ORM & SQL Interview Script - Complete Guide

Every ORM and SQL concept explained with: **Interviewer Question → Your Answer → Code Example → VibeVault Project Reference**

---

1. SQL BASICS

What is SQL?

Interviewer: "What is SQL?" **You:** "SQL (Structured Query Language) is the standard language for managing relational databases - creating, reading, updating, and deleting data."

```
-- The 4 main SQL operations (CRUD):
SELECT * FROM memories;              -- Read
INSERT INTO memories VALUES (...);   -- Create
UPDATE memories SET title = '..';    -- Update
DELETE FROM memories WHERE id = 1;   -- Delete
```

SELECT Queries

Interviewer: "Write a SELECT query." **You:** "SELECT retrieves data from tables with optional filtering, ordering, and limiting."

```
-- Basic SELECT
SELECT * FROM memories;

-- Select specific columns
SELECT id, title, created_at FROM memories;

-- With WHERE clause
SELECT * FROM memories WHERE user_id = 1;

-- Multiple conditions
SELECT * FROM memories
WHERE user_id = 1 AND analysis_status = 'done';

-- OR condition
SELECT * FROM memories
WHERE media_type = 'photo' OR media_type = 'audio';

-- LIKE for pattern matching
SELECT * FROM memories WHERE title LIKE '%vacation%';

-- ORDER BY
SELECT * FROM memories ORDER BY created_at DESC;

-- LIMIT
```

```sql
SELECT * FROM memories LIMIT 20;

-- OFFSET for pagination
SELECT * FROM memories LIMIT 20 OFFSET 40;  -- Page 3
```

INSERT, UPDATE, DELETE

Interviewer: "Show INSERT, UPDATE, DELETE statements." **You:** "These are the data manipulation commands."

```sql
-- INSERT
INSERT INTO memories (user_id, title, text, media_type)
VALUES (1, 'Beach Day', 'Had fun at the beach', 'text');

-- INSERT multiple rows
INSERT INTO memories (user_id, title, text) VALUES
(1, 'Memory 1', 'Text 1'),
(1, 'Memory 2', 'Text 2');

-- UPDATE
UPDATE memories
SET title = 'Updated Title', text = 'New content'
WHERE id = 5;

-- UPDATE with condition
UPDATE memories
SET analysis_status = 'pending'
WHERE analysis_status IS NULL;

-- DELETE
DELETE FROM memories WHERE id = 5;

-- DELETE with condition
DELETE FROM memories
WHERE user_id = 1 AND created_at < '2024-01-01';
```

JOINs

Interviewer: "Explain SQL JOINs." **You:** "JOINs combine rows from multiple tables based on related columns."

```sql
-- INNER JOIN - only matching rows
SELECT m.title, u.username
FROM memories m
INNER JOIN users u ON m.user_id = u.id;

-- LEFT JOIN - all from left, matching from right
SELECT u.username, COUNT(m.id) as memory_count
FROM users u
LEFT JOIN memories m ON u.id = m.user_id
GROUP BY u.username;
```

```sql
-- RIGHT JOIN - all from right, matching from left
SELECT m.title, u.username
FROM memories m
RIGHT JOIN users u ON m.user_id = u.id;

-- Self JOIN (e.g., for hierarchical data)
SELECT e.name, m.name as manager
FROM employees e
LEFT JOIN employees m ON e.manager_id = m.id;
```

In VibeVault: "I use JOINs to get memories with user information in a single query."

