MySQL: Basics

Topics Covered

- MySQL Installation
- Creating Databases & Tables
- Data Types & Constraints
- Altering Tables
- Views

MySQL Installation

Why MySQL?

- Free, open-source
- Widely used in industry
- Excellent learning platform

Installation Resources

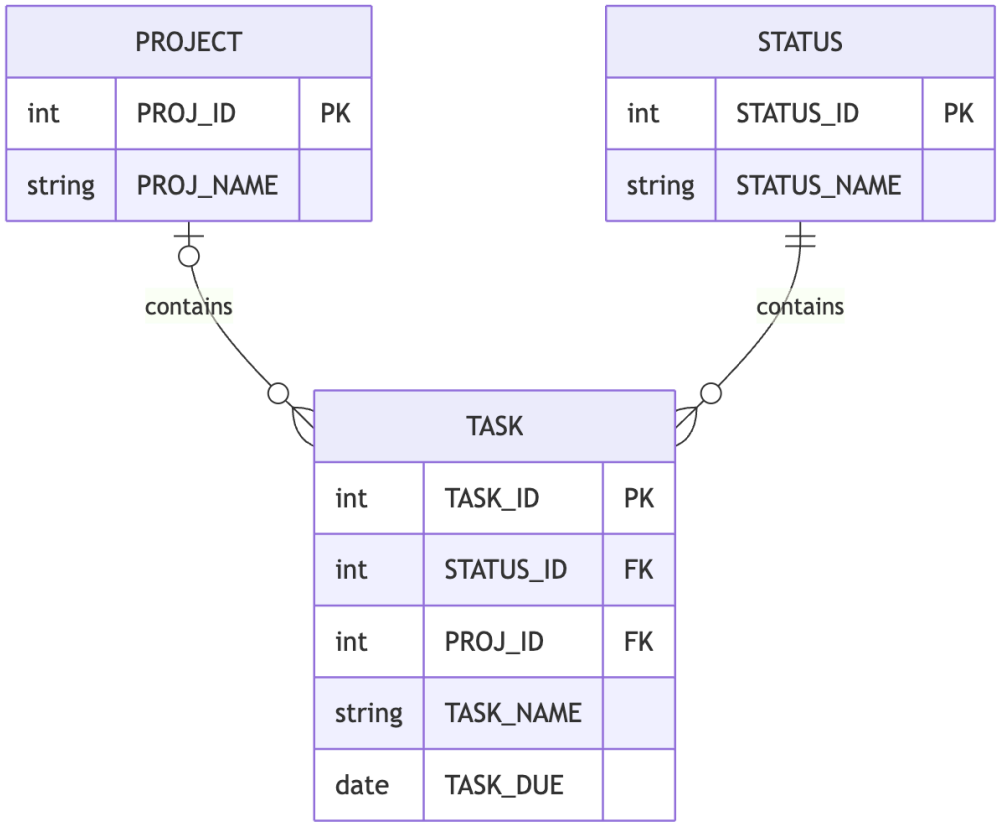
- Download: https://dev.mysql.com/downloads/installer/
- Tutorial: https://www.mysqltutorial.org

Getting Started

```
mysql -u root -v
```

Note: Shell is required; Workbench is optional

Example Model: Getting Things Done (GTD), Simply



STATUS_ID	STATUS_NAME
1	"Next"
2	"Waiting"
3	"Scheduled"
4	"Someday"

PROJ_ID	PROJ_NAME
1	"Teach CSD 331"
2	"Teach CS 143"
3	"Get better at pickleball"
4	"Plan summer vacation"

TASK_ID	TASK_NAME	TASK_DUE	STATUS_ID	PROJ_ID
1	"Buy milk"	null	1	null
2	"Prep next quiz"	04/15/2025	3	1
3	"Prep next quiz"	04/15/2025	3	2
4	"Amazon package"	null	2	3

Recommended Workflow

1. Write SQL commands in `.sql` script files
2. Make commands idempotent (use `IF NOT EXISTS`)
3. Execute in MySQL shell

```
mysql -u root -v
mysql> SOURCE /path/to/your/script.sql;
```

Creating a Database

```sql
-- Show existing databases
SHOW DATABASES;

-- Create new database
CREATE DATABASE IF NOT EXISTS gtd_db;

-- Select database to use
USE gtd_db;

-- Verify selected database
SELECT database();
```

Creating Tables: Basic Syntax

```
CREATE TABLE table_name(
    column1 datatype constraints,
    column2 datatype constraints,
    ...
    PRIMARY KEY ...,
    FOREIGN KEY ...,
    CONSTRAINT ...
);
```

Data Types: Numeric

Type	Description	Range
`INT`	4-byte signed integer	-2^{31} to $2^{31} - 1$
`INT UNSIGNED`	4-byte unsigned integer	0 to $2^{32} - 1$
`DECIMAL(P, D)`	Fixed precision	P: 1-65 digits, D: 0-30 decimals
`BOOLEAN`	Really TINYINT	1-byte integer

Variants: `SMALLINT`, `BIGINT`

Data Types: String & Date

String Types

- `CHAR(L)` : Fixed-length (L ≤ 255)
  - Use for: area codes, state abbreviations
- `VARCHAR(L)` : Variable-length (L ≤ 255)
  - Use for: names, descriptions

Date/Time Types

- `DATE` : CCYY-MM-DD format
- `TIME` : hh:mm:ss format
- `DATETIME` : CCYY-MM-DD hh:mm:ss format

Creating Tables: Order Matters!

Rule: Create Parent Tables First

- Start with tables that have no foreign keys
- Why? To avoid violating referential integrity

In our GTD model:

1. `status` (no FKs)
2. `project` (no FKs)
3. `task` (has FKs to status and project)

Example: Creating STATUS Table

```sql
-- Option 1: PK as column constraint
CREATE TABLE IF NOT EXISTS status(
    status_id INT UNSIGNED PRIMARY KEY,
    status_name VARCHAR(255) NOT NULL
);

-- Option 2: PK as table constraint (useful for composite keys)
CREATE TABLE IF NOT EXISTS status(
    status_id INT UNSIGNED,
    status_name VARCHAR(255) NOT NULL,
    PRIMARY KEY(status_id)
);

DESCRIBE status;
```

AUTO_INCREMENT Feature

```sql
-- Let the database assign unique IDs automatically
CREATE TABLE status(
    status_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    status_name VARCHAR(255) NOT NULL
);
```

Benefit: No need to manually track ID values

Example: Creating PROJECT Table

```
CREATE TABLE IF NOT EXISTS project(
    proj_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    proj_name VARCHAR(255) NOT NULL
);
```

Example: Creating TASK Table

```
CREATE TABLE task(
    task_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    task_name VARCHAR(255) NOT NULL,
    task_due DATE,
    status_id INT UNSIGNED NOT NULL,
    proj_id INT UNSIGNED,
    FOREIGN KEY(status_id)
        REFERENCES status(status_id),
    FOREIGN KEY(proj_id)
        REFERENCES project(proj_id)
);
```

View All Tables

```
SHOW TABLES;
```

Constraints Overview

Six types of constraints:

1. `PRIMARY KEY`
2. `FOREIGN KEY`
3. `UNIQUE`
4. `NOT NULL`
5. `DEFAULT`
6. `CHECK`

PRIMARY KEY Constraint

What it does:

- Makes column(s) **non-optional** (NOT NULL)
- Ensures values are **unique**
- Creates an **index** for efficient retrieval

Syntax Options:

```
-- Column constraint
status_id INT UNSIGNED PRIMARY KEY

-- Table constraint (for composite keys)
PRIMARY KEY(column1, column2)
```

PRIMARY KEY: ALTER Operations

```sql
-- Add a primary key
ALTER TABLE table_name
ADD PRIMARY KEY(column1, column2, ...);

-- Drop a primary key (not recommended!)
ALTER TABLE table_name
DROP PRIMARY KEY;
```

FOREIGN KEY Constraint

Purpose

- Specifies relationship to parent table's primary key
- Enforces **referential integrity**
- Database ensures FK refers to existing parent row

Syntax:

```
[CONSTRAINT constraint_name]
    FOREIGN KEY [foreign_key_name] (column_name, ...)
    REFERENCES parent_table(column_name,...)
    [ON DELETE reference_option]
    [ON UPDATE reference_option]
```

FOREIGN KEY: Reference Options

Option	Behavior	Use Case
`CASCADE`	Propagate change to child rows	Existence-dependent children
`SET NULL`	Set child FK to NULL	Optional parent relationship
`RESTRICT`	Prevent parent changes	**DEFAULT**

FOREIGN KEY: Examples

Referential Integrity Violation

```
-- This FAILS: status_id 100 doesn't exist
INSERT INTO task(task_name, status_id)
VALUES('play', 100);
```

FOREIGN KEY: RESTRICT (default)

```sql
-- This FAILS: can't delete parent with children
DELETE FROM status WHERE status_id = 1;

-- This FAILS: can't update parent PK with children
UPDATE status SET status_id = 100 WHERE status_id = 1;

-- This SUCCEEDS: non-key column update OK
UPDATE status SET status_name = 'NextAction' WHERE status_id = 1;
```

FOREIGN KEY: SET NULL

```
-- Show current constraint names
SHOW CREATE TABLE task;

-- Drop existing constraint
ALTER TABLE task DROP FOREIGN KEY `task_ibfk_2`;

-- Add new constraint with SET NULL
ALTER TABLE task
ADD CONSTRAINT `task_ibfk_2`
    FOREIGN KEY (proj_id) REFERENCES project(proj_id)
        ON DELETE SET NULL;
```

Now when a parent row is deleted, any referring FKs will be set to NULL

FOREIGN KEY: CASCADE

```
-- Add CASCADE on UPDATE
ALTER TABLE task DROP FOREIGN KEY `task_ibfk_2`;

ALTER TABLE task
ADD CONSTRAINT `task_ibfk_2`
    FOREIGN KEY (proj_id) REFERENCES project(proj_id)
        ON DELETE SET NULL
        ON UPDATE CASCADE;

-- Now this propagates to child rows
UPDATE project SET proj_id = 6 WHERE proj_name = 'Teach CSD 331';
```

UNIQUE Constraint

Properties

- Ensures no duplicate values in column(s)
- Uses index to enforce uniqueness
- NULL values treated as unique (multiple NULLs allowed)

Syntax:

```
-- Column constraint
phone VARCHAR(15) UNIQUE

-- Table constraint (for composite uniqueness)
CONSTRAINT c_name_address UNIQUE (name, address)
```

NOT NULL Constraint

Purpose

- Makes column mandatory
- Value must be provided

Syntax:

```
status_name VARCHAR(255) NOT NULL
```

Use for: Required/mandatory attributes

DEFAULT Constraint

Purpose

- Specifies default value for column
- Used when INSERT/UPDATE omits value

Syntax:

```
created_date DATE DEFAULT (CURDATE())
```

CHECK Constraint

Purpose

- Enforces boolean condition on row data

Syntax:

```
-- Column constraint
cost DECIMAL(10, 2) NOT NULL CHECK (cost >= 0),
price DECIMAL(10, 2) NOT NULL CHECK (price >= 0)

-- Table constraint (multi-column)
CONSTRAINT chk_price_gt_cost CHECK (price > cost)
```

Indexes

Purpose

- Faster data retrieval
- Enforce uniqueness

Syntax:

```
CREATE [UNIQUE] INDEX index_name
ON table_name(column1, column2, ...)
```

Examples:

```
-- Index for sorting by due date
CREATE INDEX idx_task_due_date ON task(task_due);

-- Index for searching by name
CREATE INDEX idx_task_name ON task(task_name);
```

Populating Tables: INSERT Syntax

```
INSERT INTO table_name(column_list)
VALUES
    (value_list_1),
    (value_list_2),
    ...
    (value_list_n);
```

Remember: Start with parent tables!

Populating Tables: Examples

```
-- Insert status values
INSERT INTO status(status_name)
VALUES ('Next'), ('Waiting'), ('Scheduled'), ('Someday');

SELECT * FROM status;

-- Insert projects
INSERT INTO project(proj_name)
VALUES
    ('Teach CSD 331'),
    ('Teach CS 143'),
    ('Get better at pickleball'),
    ('Plan summer vacation');
```

Populating TASK Table

```sql
INSERT INTO task(task_name, task_due, status_id, proj_id)
VALUES
    ('Buy milk', NULL, 1, NULL),
    ('Prep next quiz', '2025-04-15', 3, 1),
    ('Prep next quiz', '2025-04-18', 3, 2),
    ('Amazon package', NULL, 2, 3);

SELECT * FROM task;
```

Note: AUTO_INCREMENT handles task_id automatically

Altering Tables Overview

ALTER TABLE Command

Use with:

- `ADD` - Add columns/constraints
- `MODIFY` - Change column definition
- `CHANGE COLUMN` - Rename column
- `DROP` - Remove columns/constraints

Altering Tables: ADD Column

```sql
-- Add single column
ALTER TABLE vehicle
ADD model VARCHAR(100) NOT NULL;

-- Add multiple columns
ALTER TABLE vehicle
ADD color VARCHAR(50),
ADD note VARCHAR(255);
```

Altering Tables: MODIFY Column

```sql
-- Change column definition
ALTER TABLE vehicle
MODIFY note VARCHAR(100) NOT NULL;
```

Note: Can specify multiple MODIFY expressions

Warning: DBMS may have restrictions on modifications

Altering Tables: CHANGE COLUMN

```sql
-- Rename column
ALTER TABLE vehicle
CHANGE COLUMN note vehicle_condition VARCHAR(100) NOT NULL;
```

Altering Tables: DROP Column

```
ALTER TABLE vehicle
DROP COLUMN vehicle_condition;
```

⚠️ CAUTION

This deletes data permanently!

Altering Tables: ADD Constraints

```sql
-- Add primary key
ALTER TABLE table_name
ADD PRIMARY KEY (...);

-- Add foreign key
ALTER TABLE table_name
ADD FOREIGN KEY (...) REFERENCES ...;

-- Add check constraint
ALTER TABLE table_name
ADD CHECK (...);
```

DROP TABLE

```
DROP TABLE table_name;
```

⚠️ CAUTION

- **Deletes all data permanently!**
- DBMS enforces referential integrity
- Cannot drop parent table with FK references

Views: Overview

What is a View?

- **Virtual table** based on a SELECT query
- **Base tables**: Tables on which view is built
- A view can be a base table for another view!

Key Property

- **Dynamically updated**: SELECT query executes each time view is accessed

Creating a View

Syntax:

```
CREATE VIEW view_name [(column_list)]
AS SELECT query
```

Example: Overdue Tasks

```
CREATE VIEW overdue_tasks AS
    SELECT task_name, task_due
    FROM task
    WHERE task_due IS NOT NULL
        AND task_due < CURDATE();
```

Properties of Views

Usage

- View name can be used anywhere a table name is used
- Views are dynamically updated

Common Use Cases

1. **Access control**: Limit which columns/rows users can see
2. **Convenience**: Save commonly needed reports

Example:

```
-- Use view like a table
SELECT * FROM overdue_tasks;
```

Summary

Key Concepts Covered

✓ MySQL installation and setup
✓ Creating databases and tables
✓ Data types (numeric, string, date)
✓ Six types of constraints
✓ Foreign key reference options
✓ Indexes for performance
✓ Altering table structure
✓ Creating and using views

Best Practices Recap

1. **Write idempotent scripts** with `IF NOT EXISTS`
2. **Create parent tables first** (no FKs)
3. **Use AUTO_INCREMENT** for surrogate keys
4. **Choose appropriate FK reference options**
    - CASCADE for dependent children
    - SET NULL for optional relationships
    - RESTRICT when parent shouldn't change
5. **Be careful with DROP operations** - they delete data!

Resources

- MySQL Downloads: https://dev.mysql.com/downloads/installer/
- MySQL Tutorial: https://www.mysqltutorial.org
- Data Types Reference: https://www.mysqltutorial.org/mysql-basics/mysql-data-types/