

Three-Valued Logic - AND

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

Three-Valued Logic - OR

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

Three-Valued Logic - NOT

NOT	TRUE
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

Comparisons with NULL Values

Each NULL is distinct, so comparisons with $<$, $>$, and $=$ don't make sense.

To compare with null, use SQL operator IS, e.g., "Which books don't have editors?":

```
SELECT * FROM book WHERE editor IS NULL;
```

Inner joins include only tuples for which the join condition evaluates to TRUE.

The IN Operator

```
mysql> select * from book where month in  
      ('April', 'July');
```

book_id	book_title	month	year	editor
1	CACM	April	1960	8
2	CACM	July	1974	8
3	BST	July	1948	2
7	AAAI	July	2012	9
8	NIPS	July	2012	9

5 rows in set (0.00 sec)

Nested Queries, a.k.a., Sub-Selects

List all the books published in the same month in which an issue of CACM was published.

```
mysql> select book_title, month
-> from book
-> where month in (select month
->                  from book
->                  where book_title =
->                        'CACM');
```

```
+-----+-----+
| book_title | month |
+-----+-----+
| CACM       | April |
| CACM       | July  |
| BST        | July  |
| AAAI       | July  |
| NIPS       | July  |
+-----+-----+
5 rows in set (0.00 sec)
```


Extended Example 1: Which dorms have fewer occupants than Caldwell?

Step 1: how many occupants in Caldwell?

```
mysql> select count(*) as caldwell_occupancy  
-> from dorm join student using(dorm_id)  
-> where dorm.name = 'caldwell';
```

```
+-----+  
| caldwell_occupancy |  
+-----+  
|                    4 |  
+-----+  
1 row in set (0.00 sec)
```

Occupancy Less than Caldwell

Now we use the previous "caldwell_{occupancy}" query as a subquery.

```
mysql> select dorm.name as dorm_name, count(*)
       as occupancy
       -> from dorm join student using (dorm_id)
       -> group by dorm_name
       -> having occupancy < (select count(*) as
                           caldwell_occupancy
                           ->
                           from dorm join
                           student using(dorm_id)
                           ->
                           where dorm.name =
                           'caldwell');
```

```
+-----+-----+
| dorm_name | occupancy |
+-----+-----+
| Armstrong |          3 |
| Brown     |          3 |
+-----+-----+
2 rows in set (0.00 sec)
```

Notice that we couldn't use a where clause here because occupancy is computed from a group, which isn't available at the WHERE stage of the

Extended Example 2: Which dorm has the highest average GPA?

- ▶ Step 1: Group students and their GPAs by dorm.
- ▶ Step 2: Get the average GPAs of each dorm.
- ▶ Step 3: Get the max avg GPA from step 2.

Step 1: Group students and their GPAs by dorm

```
mysql> select dorm.name as dorm_name,  
            student.name as student_name, gpa  
            -> from dorm join student using (dorm_id)  
            -> group by dorm_name, student_name, gpa;
```

dorm_name	student_name	gpa
Armstrong	Alice	3.6
Armstrong	Bob	2.7
Armstrong	Cheng	3.9
Brown	Dhruv	3.4
Brown	Ellie	4
Brown	Fong	2.3
Caldwell	Gerd	4
Caldwell	Hal	2.2
Caldwell	Isaac	2
Caldwell	Jacque	5

```
10 rows in set (0.00 sec)
```

Step 2: Get the average GPAs of each dorm.

```
mysql> select dorm.name as dorm_name, avg(gpa)
      as average_gpa
      -> from dorm join student using (dorm_id)
      -> group by dorm_name;
```

```
+-----+-----+
| dorm_name | average_gpa |
+-----+-----+
| Armstrong | 3.400000015894572 |
| Brown      | 3.23333333492279053 |
| Caldwell   | 3.3000000011920929 |
+-----+-----+
3 rows in set (0.00 sec)
```