Aggregation Functions

Interviewer: "What are aggregate functions?" **You:** "Functions that operate on multiple rows to return a single value."

```sql
-- COUNT
SELECT COUNT(*) FROM memories WHERE user_id = 1;

-- SUM
SELECT SUM(word_count) FROM memories WHERE user_id = 1;

-- AVG
SELECT AVG(sentiment_score) FROM memories WHERE user_id = 1;

-- MAX, MIN
SELECT MAX(created_at), MIN(created_at) FROM memories;

-- GROUP BY
SELECT media_type, COUNT(*) as count
FROM memories
GROUP BY media_type;

-- HAVING (filter after grouping)
SELECT user_id, COUNT(*) as count
FROM memories
GROUP BY user_id
HAVING COUNT(*) > 10;
```

Subqueries

Interviewer: "What are subqueries?" **You:** "Queries nested inside another query."

```sql
-- Subquery in WHERE
SELECT * FROM memories
WHERE user_id = (
    SELECT id FROM users WHERE username = 'demo'
```

```
);

-- Subquery in SELECT
SELECT
    title,
    (SELECT username FROM users WHERE id = memories.user_id) as username
FROM memories;

-- Subquery in FROM
SELECT avg_count
FROM (
    SELECT user_id, COUNT(*) as avg_count
    FROM memories
    GROUP BY user_id
) as user_counts;

-- EXISTS
SELECT * FROM users u
WHERE EXISTS (
    SELECT 1 FROM memories WHERE user_id = u.id
);
```

Indexes

Interviewer: "What are indexes and when do you use them?" **You:** "Indexes speed up data retrieval but slow down writes. Use on frequently queried columns."

```
-- Create index
CREATE INDEX idx_memories_user_id ON memories(user_id);

-- Composite index
CREATE INDEX idx_memories_user_status
ON memories(user_id, analysis_status);

-- Unique index
CREATE UNIQUE INDEX idx_users_email ON users(email);

-- Check existing indexes
SHOW INDEX FROM memories;

-- Drop index
DROP INDEX idx_memories_user_id ON memories;
```

When to use indexes:

- Foreign keys (user_id)
- Columns in WHERE clauses
- Columns in ORDER BY
- Columns in JOINs

Primary Key vs Foreign Key

Interviewer: "Explain Primary Key and Foreign Key." **You:** "Primary Key uniquely identifies each row. Foreign Key references a Primary Key in another table."

```
-- Primary Key
CREATE TABLE users (
    id SERIAL PRIMARY KEY,  -- Auto-incrementing PK
    username VARCHAR(150) UNIQUE NOT NULL
);

-- Foreign Key
CREATE TABLE memories (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL,
    title VARCHAR(255),
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);

-- ON DELETE options:
-- CASCADE: Delete child rows when parent deleted
-- SET NULL: Set foreign key to NULL
-- RESTRICT: Prevent parent deletion if children exist
```

Normalization

Interviewer: "What is database normalization?" **You:** "Organizing data to reduce redundancy and improve integrity through normal forms."

```
1NF (First Normal Form):
- Each column has atomic values (no lists)
- Each row is unique

2NF (Second Normal Form):
- 1NF + no partial dependencies
- All non-key columns depend on entire primary key

3NF (Third Normal Form):
- 2NF + no transitive dependencies
- Non-key columns don't depend on other non-key columns

Example - Before normalization:
| user_id | username | memory_title | memory_text |
| 1       | john     | Beach        | Fun day     |
| 1       | john     | Park         | Nice walk   |  <- username repeated!

After normalization:
Users: {id, username}
Memories: {id, user_id, title, text}
```

ACID Properties

Interviewer: "What are ACID properties?" **You:** "Guarantees for database transactions."

```
A - Atomicity
    Transaction is all-or-nothing

C - Consistency
    Database stays valid after transaction

I - Isolation
    Concurrent transactions don't interfere

D - Durability
    Committed data survives crashes

-- Example transaction
BEGIN;
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
UPDATE accounts SET balance = balance + 100 WHERE id = 2;
COMMIT;  -- Both succeed or both rollback
```

---

2. DJANGO ORM BASICS

What is ORM?

Interviewer: "What is ORM?" **You:** "Object-Relational Mapping translates Python objects to database tables. No raw SQL needed."

```python
# Instead of SQL:
# SELECT * FROM memories WHERE user_id = 1;

# Django ORM:
Memory.objects.filter(user=user)

# ORM advantages:
# 1. Database-agnostic (PostgreSQL, MySQL, SQLite)
# 2. Python syntax
# 3. Automatic SQL injection protection
# 4. Migrations track schema changes
```

Model Definition

Interviewer: "How do you define a model?" **You:** "Inherit from models.Model and define fields."

```python
# VibeVault Memory model
from django.db import models
```

```python
from django.conf import settings

class Memory(models.Model):
    # Primary key auto-created as 'id'

    # Foreign Key
    user = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE,
        related_name='memories'
    )

    # CharField with max length
    title = models.CharField(max_length=255)

    # TextField for long text
    text = models.TextField()

    # Choices field
    MEDIA_TYPES = (
        ('text', 'Text'),
        ('photo', 'Photo'),
        ('audio', 'Audio'),
    )
    media_type = models.CharField(max_length=20, choices=MEDIA_TYPES)

    # File field
    media_file = models.FileField(upload_to='memories/%Y/%m/%d/', blank=True)

    # JSON field
    metrics_json = models.JSONField(default=dict)

    # Auto timestamps
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        ordering = ['-created_at']
        verbose_name_plural = 'memories'
```

QuerySet Methods

Interviewer: "What QuerySet methods do you use?" **You:** "filter, exclude, get, all, create, update, delete, and more."

```python
# all() - Get all records
Memory.objects.all()

# filter() - WHERE clause
Memory.objects.filter(user=user)
Memory.objects.filter(media_type='photo')
```

```python
Memory.objects.filter(created_at__year=2024)

# exclude() - NOT condition
Memory.objects.exclude(analysis_status='pending')

# get() - Single object (raises exception if not found)
memory = Memory.objects.get(id=5)

# get_or_create()
memory, created = Memory.objects.get_or_create(
    user=user, title='Untitled',
    defaults={'text': 'Default content'}
)

# first(), last()
Memory.objects.filter(user=user).first()

# exists()
Memory.objects.filter(user=user).exists()

# count()
Memory.objects.filter(user=user).count()

# order_by()
Memory.objects.order_by('-created_at')

# values() - Returns dictionaries
Memory.objects.values('id', 'title')

# values_list() - Returns tuples
Memory.objects.values_list('id', flat=True)
```

Field Lookups

Interviewer: "What are field lookups?" **You:** "Double underscore syntax for complex filters."

```python
# exact (default)
Memory.objects.filter(title__exact='Beach Day')
Memory.objects.filter(title='Beach Day')  # Same

# iexact - case insensitive
Memory.objects.filter(title__iexact='beach day')

# contains / icontains
Memory.objects.filter(title__contains='vacation')
Memory.objects.filter(title__icontains='vacation')  # Case insensitive

# startswith / endswith
Memory.objects.filter(title__startswith='Summer')
```

```python
# in - list of values
Memory.objects.filter(media_type__in=['photo', 'audio'])

# gt, gte, lt, lte - comparisons
Memory.objects.filter(created_at__gte=date(2024, 1, 1))

# range
Memory.objects.filter(created_at__range=(start_date, end_date))

# isnull
Memory.objects.filter(media_file__isnull=True)

# date lookups
Memory.objects.filter(created_at__year=2024)
Memory.objects.filter(created_at__month=12)
Memory.objects.filter(created_at__date=today)

# JSON field lookups
Memory.objects.filter(metrics_json__sentiment__label='positive')
```

In VibeVault: "I use `__icontains` for text search and `__gte` for date filtering."

