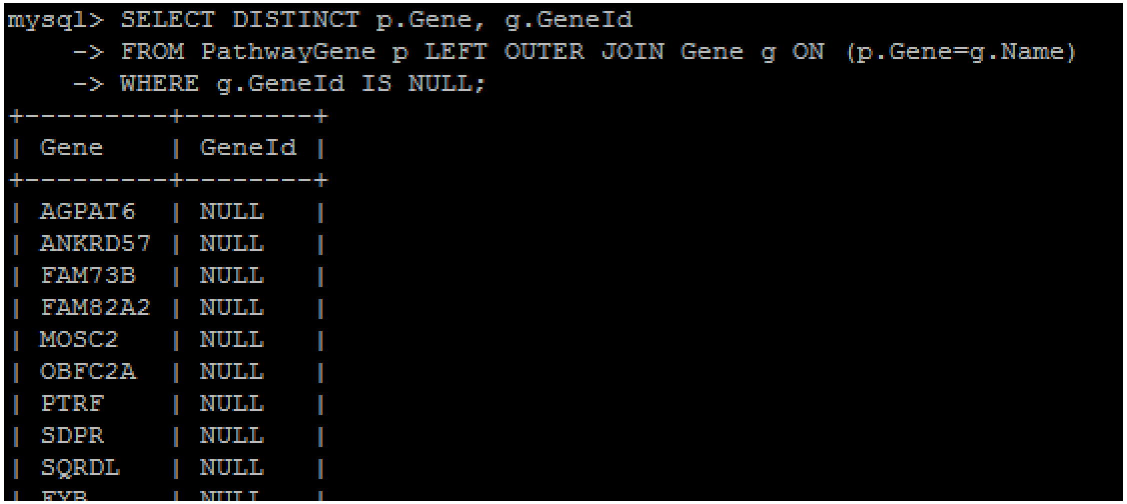


3.3.4.6 Working with NULL Values

The `NULL` value can be surprising until you get used to it. Conceptually, `NULL` means “a missing unknown value” and it is treated somewhat differently from other values.

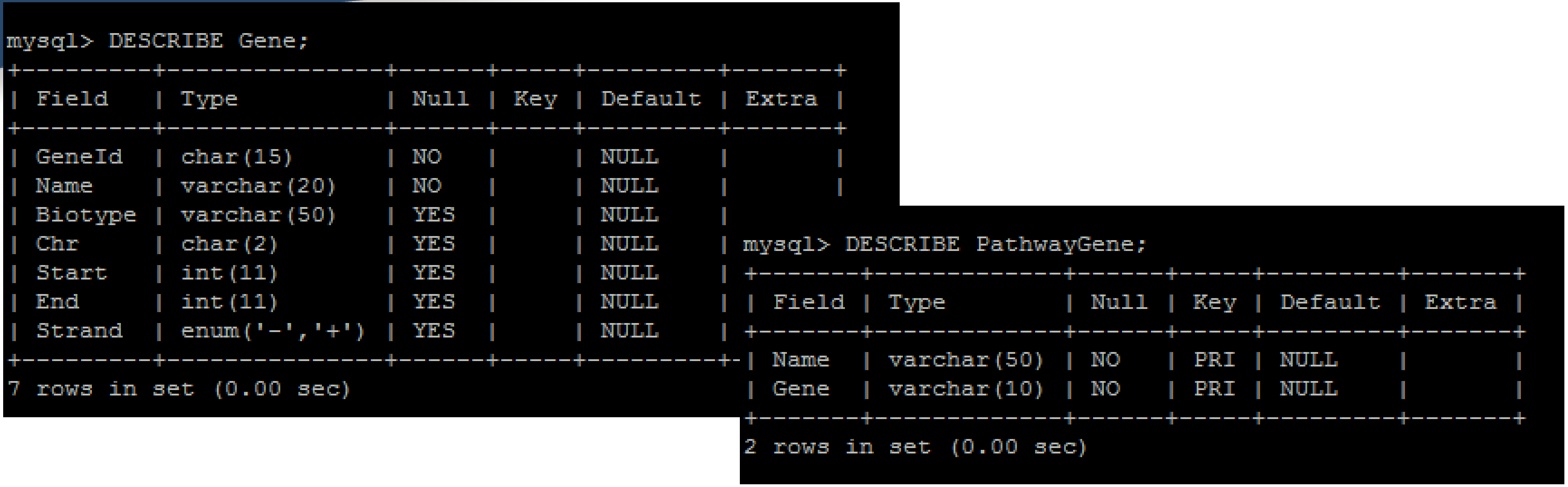
To test for `NULL`, use the `IS NULL` and `IS NOT NULL` operators, as shown here:

https://dev.mysql.com/doc/refman/5.7/en/working-with-null.html



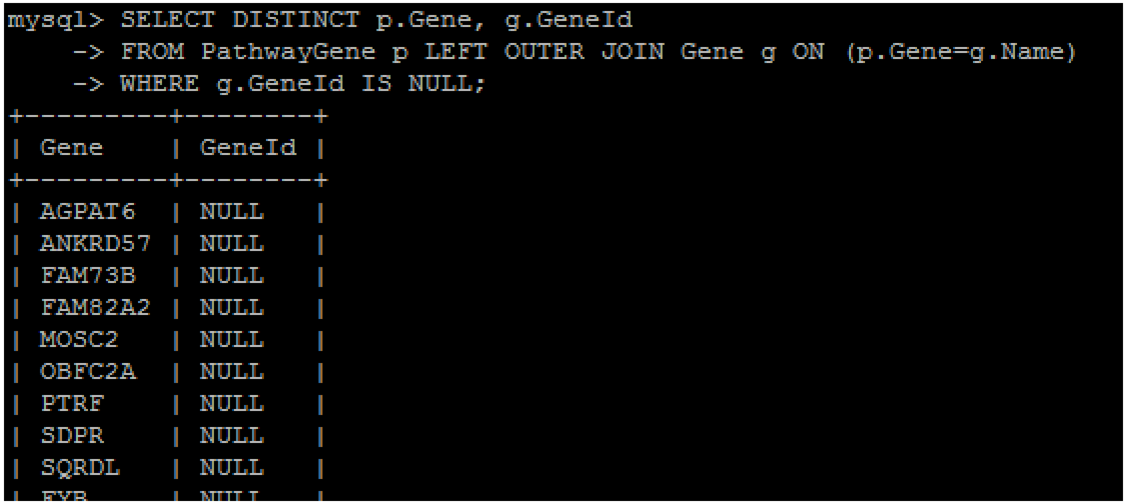
```
mysql> SELECT DISTINCT p.Gene, g.GeneId
    -> FROM PathwayGene p LEFT OUTER JOIN Gene g ON (p.Gene=g.Name)
    -> WHERE g.GeneId IS NULL;
+---------+--------+
| Gene    | GeneId |
+---------+--------+
| AGPAT6  | NULL   |
| ANKRD57 | NULL   |
| FAM73B  | NULL   |
| FAM82A2 | NULL   |
| MOSC2   | NULL   |
| OBFC2A  | NULL   |
| PTRF    | NULL   |
| SDPR    | NULL   |
| SQRDL   | NULL   |
| FYB     | NULL   |
```

Revisiting Pathway Table



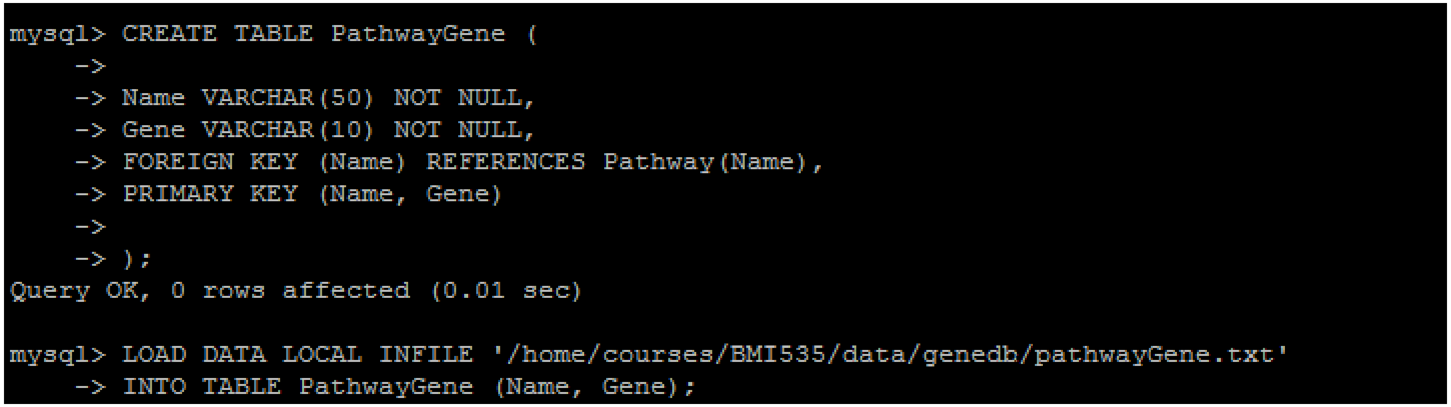
```
mysql> DESCRIBE Gene;
+---------+---------------+------+-----+---------+-------+
| Field   | Type          | Null | Key | Default | Extra |
+---------+---------------+------+-----+---------+-------+
| GeneId  | char(15)      | NO   |     | NULL    |       |
| Name    | varchar(20)   | NO   |     | NULL    |       |
| Biotype | varchar(50)   | YES  |     | NULL    |
| Chr     | char(2)       | YES  |     | NULL    |
| Start   | int(11)       | YES  |     | NULL    |
| End     | int(11)       | YES  |     | NULL    |
| Strand  | enum('-','+') | YES  |     | NULL    |
+---------+---------------+------+-----+---------+-
7 rows in set (0.00 sec)
```

```
mysql> DESCRIBE PathwayGene;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| Name  | varchar(50) | NO   | PRI | NULL    |       |
| Gene  | varchar(10) | NO   | PRI | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```



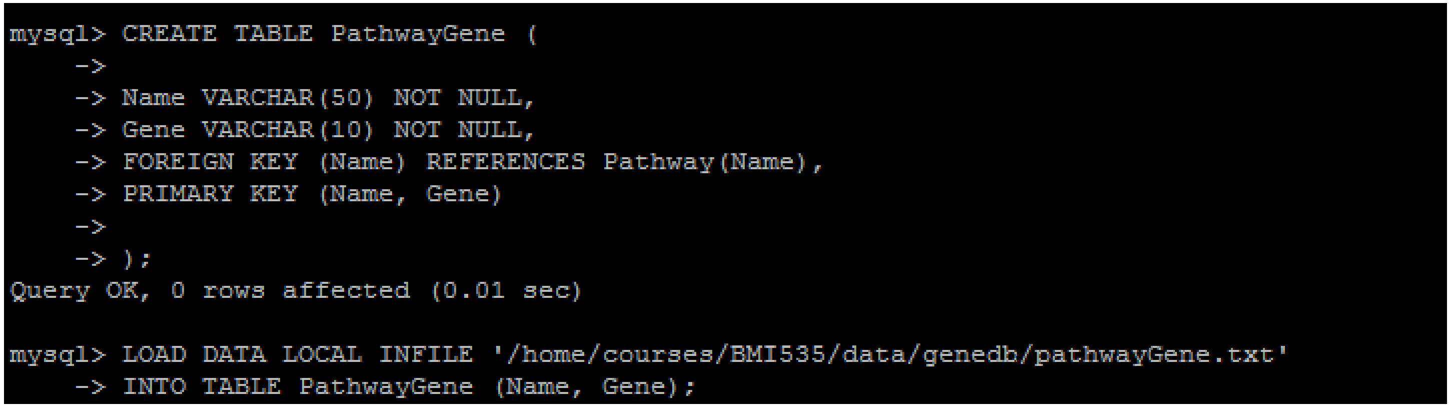
```
mysql> SELECT DISTINCT p.Gene, g.GeneId
    -> FROM PathwayGene p LEFT OUTER JOIN Gene g ON (p.Gene=g.Name)
    -> WHERE g.GeneId IS NULL;
+---------+--------+
| Gene    | GeneId |
+---------+--------+
| AGPAT6  | NULL   |
| ANKRD57 | NULL   |
| FAM73B  | NULL   |
| FAM82A2 | NULL   |
| MOSC2   | NULL   |
| OBFC2A  | NULL   |
| PTRF    | NULL   |
| SDPR    | NULL   |
| SQRDL   | NULL   |
| FYB     | NULL   |
```

Revisiting Pathway Tables

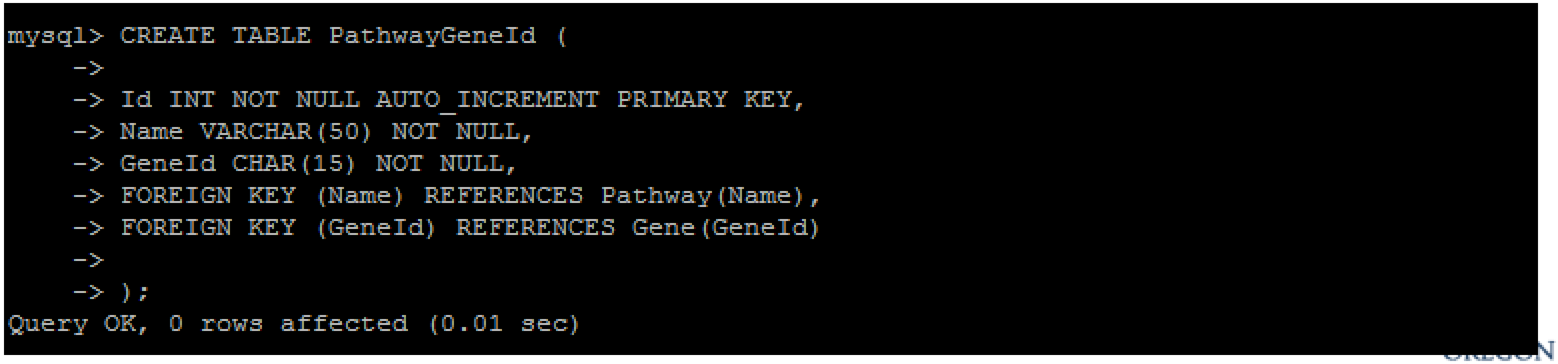