Step 2.1 Formatting Numeric Values

```
mysql> select dorm.name as dorm_name,  
        format(avg(gpa), 2) as average_gpa  
        -> from dorm join student using (dorm_id)  
        -> group by dorm_name;
```

```
+-----+-----+  
| dorm_name | average_gpa |  
+-----+-----+  
| Armstrong | 3.40        |  
| Brown     | 3.23        |  
| Caldwell  | 3.30        |  
+-----+-----+  
3 rows in set (0.01 sec)
```

FORMAT(x,d[,locale])

- ▶ Formats the number *x* to *d* decimals using a format like 'nn,nnn.nnn' and returns the result as a string. If *d* is 0, the result has no decimal point or fractional part.
- ▶ *locale* defaults to the value of the *lc_time_names* system variable.

```
mysql> select @@lc_time_names;  
+-----+  
| @@lc_time_names |  
+-----+  
| en_US           |  
+-----+  
1 row in set (0.00 sec)
```

Step 3: Get max average gpa from average gpa results.

Using a nested query:

```
mysql> select dorm_name, max(average_gpa) as  
max_average_gpa  
-> from (select dorm.name as dorm_name,  
format(avg(gpa), 2) as average_gpa  
-> from dorm join student using  
(dorm_id)  
-> group by dorm_name) as avg_gpas;
```

```
+-----+-----+  
| dorm_name | max_average_gpa |  
+-----+-----+  
| Armstrong | 3.40            |  
+-----+-----+  
1 row in set (0.00 sec)
```


Semantic Constraints

The relational model can only encode:

- ▶ Domain constraints
- ▶ Key constraints
- ▶ Foreign key constraints

We call constraints on arbitrary values in tuples within and between relations **semantic constraints**.

While the relational model has no concept of semantic constraints, SQL can handle semantic constraints with **assertions** and **triggers**.

Assertions

```
CREATE ASSERTION <assertion-name>  
CHECK (<condition>)
```

<condition> can be any SQL statement that evaluates to TRUE, FALSE or UNKNOWN. Any databases INSERT or UPDATE that causes the <condition> to be FALSE is rejected by the database engine. While CREATE ASSERTION is part of the SQL standard, no major DBMS today (including MySQL) implements assertions.

Triggers

A trigger is a piece of SQL code associated with a table that executes when an event occurs on the table.

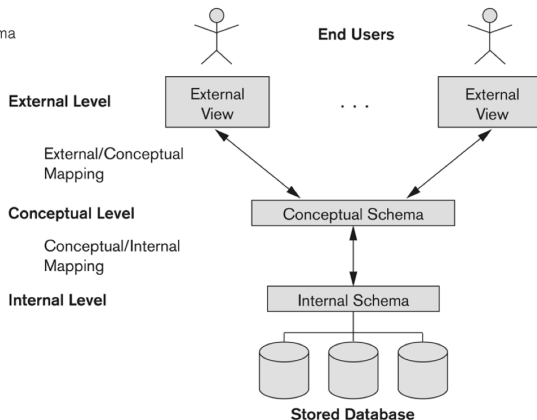
A trigger can't directly prevent an INSERT or UPDATE from occurring due to a semantic constraint violation, but a trigger can modify values being inserted or updated to avoid the violation or log the violation in a message table or execute a stored procedure or external program.

The Three-Schema Architecture

Remember the three-schema architecture?

Figure 2.2

The three-schema architecture.



Views

```
mysql> create view cacm_issues as
->   select * from book
->   where book_title = 'CACM';
Query OK, 0 rows affected (0.00 sec)
```

The CREATE VIEW statement is the mapping between the internal schema (base tables) and the external schema(s) (derived tables). They're all part of the database:

```
mysql> show tables;
+-----+
| Tables_in_pubs |
+-----+
| author          |
| author_pub      |
| book            |
| cacm_issues     |
| pub            |
+-----+
5 rows in set (0.00 sec)
```

A View is Like a Table

You can get data from the table:

```
mysql> select * from cacm_issues;
```

book_id	book_title	month	year	editor
1	CACM	April	1960	8
2	CACM	July	1974	8

2 rows in set (0.00 sec)

And you can update data in the view, which modifies the underlying base tables.

Users and Permissions

The value of external schemas is that they can give specific users customized vies of the database. We do this with permissions: