



OVERVIEW & SETUP

InfluxDB powerful database solution
Install (Docker):
docker run -d --name mydb \
-e DB_PASSWORD=secret \
-p 5432:5432 influxdb:latest
Connect: host, port, user, password,
database name

INSTALLATION & CONFIG

Package manager install
brew install influxdb
Or download from official site
Config file: influxdb.conf or
similar
Set max connections, memory, storage
paths
Start service
systemctl start influxdb

DATA MODELING

Define schemas, tables/collections,
data types
-- Create table (SQL example):
CREATE TABLE users (
id SERIAL PRIMARY KEY,
name VARCHAR(100) NOT NULL,
email VARCHAR(200) UNIQUE NOT NULL
);
Normalize for consistency;
denormalize for performance

CRUD OPERATIONS

-- INSERT
INSERT INTO users (name, email) VALUES
('Alice', 'a@b.com');
-- SELECT
SELECT * FROM users WHERE active = true;
-- UPDATE
UPDATE users SET name = 'Bob' WHERE id =
1;
-- DELETE
DELETE FROM users WHERE id = 1;

QUERYING

```
SELECT u.*, o.total  
FROM users u  
JOIN orders o ON o.user_id = u.id  
WHERE o.status = 'paid'  
ORDER BY o.created_at DESC  
LIMIT 10 OFFSET 20;  
Use EXPLAIN/EXPLAIN ANALYZE to  
inspect query plans
```

INDEXING

```
CREATE INDEX idx_users_email ON  
users(email);  
CREATE UNIQUE INDEX  
idx_users_email_unique ON users(email);  
-- Composite index:  
CREATE INDEX idx_orders ON  
orders(user_id, status, created_at);  
Index columns used in WHERE, JOIN,  
ORDER BY clauses  
Too many indexes slow writes  
profile before adding
```

TRANSACTIONS

```
BEGIN;  
UPDATE accounts SET balance = balance -  
100 WHERE id = 1;  
UPDATE accounts SET balance = balance +  
100 WHERE id = 2;  
COMMIT;  
-- On error:  
ROLLBACK;  
ACID: Atomicity, Consistency,  
Isolation, Durability
```

REPLICATION & HA

Primary-replica replication for read
scaling
Primary: enable binary logging / WAL
Replica: point to primary host
Automatic failover with
Patroni/Sentinel/etc
Read from replicas; write to primary
only
Monitor replication lag alert if >
threshold

SECURITY

Create least-privilege database
users
CREATE USER appuser WITH PASSWORD
'secret';
GRANT SELECT, INSERT, UPDATE ON users TO
appuser;
Enable SSL/TLS for connections in
transit
Encrypt sensitive columns at
application layer
Regular security updates and patches

MONITORING & PERF

Monitor query performance,
connections, disk I/O
-- Slow query log
SET log_min_duration_statement = 1000;
-- 1s
EXPLAIN ANALYZE for individual query
tuning
Connection pooling: PgBouncer,
ProxySQL, HikariCP
Vacuum/ANALYZE (PostgreSQL) or
OPTIMIZE (MySQL)

CLI / TOOLS

```
psql -h host -U user -d database  
\l list databases \dt list tables  
\d users describe table structure  
-- Backup:  
pg_dump dbname > backup.sql  
-- Restore:  
psql dbname < backup.sql
```

COMMON GOTCHAS

Never store plaintext passwords use
bcrypt/Argon2
Use connection pooling raw
connections are expensive
Always parameterize queries prevent
SQL injection
Regular backups with point-in-time
recovery
Test failover procedures before
production incidents