```
mysql> CREATE TABLE PathwayGene (
    ->
    -> Name VARCHAR(50) NOT NULL,
    -> Gene VARCHAR(10) NOT NULL,
    -> FOREIGN KEY (Name) REFERENCES Pathway(Name),
    -> PRIMARY KEY (Name, Gene)
    ->
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> LOAD DATA LOCAL INFILE '/home/courses/BMI535/data/genedb/pathwayGene.txt'
    -> INTO TABLE PathwayGene (Name, Gene);
```

Revisiting Pathway Tables

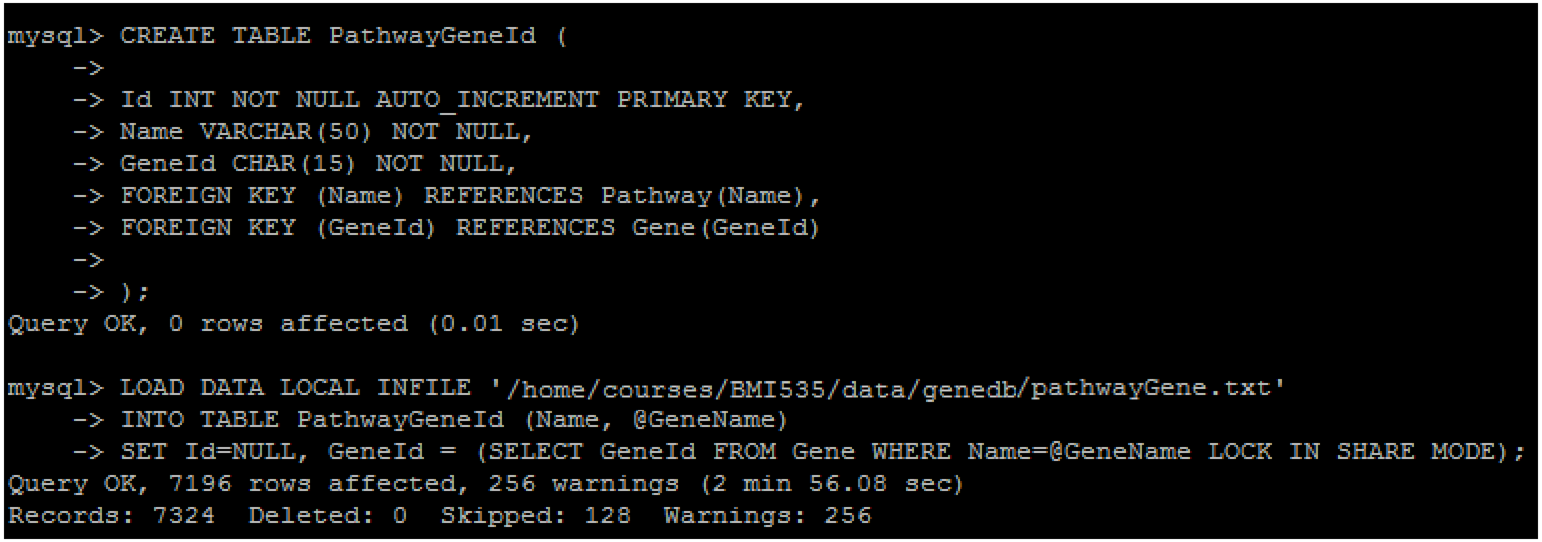


```
mysql> CREATE TABLE PathwayGene (
    ->
    -> Name VARCHAR(50) NOT NULL,
    -> Gene VARCHAR(10) NOT NULL,
    -> FOREIGN KEY (Name) REFERENCES Pathway(Name),
    -> PRIMARY KEY (Name, Gene)
    ->
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> LOAD DATA LOCAL INFILE '/home/courses/BMI535/data/genedb/pathwayGene.txt'
    -> INTO TABLE PathwayGene (Name, Gene);
```



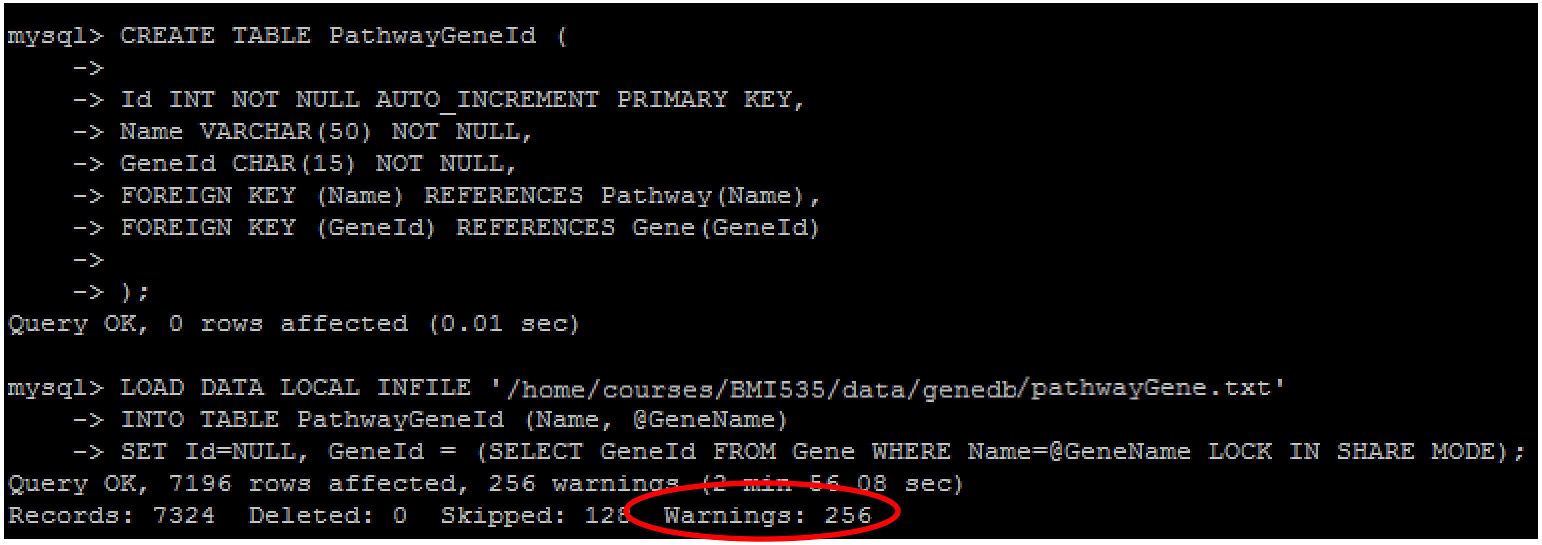
```
mysql> CREATE TABLE PathwayGeneId (
    ->
    -> Id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> Name VARCHAR(50) NOT NULL,
    -> GeneId CHAR(15) NOT NULL,
    -> FOREIGN KEY (Name) REFERENCES Pathway(Name),
    -> FOREIGN KEY (GeneId) REFERENCES Gene(GeneId)
    ->
    -> );
Query OK, 0 rows affected (0.01 sec)
```

Revisiting Pathway Tables



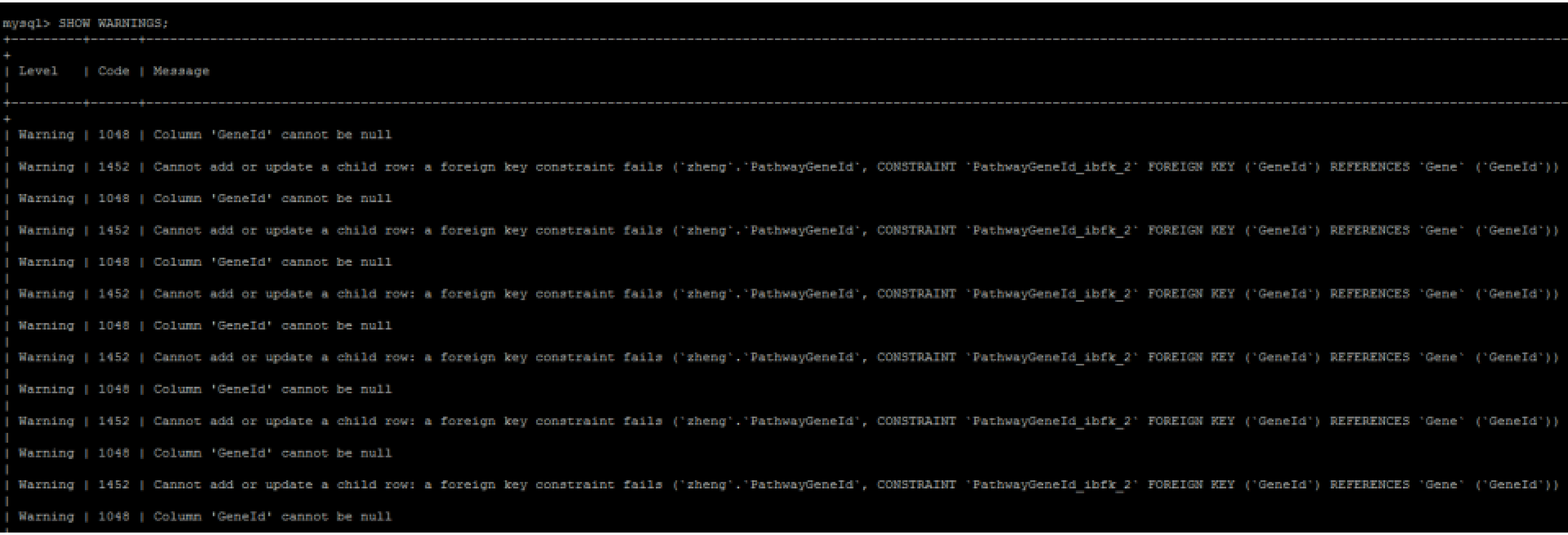
```
mysql> CREATE TABLE PathwayGeneId (
    ->
    -> Id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> Name VARCHAR(50) NOT NULL,
    -> GeneId CHAR(15) NOT NULL,
    -> FOREIGN KEY (Name) REFERENCES Pathway(Name),
    -> FOREIGN KEY (GeneId) REFERENCES Gene(GeneId)
    ->
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> LOAD DATA LOCAL INFILE '/home/courses/BMI535/data/genedb/pathwayGene.txt'
    -> INTO TABLE PathwayGeneId (Name, @GeneName)
    -> SET Id=NULL, GeneId = (SELECT GeneId FROM Gene WHERE Name=@GeneName LOCK IN SHARE MODE);
Query OK, 7196 rows affected, 256 warnings (2 min 56.08 sec)
Records: 7324  Deleted: 0  Skipped: 128  Warnings: 256
```

Revisiting Pathway Tables



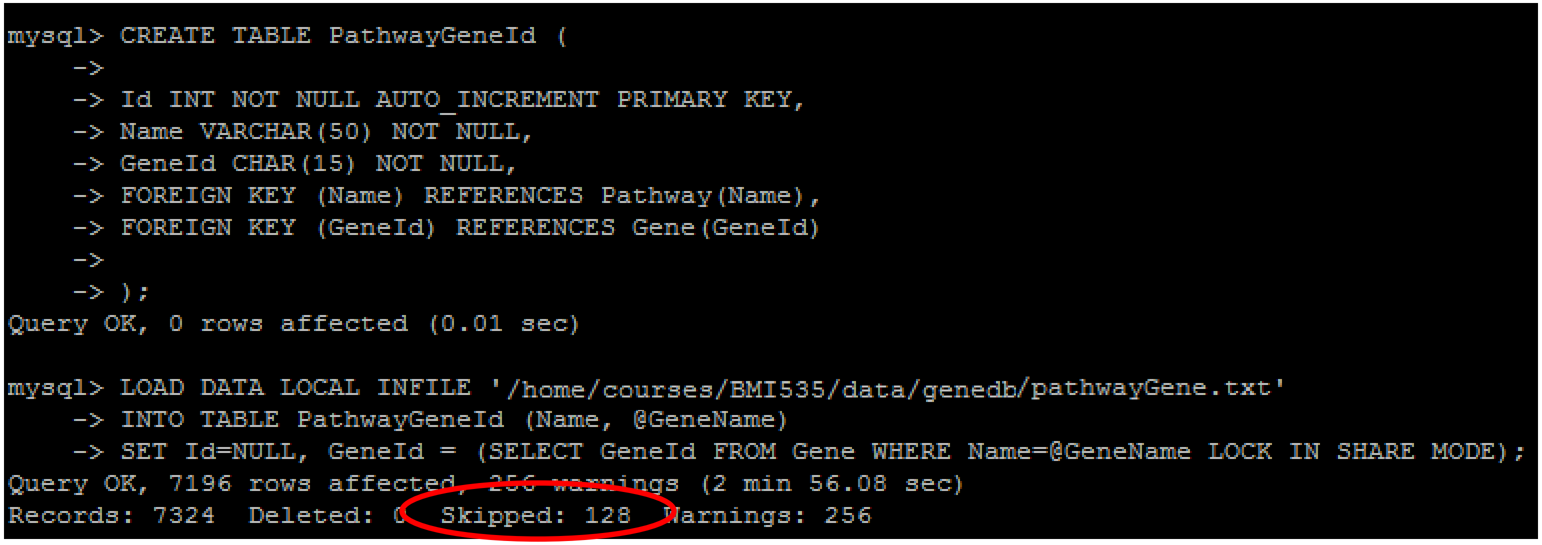
```
mysql> CREATE TABLE PathwayGeneId (
    ->
    -> Id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> Name VARCHAR(50) NOT NULL,
    -> GeneId CHAR(15) NOT NULL,
    -> FOREIGN KEY (Name) REFERENCES Pathway(Name),
    -> FOREIGN KEY (GeneId) REFERENCES Gene(GeneId)
    ->
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> LOAD DATA LOCAL INFILE '/home/courses/BMI535/data/genedb/pathwayGene.txt'
    -> INTO TABLE PathwayGeneId (Name, @GeneName)
    -> SET Id=NULL, GeneId = (SELECT GeneId FROM Gene WHERE Name=@GeneName LOCK IN SHARE MODE);
Query OK, 7196 rows affected, 256 warnings (2 min 56.08 sec)
Records: 7324  Deleted: 0  Skipped: 128  Warnings: 256
```

Revisiting Pathway Tables



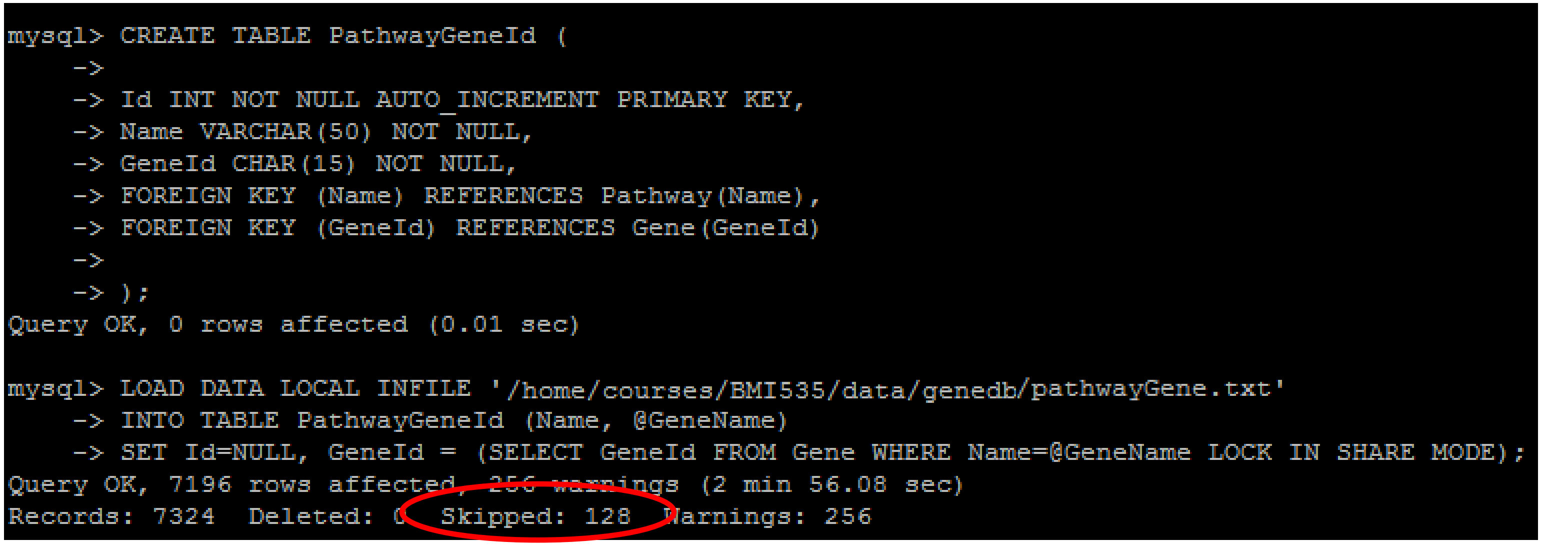
```
mysql> SHOW WARNINGS;
+---------+------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
+
| Level   | Code | Message
|
+---------+------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
+
| Warning | 1048 | Column 'GeneId' cannot be null
|
| Warning | 1452 | Cannot add or update a child row: a foreign key constraint fails (`zheng`.`PathwayGeneId`, CONSTRAINT `PathwayGeneId_ibfk_2` FOREIGN KEY (`GeneId`) REFERENCES `Gene` (`GeneId`))
|
| Warning | 1048 | Column 'GeneId' cannot be null
|
| Warning | 1452 | Cannot add or update a child row: a foreign key constraint fails (`zheng`.`PathwayGeneId`, CONSTRAINT `PathwayGeneId_ibfk_2` FOREIGN KEY (`GeneId`) REFERENCES `Gene` (`GeneId`))
|
| Warning | 1048 | Column 'GeneId' cannot be null
|
| Warning | 1452 | Cannot add or update a child row: a foreign key constraint fails (`zheng`.`PathwayGeneId`, CONSTRAINT `PathwayGeneId_ibfk_2` FOREIGN KEY (`GeneId`) REFERENCES `Gene` (`GeneId`))
|
| Warning | 1048 | Column 'GeneId' cannot be null
|
| Warning | 1452 | Cannot add or update a child row: a foreign key constraint fails (`zheng`.`PathwayGeneId`, CONSTRAINT `PathwayGeneId_ibfk_2` FOREIGN KEY (`GeneId`) REFERENCES `Gene` (`GeneId`))
|
| Warning | 1048 | Column 'GeneId' cannot be null
|
| Warning | 1452 | Cannot add or update a child row: a foreign key constraint fails (`zheng`.`PathwayGeneId`, CONSTRAINT `PathwayGeneId_ibfk_2` FOREIGN KEY (`GeneId`) REFERENCES `Gene` (`GeneId`))
|
| Warning | 1048 | Column 'GeneId' cannot be null
|
| Warning | 1452 | Cannot add or update a child row: a foreign key constraint fails (`zheng`.`PathwayGeneId`, CONSTRAINT `PathwayGeneId_ibfk_2` FOREIGN KEY (`GeneId`) REFERENCES `Gene` (`GeneId`))
|
| Warning | 1048 | Column 'GeneId' cannot be null
```

Revisiting Pathway Tables

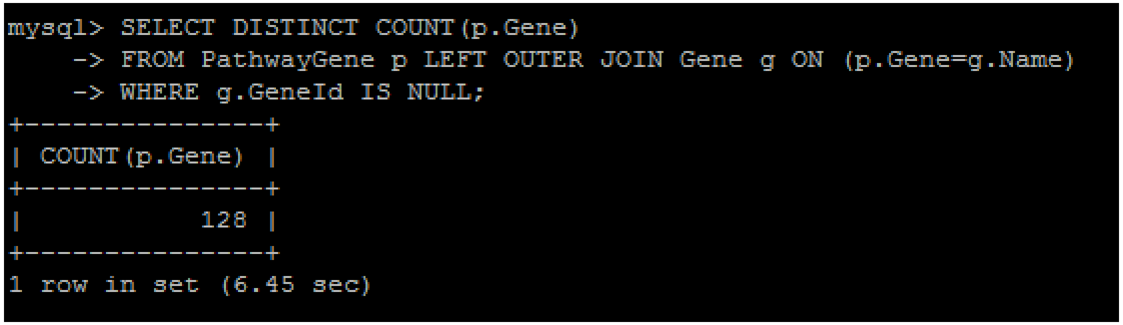


```
mysql> CREATE TABLE PathwayGeneId (
    ->
    -> Id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> Name VARCHAR(50) NOT NULL,
    -> GeneId CHAR(15) NOT NULL,
    -> FOREIGN KEY (Name) REFERENCES Pathway(Name),
    -> FOREIGN KEY (GeneId) REFERENCES Gene(GeneId)
    ->
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> LOAD DATA LOCAL INFILE '/home/courses/BMI535/data/genedb/pathwayGene.txt'
    -> INTO TABLE PathwayGeneId (Name, @GeneName)
    -> SET Id=NULL, GeneId = (SELECT GeneId FROM Gene WHERE Name=@GeneName LOCK IN SHARE MODE);
Query OK, 7196 rows affected, 256 warnings (2 min 56.08 sec)
Records: 7324  Deleted: 0  Skipped: 128  Warnings: 256
```

Revisiting Pathway Tables

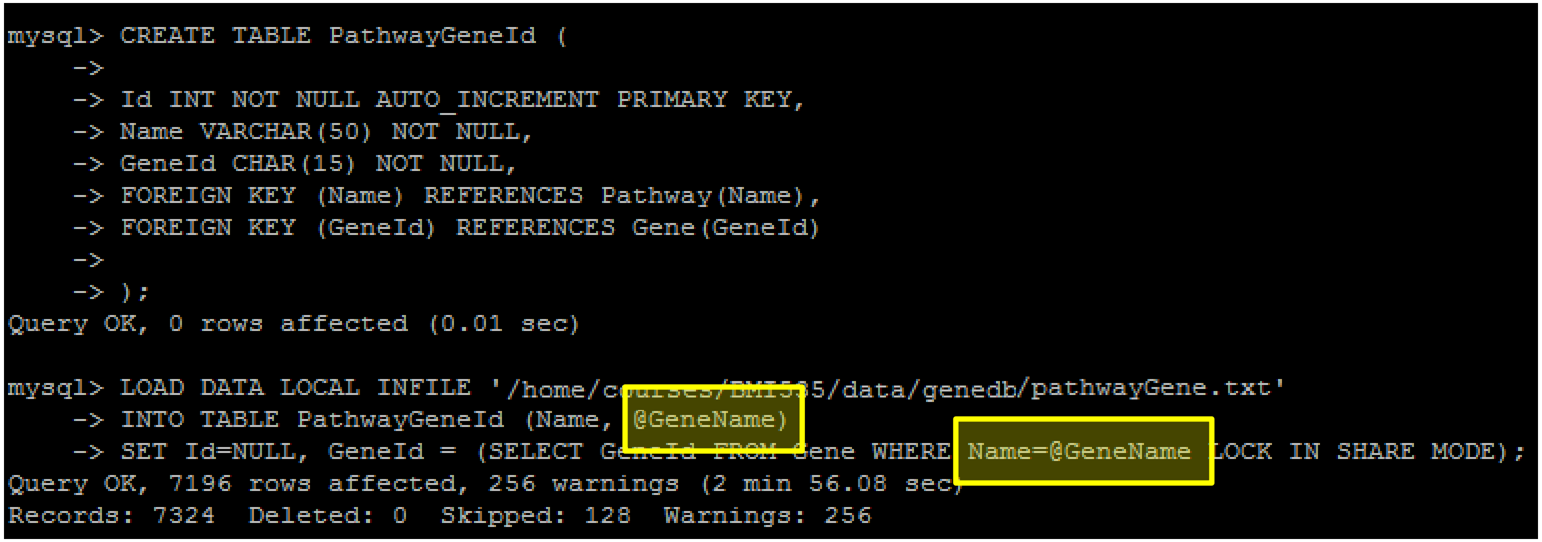


```
mysql> CREATE TABLE PathwayGeneId (
    ->
    -> Id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> Name VARCHAR(50) NOT NULL,
    -> GeneId CHAR(15) NOT NULL,
    -> FOREIGN KEY (Name) REFERENCES Pathway(Name),
    -> FOREIGN KEY (GeneId) REFERENCES Gene(GeneId)
    ->
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> LOAD DATA LOCAL INFILE '/home/courses/BMI535/data/genedb/pathwayGene.txt'
    -> INTO TABLE PathwayGeneId (Name, @GeneName)
    -> SET Id=NULL, GeneId = (SELECT GeneId FROM Gene WHERE Name=@GeneName LOCK IN SHARE MODE);
Query OK, 7196 rows affected, 256 warnings (2 min 56.08 sec)
Records: 7324  Deleted: 0  Skipped: 128  Warnings: 256
```



```
mysql> SELECT DISTINCT COUNT(p.Gene)
    -> FROM PathwayGene p LEFT OUTER JOIN Gene g ON (p.Gene=g.Name)
    -> WHERE g.GeneId IS NULL;
+---------------+
| COUNT(p.Gene) |
+---------------+
|           128 |
+---------------+
1 row in set (6.45 sec)
```

Revisiting Pathway Tables

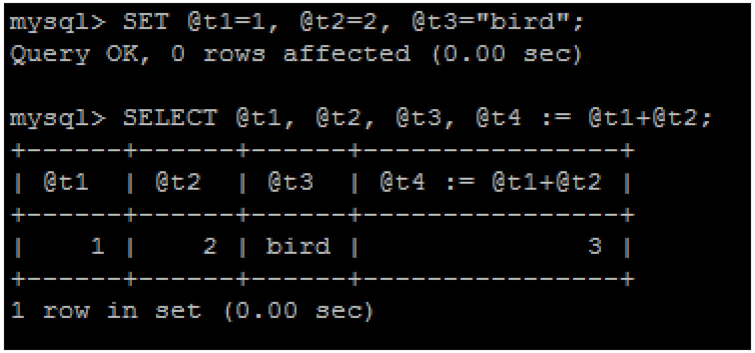


```
mysql> CREATE TABLE PathwayGeneId (
    ->
    -> Id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> Name VARCHAR(50) NOT NULL,
    -> GeneId CHAR(15) NOT NULL,
    -> FOREIGN KEY (Name) REFERENCES Pathway(Name),
    -> FOREIGN KEY (GeneId) REFERENCES Gene(GeneId)
    ->
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> LOAD DATA LOCAL INFILE '/home/courses/BMI535/data/genedb/pathwayGene.txt'
    -> INTO TABLE PathwayGeneId (Name, @GeneName)
    -> SET Id=NULL, GeneId = (SELECT GeneId FROM Gene WHERE Name=@GeneName LOCK IN SHARE MODE);
Query OK, 7196 rows affected, 256 warnings (2 min 56.08 sec)
Records: 7324  Deleted: 0  Skipped: 128  Warnings: 256
```

User-Defined Variables

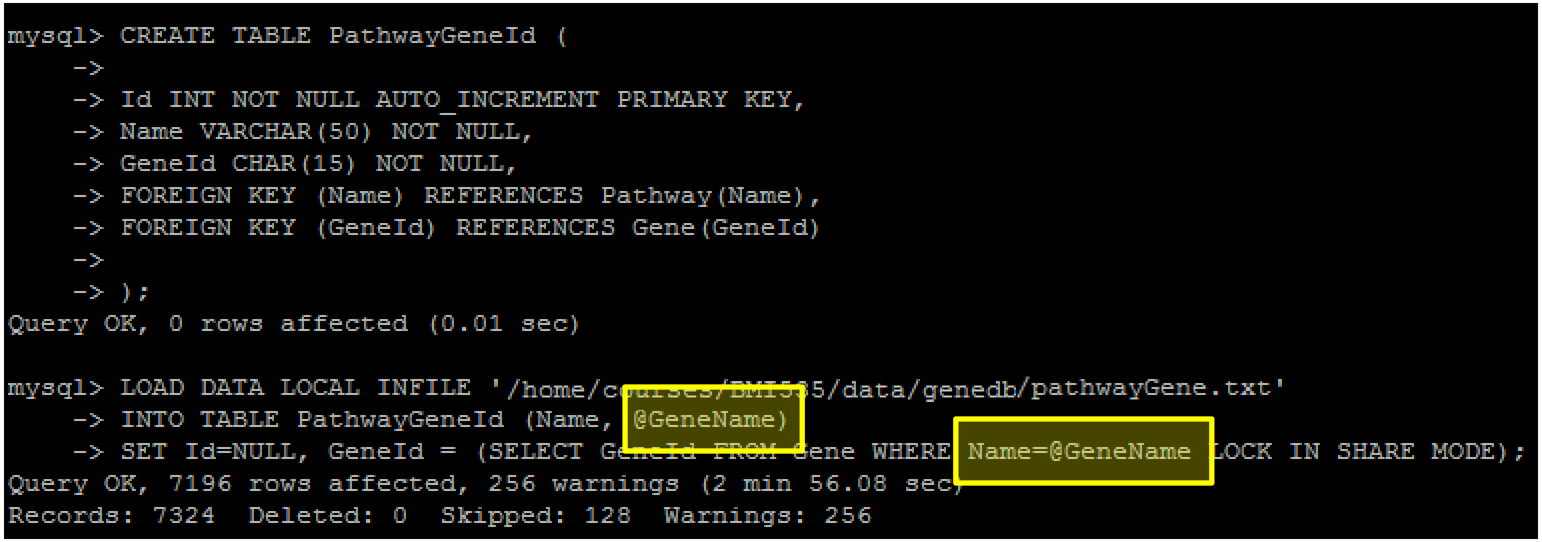
Store a user-defined variable in one statement and refer to it in a later statement

- @var_name
- not case sensitive
- session specific
- assignment
  - SET (=, :=)
  - SELECT (:=)



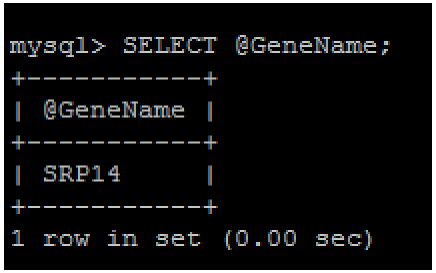
```
mysql> SET @t1=1, @t2=2, @t3="bird";
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @t1, @t2, @t3, @t4 := @t1+@t2;
+------+------+------+----------------+
| @t1  | @t2  | @t3  | @t4 := @t1+@t2 |
+------+------+------+----------------+
|    1 |    2 | bird |              3 |
+------+------+------+----------------+
1 row in set (0.00 sec)
```

Revisiting Pathway Tables

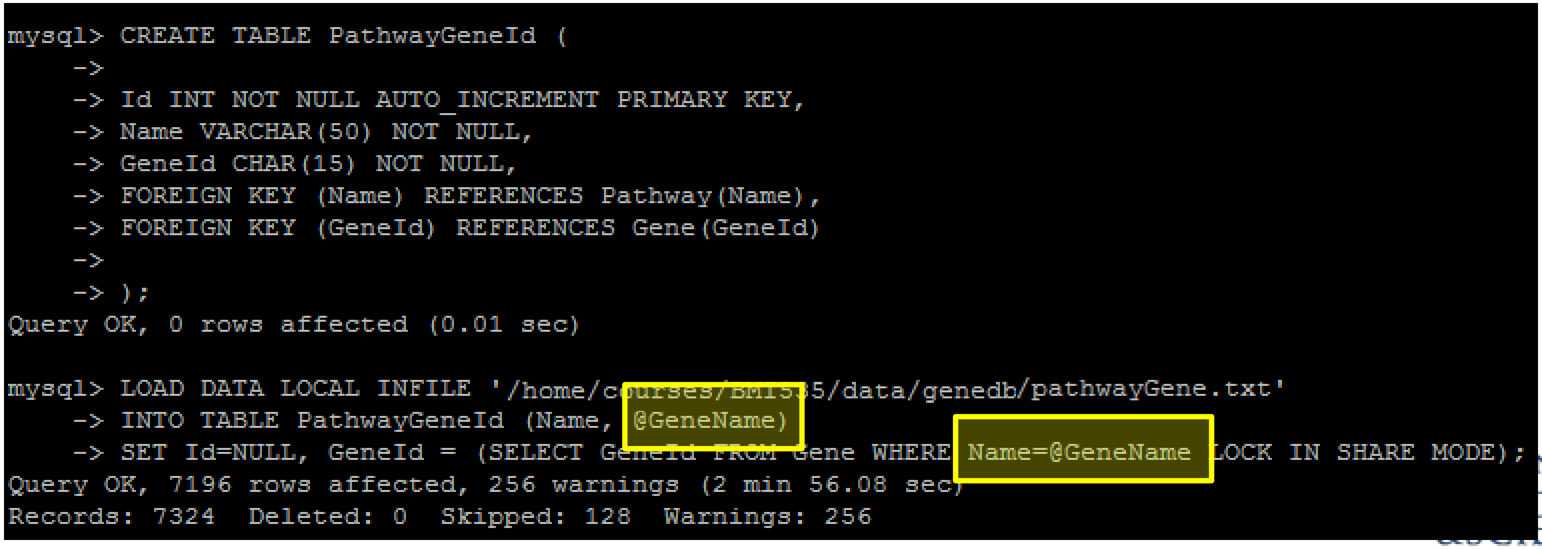


```
mysql> CREATE TABLE PathwayGeneId (
    ->
    -> Id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> Name VARCHAR(50) NOT NULL,
    -> GeneId CHAR(15) NOT NULL,
    -> FOREIGN KEY (Name) REFERENCES Pathway(Name),
    -> FOREIGN KEY (GeneId) REFERENCES Gene(GeneId)
    ->
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> LOAD DATA LOCAL INFILE '/home/courses/BMI535/data/genedb/pathwayGene.txt'
    -> INTO TABLE PathwayGeneId (Name, @GeneName)
    -> SET Id=NULL, GeneId = (SELECT GeneId FROM Gene WHERE Name=@GeneName LOCK IN SHARE MODE);
Query OK, 7196 rows affected, 256 warnings (2 min 56.08 sec)
Records: 7324  Deleted: 0  Skipped: 128  Warnings: 256
```

User-Defined Variables



```
mysql> SELECT @GeneName;
+------------+
| @GeneName |
+------------+
| SRP14      |
+------------+
1 row in set (0.00 sec)
```

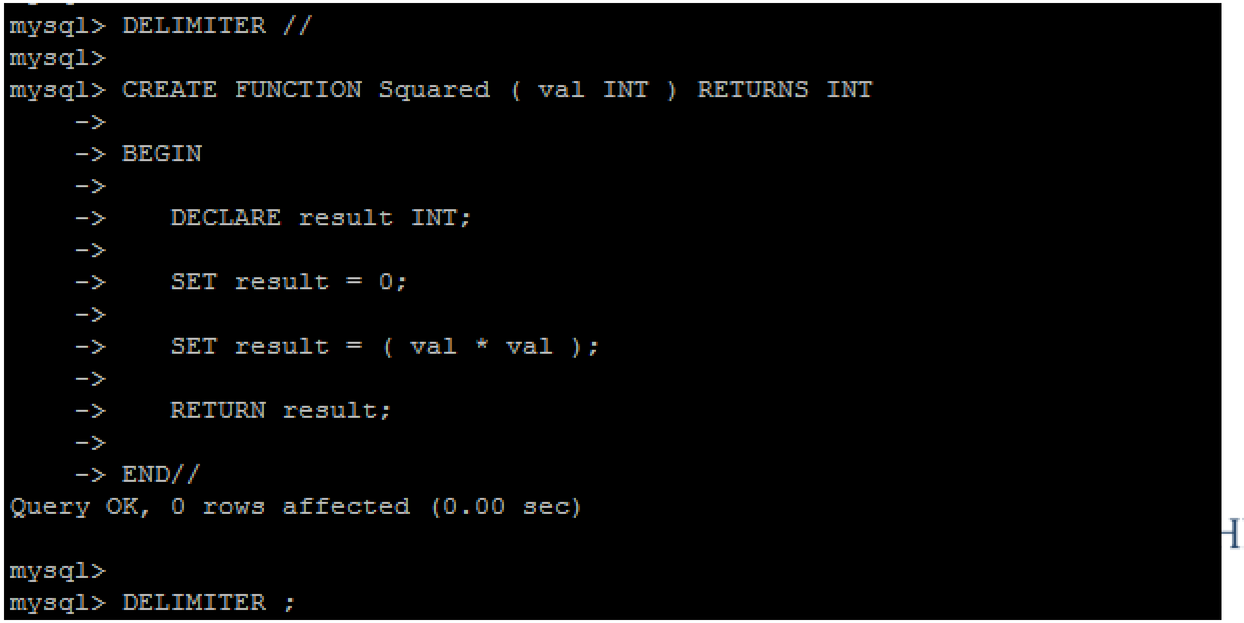


```
mysql> CREATE TABLE PathwayGeneId (
    ->
    -> Id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> Name VARCHAR(50) NOT NULL,
    -> GeneId CHAR(15) NOT NULL,
    -> FOREIGN KEY (Name) REFERENCES Pathway(Name),
    -> FOREIGN KEY (GeneId) REFERENCES Gene(GeneId)
    ->
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> LOAD DATA LOCAL INFILE '/home/courses/BMI535/data/genedb/pathwayGene.txt'
    -> INTO TABLE PathwayGeneId (Name, @GeneName)
    -> SET Id=NULL, GeneId = (SELECT GeneId FROM Gene WHERE Name=@GeneName LOCK IN SHARE MODE);
Query OK, 7196 rows affected, 256 warnings (2 min 56.08 sec)
Records: 7324  Deleted: 0  Skipped: 128  Warnings: 256
```

User-Defined Functions

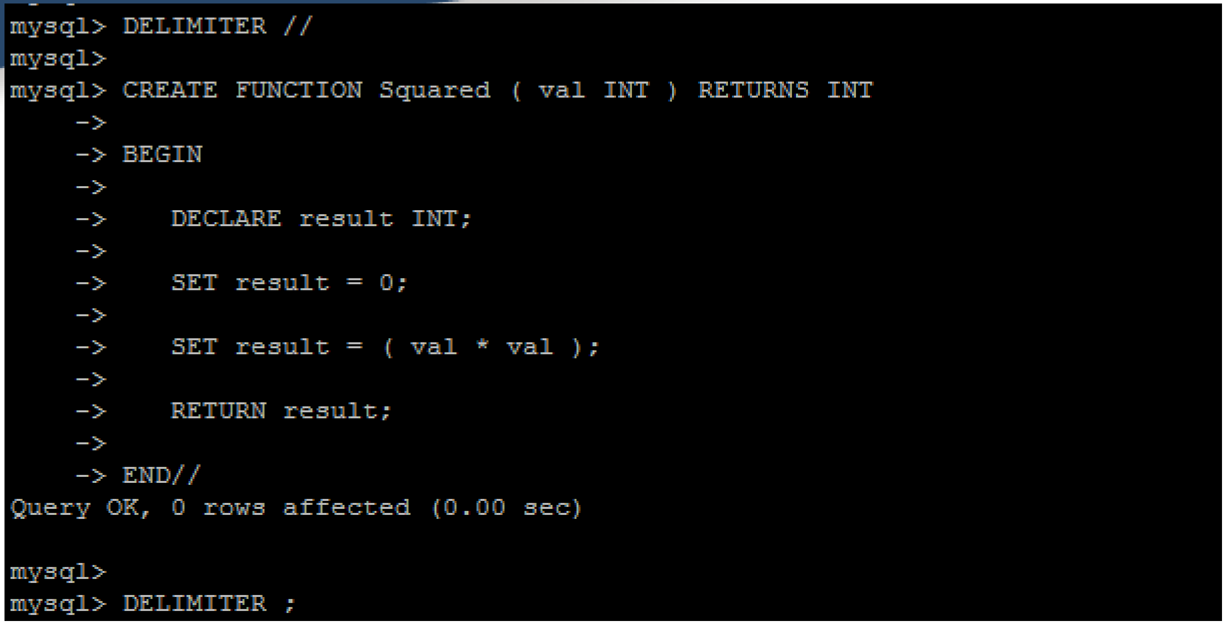
User defined function

- extend built-in functions, returns a value, used in SELECT
- setting different DELIMITER
- DECLARE local variable

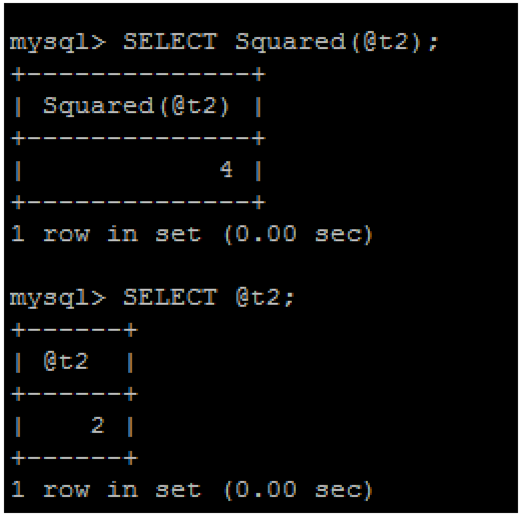


```
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION Squared ( val INT ) RETURNS INT
    ->
    -> BEGIN
    ->
    ->    DECLARE result INT;
    ->
    ->    SET result = 0;
    ->
    ->    SET result = ( val * val );
    ->
    ->    RETURN result;
    ->
    -> END//
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> DELIMITER ;
```

User-Defined Functions



```
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION Squared ( val INT ) RETURNS INT
    ->
    -> BEGIN
    ->
    ->    DECLARE result INT;
    ->
    ->    SET result = 0;
    ->
    ->    SET result = ( val * val );
    ->
    ->    RETURN result;
    ->
    -> END//
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> DELIMITER ;
```

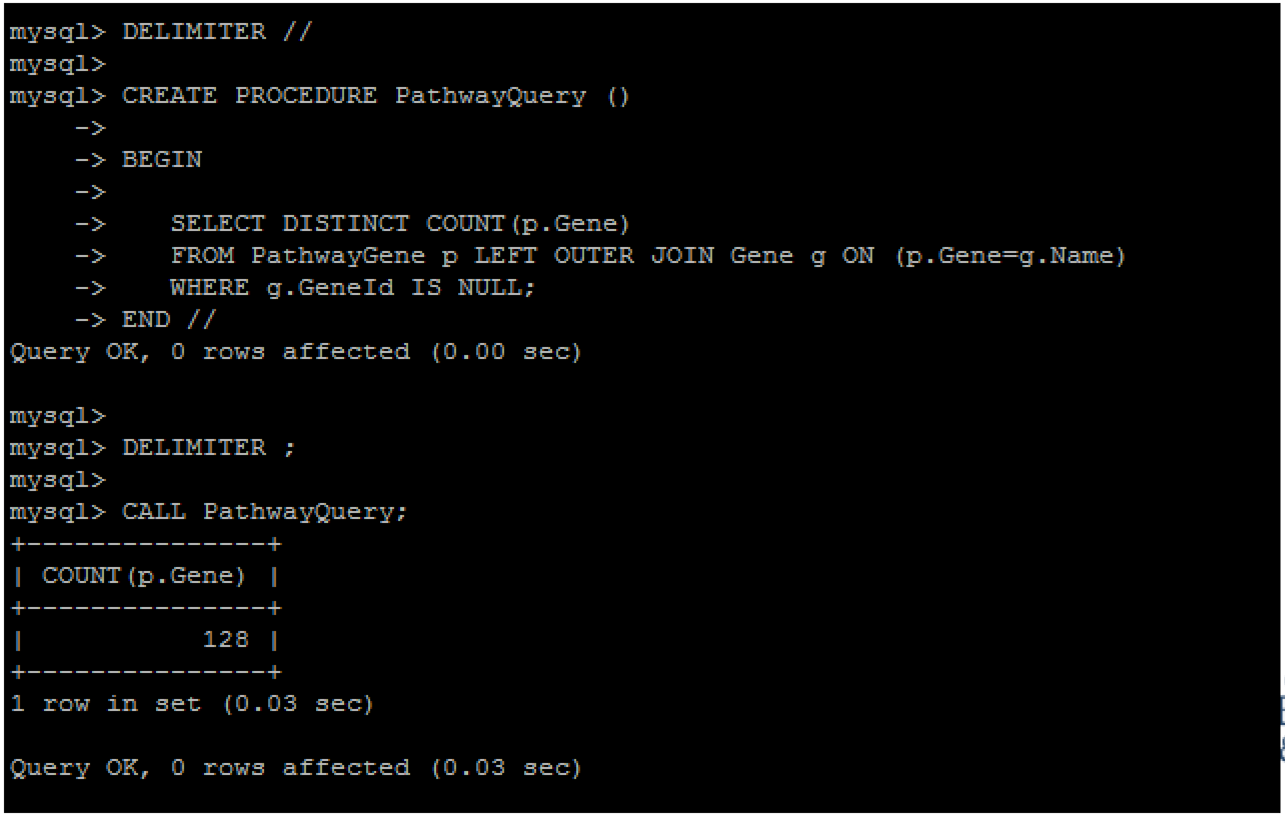


```
mysql> SELECT Squared(@t2);
+---------------+
| Squared(@t2) |
+---------------+
|             4 |
+---------------+
1 row in set (0.00 sec)

mysql> SELECT @t2;
+------+
| @t2  |
+------+
|    2 |
+------+
1 row in set (0.00 sec)
```

Stored Procedures

Stored Procedure

- CALL a list of commands

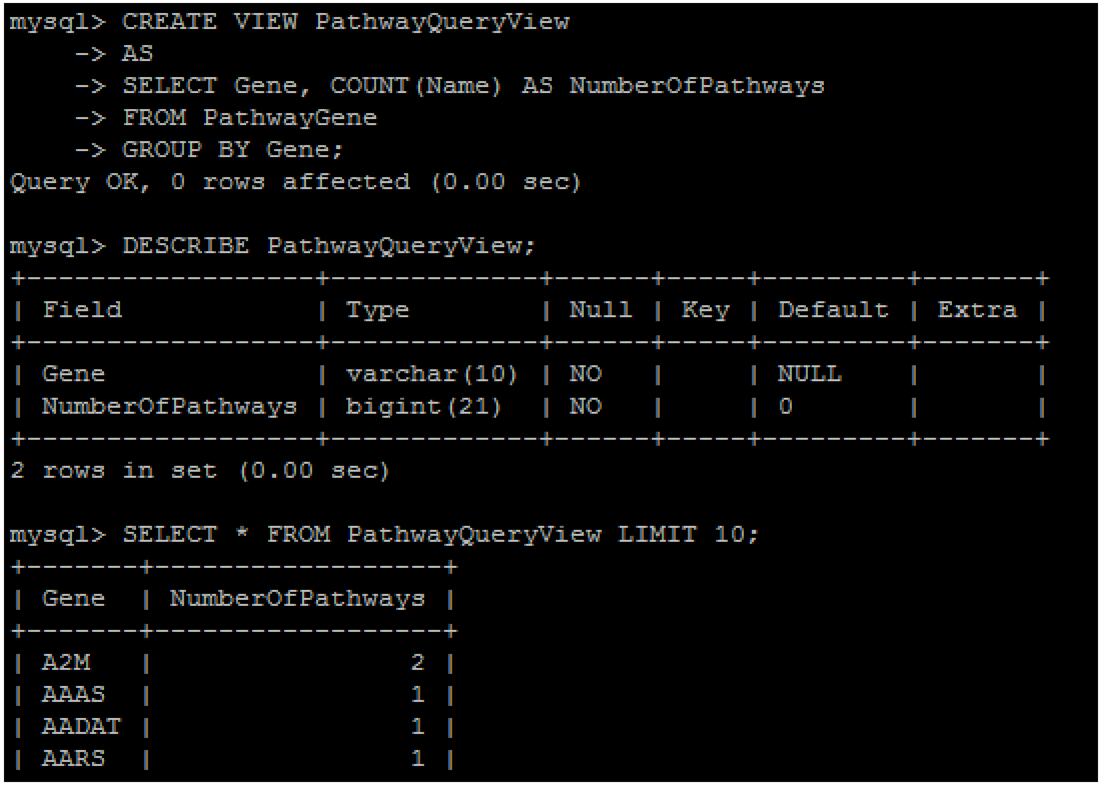


```
mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE PathwayQuery ()
    ->
    -> BEGIN
    ->
    ->    SELECT DISTINCT COUNT(p.Gene)
    ->    FROM PathwayGene p LEFT OUTER JOIN Gene g ON (p.Gene=g.Name)
    ->    WHERE g.GeneId IS NULL;
    -> END //
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> DELIMITER ;
mysql>
mysql> CALL PathwayQuery;
+---------------+
| COUNT(p.Gene) |
+---------------+
|           128 |
+---------------+
1 row in set (0.03 sec)

Query OK, 0 rows affected (0.03 sec)
```

Table Views

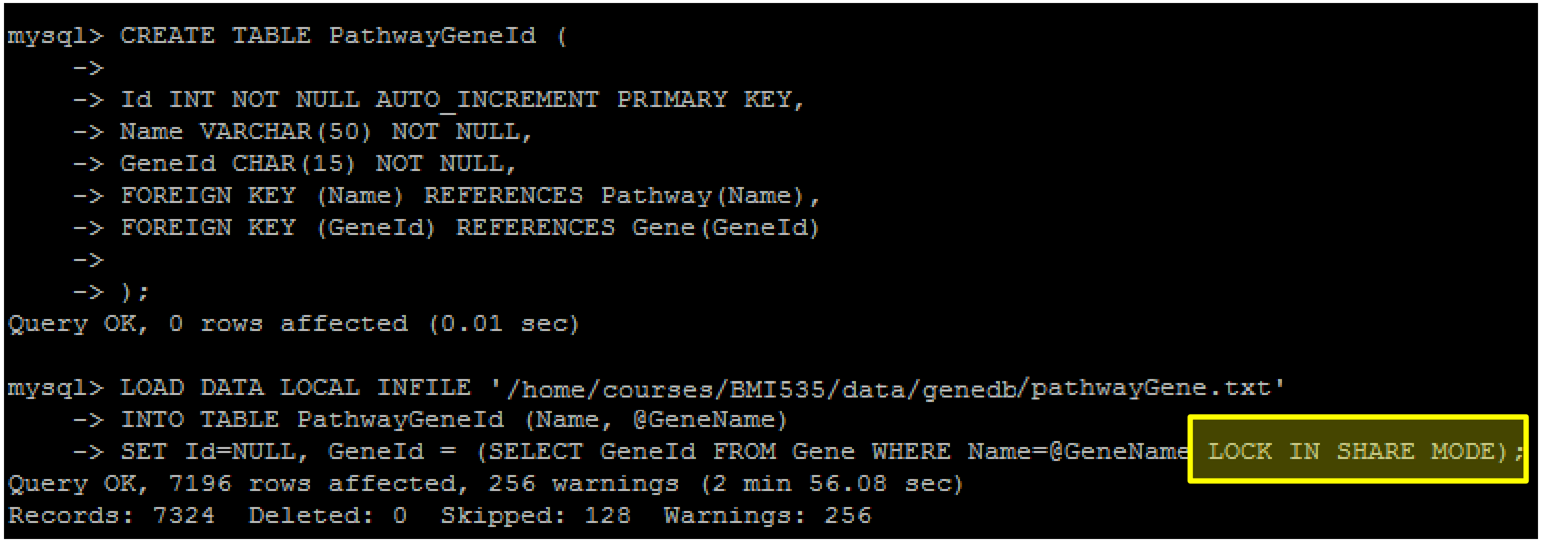
Views

- a query result encapsulated in a virtual table



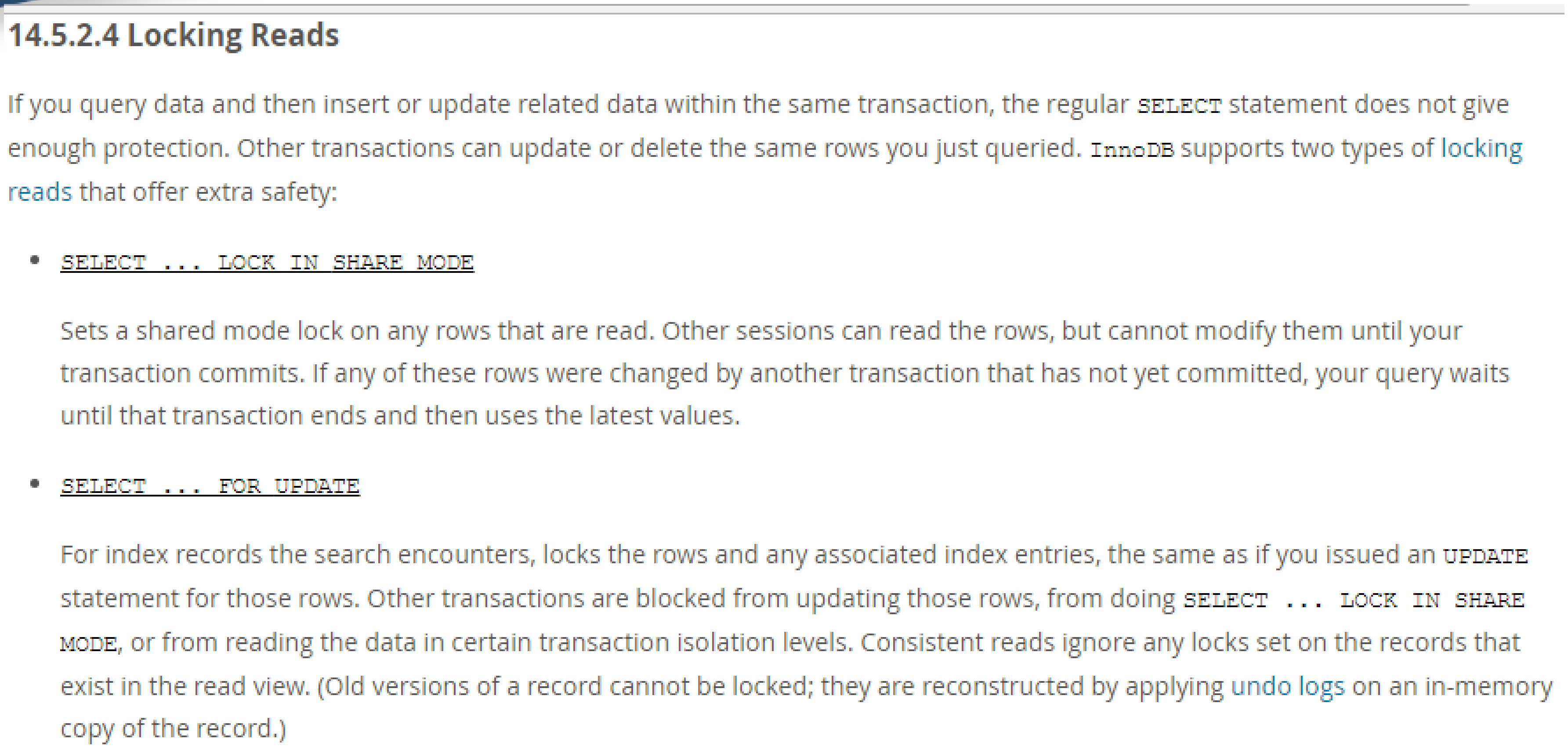
```
mysql> CREATE VIEW PathwayQueryView
    -> AS
    -> SELECT Gene, COUNT(Name) AS NumberOfPathways
    -> FROM PathwayGene
    -> GROUP BY Gene;
Query OK, 0 rows affected (0.00 sec)

mysql> DESCRIBE PathwayQueryView;
+------------------+-------------+------+-----+---------+-------+
| Field            | Type        | Null | Key | Default | Extra |
+------------------+-------------+------+-----+---------+-------+
| Gene             | varchar(10) | NO   |     | NULL    |       |
| NumberOfPathways | bigint(21)  | NO   |     | 0       |       |
+------------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM PathwayQueryView LIMIT 10;
+--------+------------------+
| Gene   | NumberOfPathways |
+--------+------------------+
| A2M    |                2 |
| AAAS   |                1 |
| AADAT  |                1 |
| AARS   |                1 |
```

Revisiting Pathway Tables



```
mysql> CREATE TABLE PathwayGeneId (
    ->
    -> Id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> Name VARCHAR(50) NOT NULL,
    -> GeneId CHAR(15) NOT NULL,
    -> FOREIGN KEY (Name) REFERENCES Pathway(Name),
    -> FOREIGN KEY (GeneId) REFERENCES Gene(GeneId)
    ->
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> LOAD DATA LOCAL INFILE '/home/courses/BMI535/data/genedb/pathwayGene.txt'
    -> INTO TABLE PathwayGeneId (Name, @GeneName)
    -> SET Id=NULL, GeneId = (SELECT GeneId FROM Gene WHERE Name=@GeneName LOCK IN SHARE MODE);
Query OK, 7196 rows affected, 256 warnings (2 min 56.08 sec)
Records: 7324  Deleted: 0  Skipped: 128  Warnings: 256
```

Database Locks



14.5.2.4 Locking Reads

If you query data and then insert or update related data within the same transaction, the regular `SELECT` statement does not give enough protection. Other transactions can update or delete the same rows you just queried. `InnoDB` supports two types of locking reads that offer extra safety:

- `SELECT ... LOCK IN SHARE MODE`

  Sets a shared mode lock on any rows that are read. Other sessions can read the rows, but cannot modify them until your transaction commits. If any of these rows were changed by another transaction that has not yet committed, your query waits until that transaction ends and then uses the latest values.

- `SELECT ... FOR UPDATE`

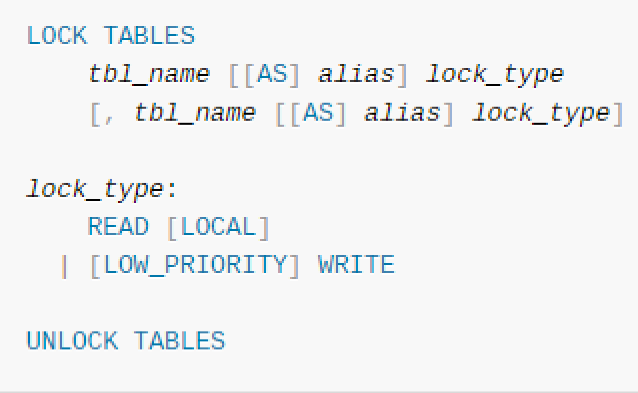
  For index records the search encounters, locks the rows and any associated index entries, the same as if you issued an `UPDATE` statement for those rows. Other transactions are blocked from updating those rows, from doing `SELECT ... LOCK IN SHARE MODE`, or from reading the data in certain transaction isolation levels. Consistent reads ignore any locks set on the records that exist in the read view. (Old versions of a record cannot be locked; they are reconstructed by applying undo logs on an in-memory copy of the record.)

https://dev.mysql.com/doc/refman/5.7/en/innodb-locking-reads.html

Database Locks

Locking

- specific rows
- specific tables



```
LOCK TABLES
    tbl_name [[AS] alias] lock_type
    [, tbl_name [[AS] alias] lock_type]

lock_type:
    READ [LOCAL]
  | [LOW_PRIORITY] WRITE

UNLOCK TABLES
```

https://dev.mysql.com/doc/refman/5.7/en/lock-tables.html

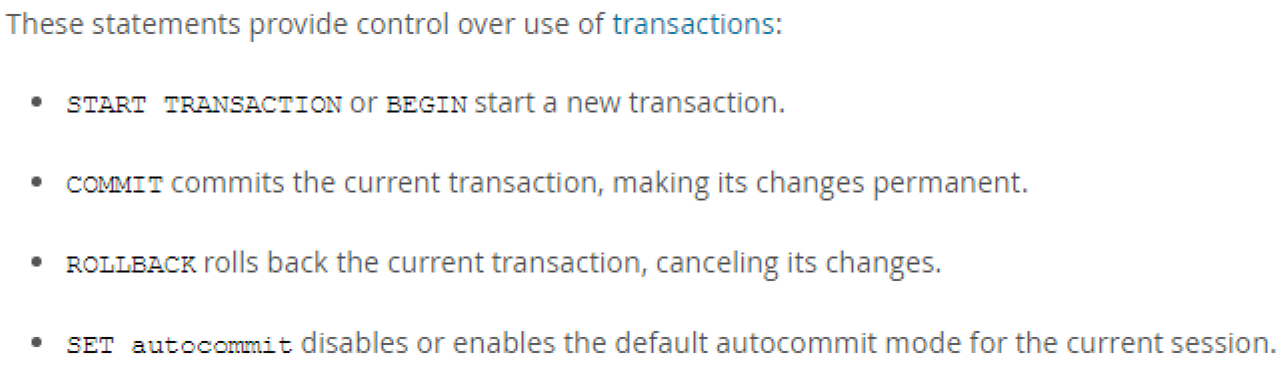
Usage

- exclusive usage (keep in mind database performance)
    - updates
    - access across multiple tables
- maintain concurrency and consistency
- at times used to mimic transactions

Database Transactions

Transactions

- sequential group of database manipulation operations to be performed as a single unit of work



These statements provide control over use of transactions:

- `START TRANSACTION` or `BEGIN` start a new transaction.
- `COMMIT` commits the current transaction, making its changes permanent.
- `ROLLBACK` rolls back the current transaction, canceling its changes.
- `SET autocommit` disables or enables the default autocommit mode for the current session.

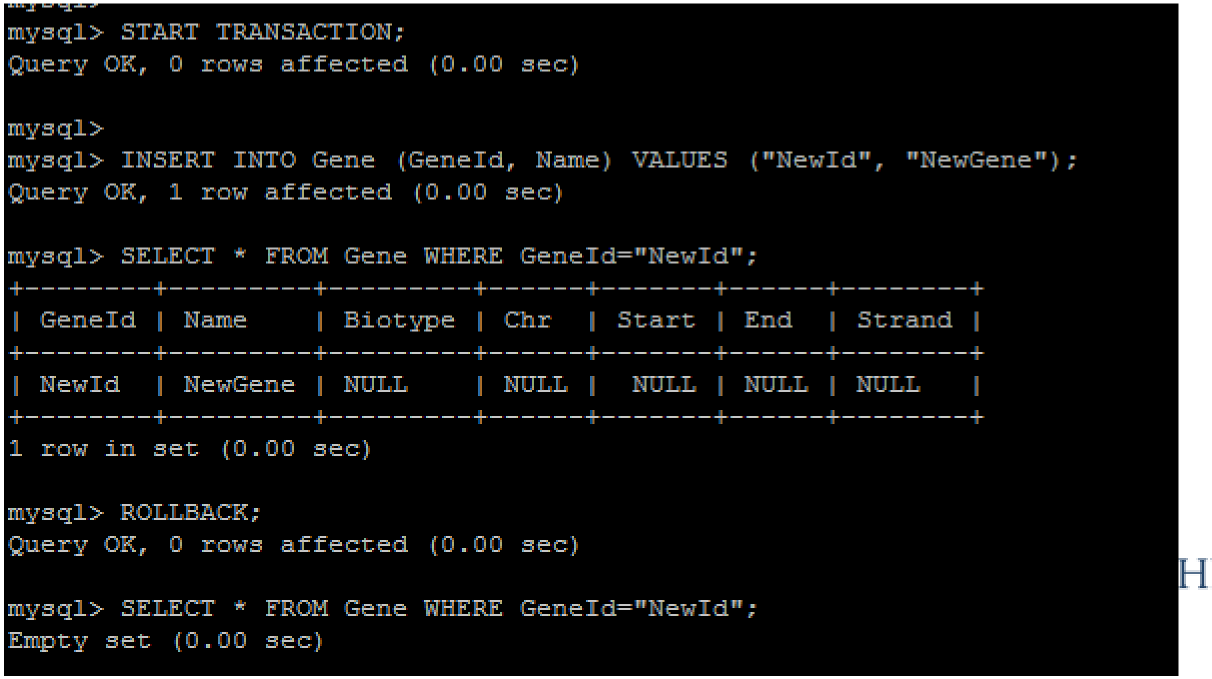
https://dev.mysql.com/doc/refman/5.7/en/commit.html

Database Transactions

START TRANSACTION – obtain database snapshot, turn off auto-commit

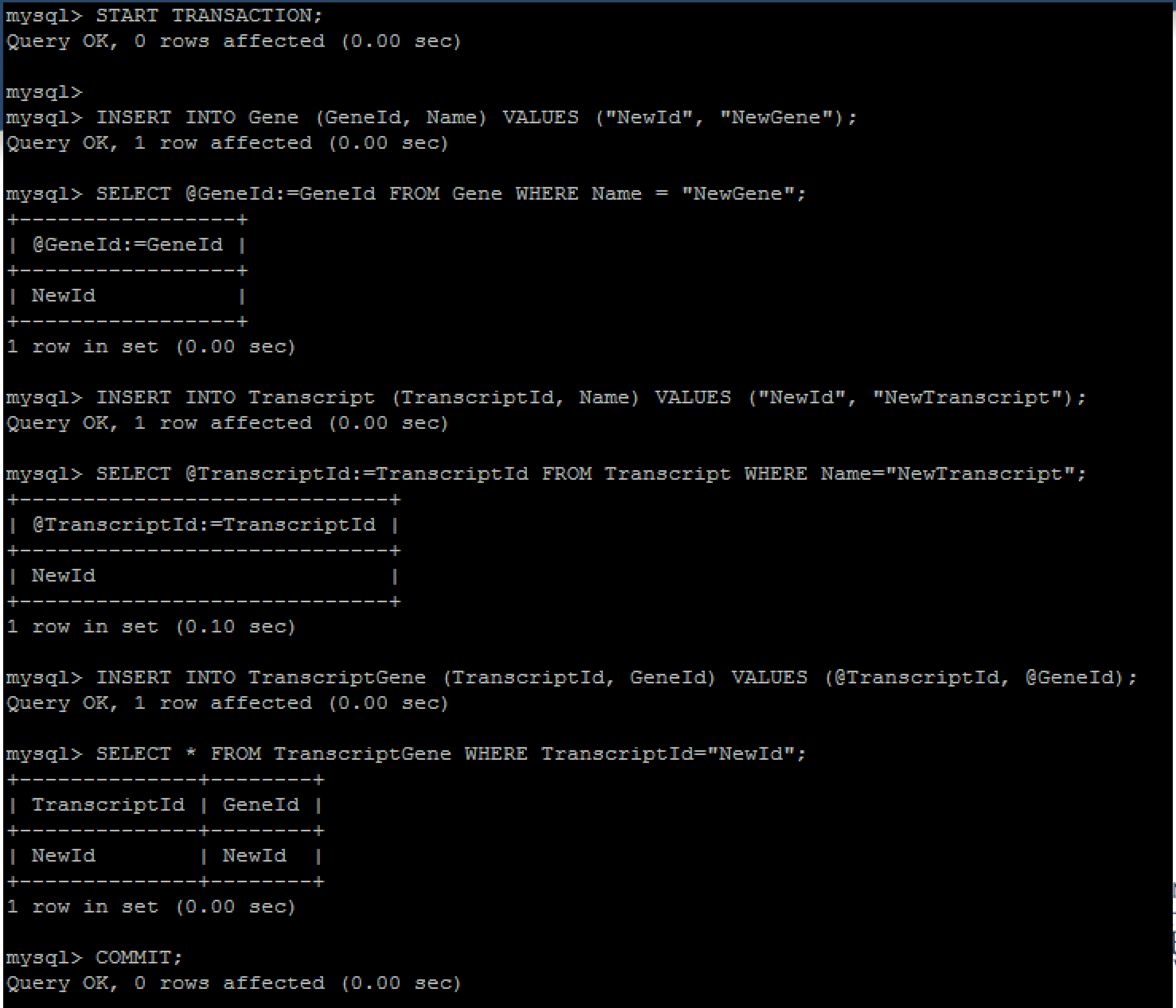
ROLLBACK – finish transaction, do not commit statements

COMMIT – finish transaction, commit all statements



```
mysql>
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> INSERT INTO Gene (GeneId, Name) VALUES ("NewId", "NewGene");
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM Gene WHERE GeneId="NewId";
+--------+---------+---------+------+-------+------+--------+
| GeneId | Name    | Biotype | Chr  | Start | End  | Strand |
+--------+---------+---------+------+-------+------+--------+
| NewId  | NewGene | NULL    | NULL |  NULL | NULL | NULL   |
+--------+---------+---------+------+-------+------+--------+
1 row in set (0.00 sec)

mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM Gene WHERE GeneId="NewId";
Empty set (0.00 sec)
```

Database Transactions



```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> INSERT INTO Gene (GeneId, Name) VALUES ("NewId", "NewGene");
Query OK, 1 row affected (0.00 sec)

mysql> SELECT @GeneId:=GeneId FROM Gene WHERE Name = "NewGene";
+-----------------+
| @GeneId:=GeneId |
+-----------------+
| NewId           |
+-----------------+
1 row in set (0.00 sec)

mysql> INSERT INTO Transcript (TranscriptId, Name) VALUES ("NewId", "NewTranscript");
Query OK, 1 row affected (0.00 sec)

mysql> SELECT @TranscriptId:=TranscriptId FROM Transcript WHERE Name="NewTranscript";
+-----------------------------+
| @TranscriptId:=TranscriptId |
+-----------------------------+
| NewId                       |
+-----------------------------+
1 row in set (0.10 sec)

mysql> INSERT INTO TranscriptGene (TranscriptId, GeneId) VALUES (@TranscriptId, @GeneId);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM TranscriptGene WHERE TranscriptId="NewId";
+--------------+--------+
| TranscriptId | GeneId |
+--------------+--------+
| NewId        | NewId  |
+--------------+--------+
1 row in set (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

Database Transactions

START TRANSACTION – obtain database snapshot, turn off auto-commit

ROLLBACK – finish transaction, do not commit statements

COMMIT – finish transaction, commit all statements

Other users will only be able to see your changes after the COMMIT statement.

ACID Compliant

Four properties of database transactions

- Atomicity
  - all or none of the statements are committed
- Consistency
  - database is not left in half finished/updated state
- Isolation
  - keeps transactions separate from one another
- Durability
  - log changes so can properly recover if needed

OREGON HEALTH & SCIENCE UNIVERSITY

Summary

User defined function

- extend built-in functions, returns a value, used in SELECT

Stored Procedure

- CALL a list of commands

Views

- a query result encapsulated in a virtual table

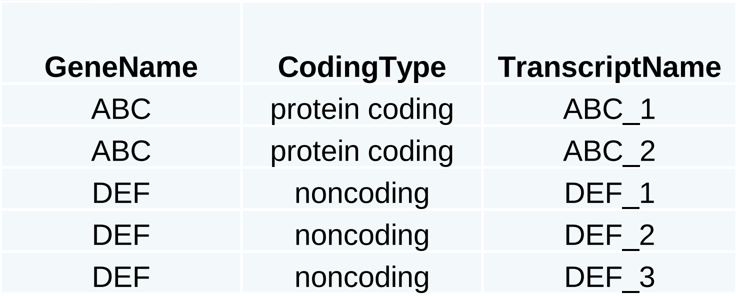
Transactions

- a set of SQL statements to be performed as a unit

Relational Database Normalization

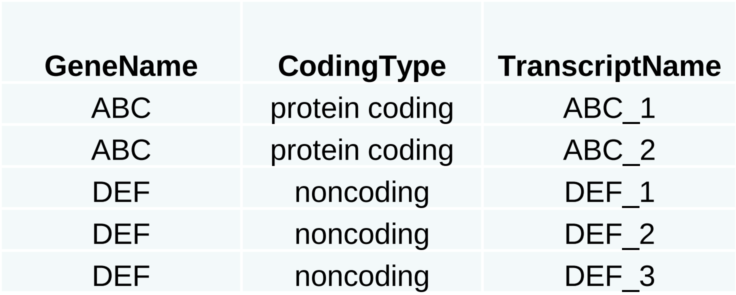
BMI 535/635

Data Table



GeneName	CodingType	TranscriptName
ABC	protein coding	ABC_1
ABC	protein coding	ABC_2
DEF	noncoding	DEF_1
DEF	noncoding	DEF_2
DEF	noncoding	DEF_3

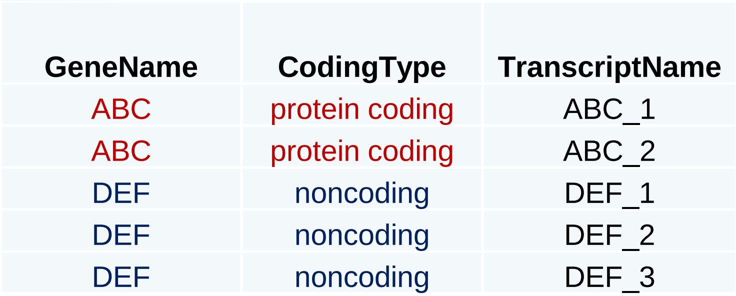
Data Table



GeneName	CodingType	TranscriptName
ABC	protein coding	ABC_1
ABC	protein coding	ABC_2
DEF	noncoding	DEF_1
DEF	noncoding	DEF_2
DEF	noncoding	DEF_3

Redundancy

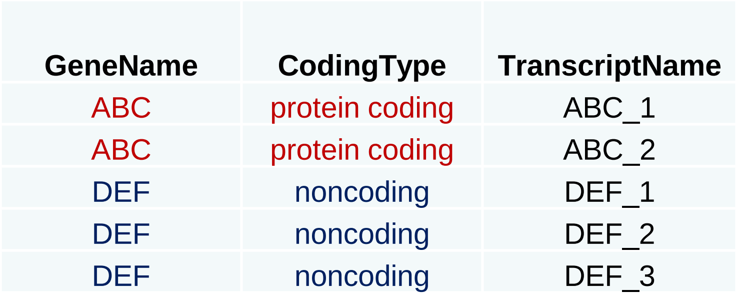
Data Table



GeneName	CodingType	TranscriptName
ABC	protein coding	ABC_1
ABC	protein coding	ABC_2
DEF	noncoding	DEF_1
DEF	noncoding	DEF_2
DEF	noncoding	DEF_3

Problems with redundancy?

Data Table



GeneName	CodingType	TranscriptName
ABC	protein coding	ABC_1
ABC	protein coding	ABC_2
DEF	noncoding	DEF_1
DEF	noncoding	DEF_2
DEF	noncoding	DEF_3

Problems with redundancy?

- increases storage
- decreases performance
- difficult to maintain data changes/integrity

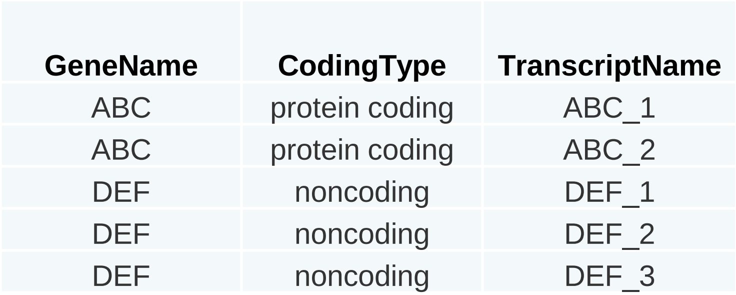
Database Normalization

Process developed by E.F. Codd in 1970 to organize a database into tables and columns. Normal forms were developed to reduce the amount of redundancy and inconsistent dependencies. Normalization organizes data into tables where each item in a row and the attribute of the item are in columns.

Goals of normalization

- eliminate redundancy
- avoid insertion/update/deletion anomalies
- consistent data dependencies
- optimize queries

Normalization



GeneName	CodingType	TranscriptName
ABC	protein coding	ABC_1
ABC	protein coding	ABC_2
DEF	noncoding	DEF_1
DEF	noncoding	DEF_2
DEF	noncoding	DEF_3

Update anomaly

- multiple rows may need to be updated
- potential for data inconsistencies/errors
    - update ‘ABC’, ‘DEF’ multiple times
    - update ‘protein coding’ multiple times

Normalization

GeneName	CodingType	TranscriptName
ABC	protein coding	ABC_1
ABC	protein coding	ABC_2
DEF	noncoding	DEF_1
DEF	noncoding	DEF_2
DEF	noncoding	DEF_3

Insertion anomaly

- all information in a row may not need to be updated

- unrelated/unnecessary information may need to be inserted

Normalization

GeneName	CodingType	TranscriptName
ABC	protein coding	ABC_1
ABC	protein coding	ABC_2
DEF	noncoding	DEF_1
DEF	noncoding	DEF_2
DEF	noncoding	DEF_3
XYZ	predicted	NULL

Insertion anomaly

- all information in a row may not need to be updated

- unrelated/unnecessary information may need to be inserted

Normalization

GeneName	CodingType	TranscriptName
ABC	protein coding	ABC_1
ABC	protein coding	ABC_2
DEF	noncoding	DEF_1
DEF	noncoding	DEF_2
DEF	noncoding	DEF_3
XYZ	predicted	NULL

Deletion anomaly

- information may be lost upon deletion
  - if we delete all NULL transcript, we will lose associated gene information

Database Normalization

Process developed by E.F. Codd in 1970 to organize a database into tables and columns. Normal forms were developed to reduce the amount of redundancy and inconsistent dependencies. Normalization organizes data into tables where each item in a row and the attribute of the item are in columns.

Normal Forms

- First normal form
- Second normal form
- Third normal form
- Boyce-Codd (BCNF, 3.5 NF)
- ….

First Normal Form

First Normal Form

- no two rows of data contain repeating information

- single entity for each column in a row

- each row should have a primary key

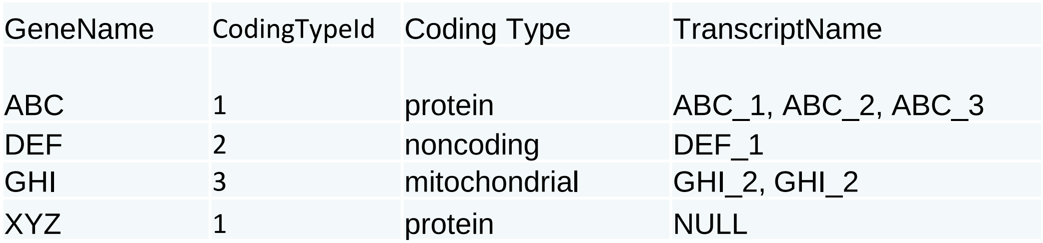
First Normal Form

First Normal Form

- no two rows of data contain repeating information
- single entity for each column in a row
- each row should have a primary key

GeneName	CodingTypeId	Coding Type	TranscriptName
ABC	1	protein	ABC_1, ABC_2, ABC_3
DEF	2	noncoding	DEF_1
GHI	3	mitochondrial	GHI_2, GHI_2
XYZ	1	protein	NULL

First Normal Form



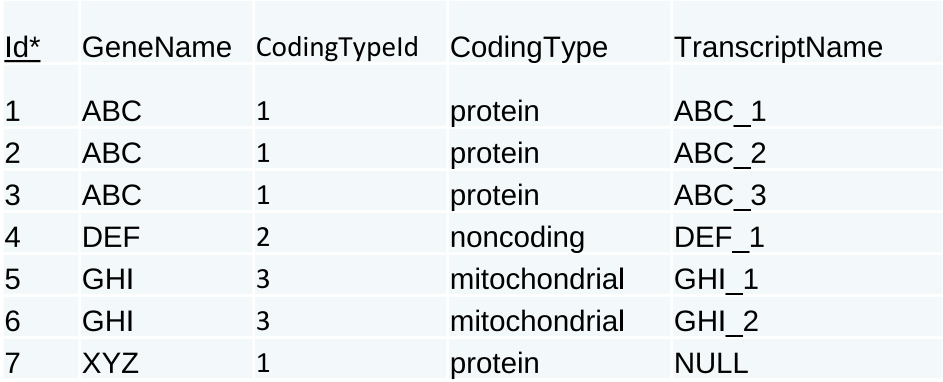
GeneName	CodingTypeId	Coding Type	TranscriptName
ABC	1	protein	ABC_1, ABC_2, ABC_3
DEF	2	noncoding	DEF_1
GHI	3	mitochondrial	GHI_2, GHI_2
XYZ	1	protein	NULL

First Normal Form

<u>Id*</u>	GeneName	CodingTypeId	CodingType	TranscriptName
1	ABC	1	protein	ABC_1
2	ABC	1	protein	ABC_2
3	ABC	1	protein	ABC_3
4	DEF	2	noncoding	DEF_1
5	GHI	3	mitochondrial	GHI_1
6	GHI	3	mitochondrial	GHI_2
7	XYZ	1	protein	NULL

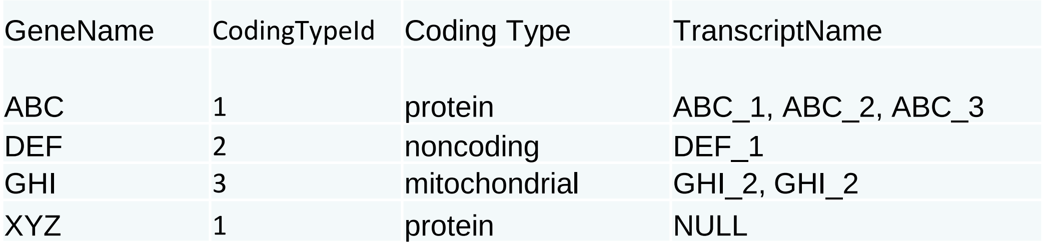
GeneName	CodingTypeId	Coding Type	TranscriptName
ABC	1	protein	ABC_1, ABC_2, ABC_3
DEF	2	noncoding	DEF_1
GHI	3	mitochondrial	GHI_2, GHI_2
XYZ	1	protein	NULL

First Normal Form



Id*	GeneName	CodingTypeId	CodingType	TranscriptName
1	ABC	1	protein	ABC_1
2	ABC	1	protein	ABC_2
3	ABC	1	protein	ABC_3
4	DEF	2	noncoding	DEF_1
5	GHI	3	mitochondrial	GHI_1
6	GHI	3	mitochondrial	GHI_2
7	XYZ	1	protein	NULL

*potential to increase data redundancy



GeneName	CodingTypeId	Coding Type	TranscriptName
ABC	1	protein	ABC_1, ABC_2, ABC_3
DEF	2	noncoding	DEF_1
GHI	3	mitochondrial	GHI_2, GHI_2
XYZ	1	protein	NULL

Second Normal Form

Second Normal Form

- 1NF

- all non-key columns are dependent on the primary key

Second Normal Form

Second Normal Form

- 1NF
- all non-key columns are dependent on the primary key

Are all our non-key columns dependent on our primary key?

Id*	GeneName	CodingTypeId	CodingType	TranscriptName
1	ABC	1	protein	ABC_1
2	ABC	1	protein	ABC_2
3	ABC	1	protein	ABC_3
4	DEF	2	noncoding	DEF_1
5	GHI	3	mitochondrial	GHI_1
6	GHI	3	mitochondrial	GHI_2
7	XYZ	1	protein	NULL

Second Normal Form

Second Normal Form

- 1NF
- all non-key columns are dependent on the primary key
    - GeneName is not dependent on Id*
    - CodingTypeId is not dependent on Id*
    - CodingType is not dependent on Id*

Id*	GeneName	CodingTypeId	CodingType	TranscriptName
1	ABC	1	protein	ABC_1
2	ABC	1	protein	ABC_2
3	ABC	1	protein	ABC_3
4	DEF	2	noncoding	DEF_1
5	GHI	3	mitochondrial	GHI_1
6	GHI	3	mitochondrial	GHI_2
7	XYZ	1	protein	NULL

Second Normal Form

Id*	GeneName	CodingTypeId	CodingType	TranscriptName
1	ABC	1	protein	ABC_1
2	ABC	1	protein	ABC_2
3	ABC	1	protein	ABC_3
4	DEF	2	noncoding	DEF_1
5	GHI	3	mitochondrial	GHI_1
6	GHI	3	mitochondrial	GHI_2
7	XYZ	1	protein	NULL

Second Normal Form

GeneName*	CodingTypeId	CodingType
ABC	1	protein
DEF	2	noncoding
GHI	3	mitochondrial
XYZ	1	protein

foreign key

GeneName*	TranscriptName*
ABC	ABC_1
ABC	ABC_2
ABC	ABC_3
DEF	DEF_1
GHI	GHI_1
GHI	GHI_2

decomposition

Id*	GeneName	CodingTypeId	CodingType	TranscriptName
1	ABC	1	protein	ABC_1
2	ABC	1	protein	ABC_2
3	ABC	1	protein	ABC_3
4	DEF	2	noncoding	DEF_1
5	GHI	3	mitochondrial	GHI_1
6	GHI	3	mitochondrial	GHI_2
7	XYZ	1	protein	NULL

Second Normal Form

Second Normal Form

- 1NF
- all non-key columns are dependent on the primary key
- decomposition
    - create separate tables for values that apply to multiple records
    - relate tables with foreign keys

Third Normal Form

Third Normal Form

- 2NF
- eliminate transitive functional dependencies on the primary key
    - A dependent on B
    - B dependent on C
    - C is transitively dependent on A via B

Third Normal Form

Third Normal Form

- 2NF
- eliminate transitive functional dependencies on the primary key

Do we have any functional dependencies on our primary keys?

GeneName*	CodingTypeId	CodingType
ABC	1	protein
DEF	2	noncoding
GHI	3	mitochondrial
XYZ	1	protein

GeneName*	TranscriptName*
ABC	ABC_1
ABC	ABC_2
ABC	ABC_3
DEF	DEF_1
GHI	GHI_1
GHI	GHI_2

Third Normal Form

Third Normal Form

- 2NF
- eliminate transitive functional dependencies on the primary key

CodingType is dependent on CodingTypeId which is dependent on GeneName

GeneName*	CodingTypeId	CodingType
ABC	1	protein
DEF	2	noncoding
GHI	3	mitochondrial
XYZ	1	protein

GeneName*	TranscriptName*
ABC	ABC_1
ABC	ABC_2
ABC	ABC_3
DEF	DEF_1
GHI	GHI_1
GHI	GHI_2

Third Normal Form

foreign key

foreign key

CodingTypeId*	CodingType
1	protein
2	noncoding
3	mitochondrial

GeneName*	CodingTypeId
ABC	1
DEF	2
GHI	3
XYZ	1

GeneName*	TranscriptName*
ABC	ABC_1
ABC	ABC_2
ABC	ABC_3
DEF	DEF_1
GHI	GHI_1
GHI	GHI_2

decomposition

GeneName*	CodingTypeId	CodingType
ABC	1	protein
DEF	2	noncoding
GHI	3	mitochondrial
XYZ	1	protein

GeneName*	TranscriptName*
ABC	ABC_1
ABC	ABC_2
ABC	ABC_3
DEF	DEF_1
GHI	GHI_1
GHI	GHI_2

OREGON HEALTH & SCIENCE UNIVERSITY

Third Normal Form

Third Normal Form

- 2NF
- eliminate transitive functional dependencies on the primary key
- helps to maintain consistent one-to-many relationships

Advanced Normal Forms

Boyce-Codd (BCNF, 3.5 NF)

- eliminate multiple composite keys which overlap

Fourth Normal Form

- eliminate multi-valued dependencies
  - A related to B; A related to C; B is not related to A

Fifth Normal Form

- ensures all original relationships can be reconstructed

Benefits of Database Normalization

Improved data integrity

- no insert/update/delete anomalies

Decreased storage requirements

- no redundant storage of data

More efficient database querying

- smaller tables
- more directed searching

Downsides of Database Normalization

Downsides of Database Normalization

Complex database schemas

- lots of tables
- lots of relationships
- lots of foreign keys

Difficult to describe/communicate

- differ from your initial ER diagram
- upgrade/change schema

Complex queries

- large number of joins
- query time (particularly on large tables)

Database Design

Keep normalization in mind

- do not replicate data
- ensure data integrity

If you break a normalization rule

- know why you are breaking it
- do it for a good reason

Database Management Systems (DBMS)

Software that allows for the creation, definition, and manipulation of a database

- accommodate large data sets (storage and querying)
- data consistency and multiple concurrent users
- crash recovery, logging
- security and access control

Database Management Systems (DBMS)

Software that allows for the creation, definition, and manipulation of a database

- accommodate large data sets (storage and querying)
- data consistency and multiple concurrent users
- crash recovery, logging
- security and access control

When are relational databases not appropriate?

Your Relational Database

Have your relational database set up

- performed your requirements analysis
- generated your ER diagram
- created/loaded your database schema
- normalized for efficient storage
- indexed for efficient querying

OREGON HEALTH & SCIENCE UNIVERSITY

Your Relational Database

Have your relational database set up

- performed your requirements analysis
- generated your ER diagram
- created/loaded your database schema
- normalized for efficient storage
- indexed for efficient querying

Dramatic increase in data

- running out of storage space
- query slows down

Relational Database

Potential hardware solutions

Relational Database

Potential hardware solutions

- add more storage space
- replicate data across different servers
    - master/slave setup
- partition subsets of data across different servers
    - sharding

Relational Database

Potential database solutions

Relational Database

Potential database solutions

- de-normalize tables to increase query efficiency
- drop secondary indexes to increase loading efficiency
- pre-materialize popular queries (no longer real time)

Relational Database

When are relational databases not appropriate?

- volume of data
    - too big
    - too small

- data model/characteristics
    - does not fit into rows and tables
    - data sparsity
    - data variety
    - data velocity

NoSQL Databases

Termed coined in 2009

- open-sourced, distributed computing non-relational databases

Driven by the need of big data

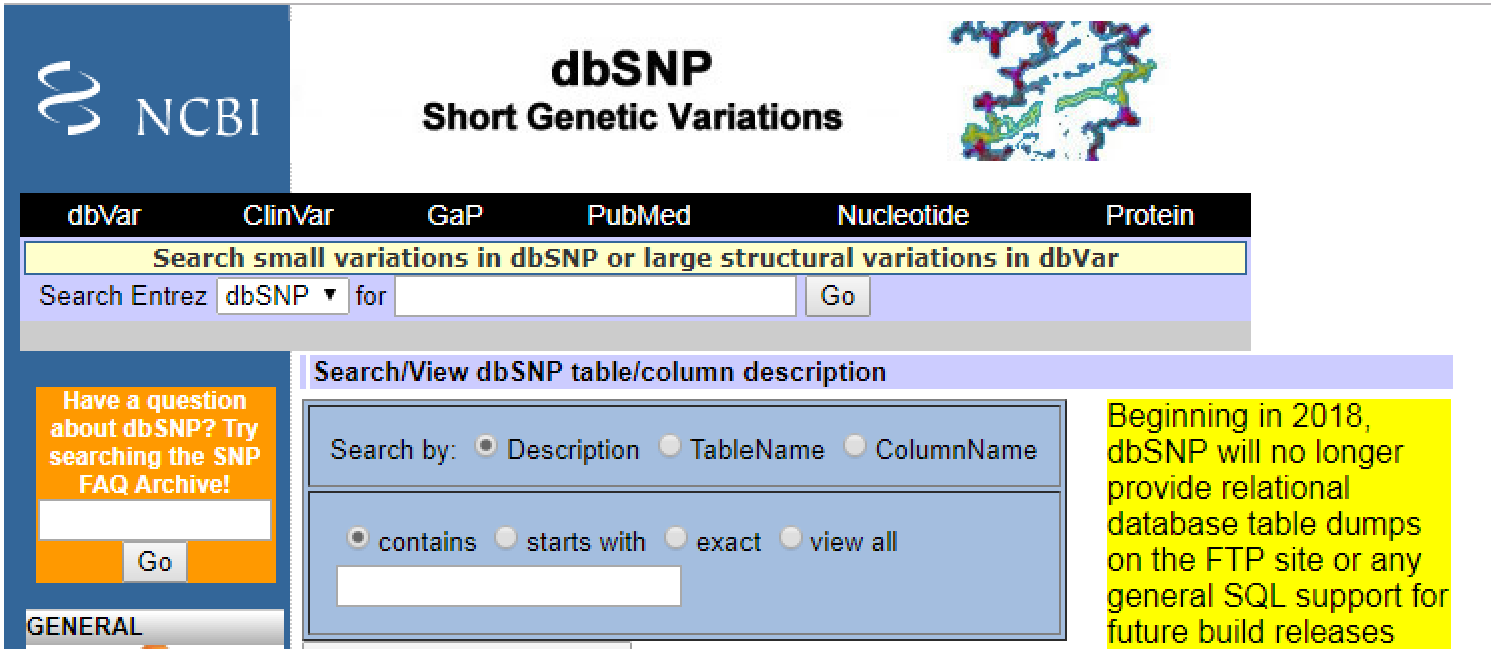
- volume, variety, velocity, …

NoSQL Databases

To accommodate big data, many

- works with clusters, distributed computing environments
- have no fixed/rigid database schemas
- do not support secondary indexes
- are not ACID compliant
- do not have structure query languages
    - simple command line/API-like interfaces

Bioinformatics Databases



dbSNP

Short Genetic Variations

Search small variations in dbSNP or large structural variations in dbVar

Search Entrez dbSNP ▾ for Go

Have a question about dbSNP? Try searching the SNP FAQ Archive!

Go

GENERAL

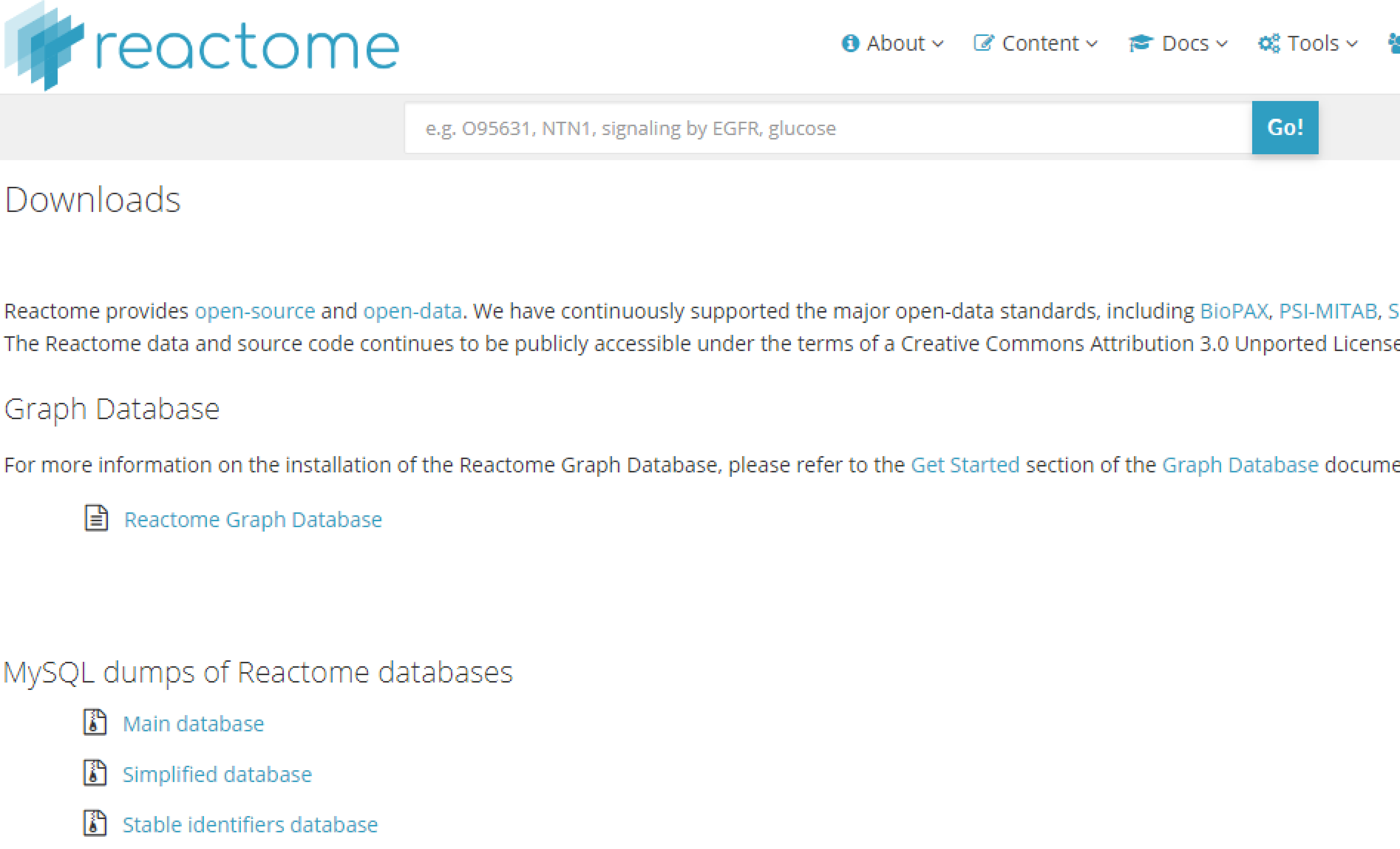
Search/View dbSNP table/column description

Search by: ● Description ○ TableName ○ ColumnName

● contains ○ starts with ○ exact ○ view all

Beginning in 2018, dbSNP will no longer provide relational database table dumps on the FTP site or any general SQL support for future build releases

Bioinformatics Databases



Downloads

Reactome provides open-source and open-data. We have continuously supported the major open-data standards, including BioPAX, PSI-MITAB, S
The Reactome data and source code continues to be publicly accessible under the terms of a Creative Commons Attribution 3.0 Unported License

Graph Database

For more information on the installation of the Reactome Graph Database, please refer to the Get Started section of the Graph Database docume

- Reactome Graph Database

MySQL dumps of Reactome databases

- Main database
- Simplified database
- Stable identifiers database