Creating and Updating

Interviewer: "How do you create and update records?" **You:** "Using create(), save(), update(), and bulk operations."

```python
# create() - Create and save in one step
memory = Memory.objects.create(
    user=request.user,
    title='New Memory',
    text='Content here'
)

# Instantiate then save
memory = Memory(user=request.user, title='New')
memory.text = 'Content'
memory.save()

# update() - Bulk update
Memory.objects.filter(user=user).update(analysis_status='pending')

# Update single object
memory = Memory.objects.get(id=5)
memory.title = 'Updated Title'
memory.save()

# save with update_fields (optimized)
memory.title = 'New Title'
memory.save(update_fields=['title'])

# bulk_create() - Insert many at once
```

```
Memory.objects.bulk_create([
    Memory(user=user, title='Memory 1'),
    Memory(user=user, title='Memory 2'),
])

# bulk_update()
memories = Memory.objects.filter(user=user)
for m in memories:
    m.analysis_status = 'done'
Memory.objects.bulk_update(memories, ['analysis_status'])
```

Deleting

Interviewer: "How do you delete records?" **You:** "Using delete() on objects or querysets."

```
# Delete single object
memory = Memory.objects.get(id=5)
memory.delete()

# Delete multiple (bulk)
Memory.objects.filter(user=user, analysis_status='failed').delete()

# CASCADE delete - When user deleted, their memories are deleted
# (configured in ForeignKey)

# Soft delete pattern
class Memory(models.Model):
    is_deleted = models.BooleanField(default=False)

    def delete(self):
        self.is_deleted = True
        self.save()

    class Meta:
        # Exclude soft-deleted from default queries
        default_manager_name = 'active_objects'
```

Q Objects (Complex Queries)

Interviewer: "How do you do OR queries in Django ORM?" **You:** "Using Q objects for complex boolean logic."

```
from django.db.models import Q

# OR condition
Memory.objects.filter(
    Q(media_type='photo') | Q(media_type='audio')
)
# SQL: WHERE media_type = 'photo' OR media_type = 'audio'
```

```
# AND with OR
Memory.objects.filter(
    Q(user=user) & (Q(media_type='photo') | Q(media_type='audio'))
)

# NOT
Memory.objects.filter(~Q(analysis_status='pending'))

# Complex example
Memory.objects.filter(
    Q(title__icontains=query) | Q(text__icontains=query),
    user=user,
    analysis_status='done'
)
```

In VibeVault: "I use Q objects in my fallback text search to match multiple fields."

Annotations and Aggregations

Interviewer: "How do you do calculations in ORM?" **You:** "Using annotate() for per-row calculations and aggregate() for totals."

```
from django.db.models import Count, Sum, Avg, Max, Min, F

# aggregate() - Single result
Memory.objects.aggregate(
    total=Count('id'),
    avg_length=Avg('text__length')
)
# Returns: {'total': 150, 'avg_length': 250.5}

# annotate() - Per-row calculation
users = User.objects.annotate(
    memory_count=Count('memories')
)
for user in users:
    print(user.username, user.memory_count)

# F expressions - Reference other fields
Memory.objects.filter(
    updated_at__gt=F('created_at')
)

# Annotate with F
Memory.objects.annotate(
    days_old=F('created_at') - now()
)

# Complex annotation
from django.db.models import Case, When, IntegerField
```

```
Memory.objects.annotate(
    score=Case(
        When(Q(title__icontains=query), then=2),
        When(Q(text__icontains=query), then=1),
        default=0,
        output_field=IntegerField()
    )
).order_by('-score')
```

In VibeVault: "I use Case/When annotations for keyword search scoring."

---

select_related and prefetch_related

Interviewer: "How do you optimize database queries?" **You:** "Using select_related for ForeignKey and prefetch_related for ManyToMany to reduce N+1 queries."

```
# N+1 Problem (BAD)
memories = Memory.objects.all()
for memory in memories:
    print(memory.user.username)  # Hits database each iteration!

# select_related - JOIN in one query (ForeignKey/OneToOne)
memories = Memory.objects.select_related('user').all()
for memory in memories:
    print(memory.user.username)  # No extra queries!

# prefetch_related - Separate query for related objects
users = User.objects.prefetch_related('memories').all()
for user in users:
    for memory in user.memories.all():  # Already loaded
        print(memory.title)

# Combining both
Memory.objects.select_related('user')\
              .prefetch_related('tags')\
              .filter(analysis_status='done')
```

In VibeVault: "I use `select_related('user')` when listing memories to avoid N+1."

Transactions

Interviewer: "How do you handle transactions in Django?" **You:** "Using transaction.atomic() for atomic operations."

```
from django.db import transaction

# Decorator
@transaction.atomic
def create_user_with_memory(data):
    user = User.objects.create(**data)
    Memory.objects.create(user=user, title='Welcome')
```

```
    # If second fails, first is rolled back

# Context manager
def update_memories(user, new_status):
    with transaction.atomic():
        memories = Memory.objects.select_for_update().filter(user=user)
        for memory in memories:
            memory.status = new_status
            memory.save()
        # All updates committed together

# Savepoints
with transaction.atomic():
    Memory.objects.create(user=user, title='First')

    try:
        with transaction.atomic():
            Memory.objects.create(user=user, title='Second')
            raise Exception("Rollback this")
    except:
        pass  # Second rolled back, First still committed
```

Raw SQL

Interviewer: "When and how do you use raw SQL?" **You:** "Only when ORM can't express the query. Use parameterized queries for safety."

```
# raw() - Returns model instances
memories = Memory.objects.raw(
    'SELECT * FROM memories WHERE MATCH(text) AGAINST(%s)',
    [search_query]
)

# cursor() - For non-model queries
from django.db import connection

with connection.cursor() as cursor:
    cursor.execute(
        "SELECT COUNT(*), media_type FROM memories GROUP BY media_type"
    )
    results = cursor.fetchall()

# NEVER do this (SQL injection!)
# cursor.execute(f"SELECT * FROM memories WHERE id = {user_input}")

# Always parameterize
cursor.execute("SELECT * FROM memories WHERE id = %s", [user_input])
```

Model Relationships

Interviewer: "Explain model relationships." **You:** "OneToOne, ForeignKey (ManyToOne), and ManyToMany."

```
# ForeignKey - Many memories to one user
class Memory(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)

# OneToOneField - One profile per user
class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)

# ManyToManyField - Memory can have many tags, tags can have many memories
class Tag(models.Model):
    name = models.CharField(max_length=50)

class Memory(models.Model):
    tags = models.ManyToManyField(Tag, related_name='memories')

# Usage
memory.tags.add(tag)
memory.tags.remove(tag)
memory.tags.all()
tag.memories.all()
```

Custom Managers

Interviewer: "What are custom managers?" **You:** "Managers that provide custom queryset methods for reusability."

```
class AnalyzedMemoryManager(models.Manager):
    def get_queryset(self):
        return super().get_queryset().filter(analysis_status='done')

    def positive_sentiment(self):
        return self.get_queryset().filter(
            metrics_json__sentiment__label='positive'
        )

    def for_user(self, user):
        return self.get_queryset().filter(user=user)

class Memory(models.Model):
    # ... fields ...

    objects = models.Manager()  # Default manager
    analyzed = AnalyzedMemoryManager()  # Custom manager

# Usage
Memory.analyzed.all()  # Only analyzed memories
```

```
Memory.analyzed.positive_sentiment()  # Analyzed + positive
Memory.analyzed.for_user(request.user)  # Analyzed + user
```

Migrations

Interviewer: "How do migrations work?" **You:** "Migrations track model changes and apply them to the database."

```
# After changing models, create migration
python manage.py makemigrations

# Apply migrations
python manage.py migrate

# Show migration status
python manage.py showmigrations

# Reverse a migration
python manage.py migrate myapp 0005

# SQL preview
python manage.py sqlmigrate myapp 0006
```

```
# Custom migration (adding data)
from django.db import migrations

def create_default_tags(apps, schema_editor):
    Tag = apps.get_model('memories', 'Tag')
    Tag.objects.create(name='personal')
    Tag.objects.create(name='work')

class Migration(migrations.Migration):
    operations = [
        migrations.RunPython(create_default_tags),
    ]
```

---

3. ADVANCED ORM PATTERNS

Queryset Chaining

Interviewer: "Why is queryset chaining efficient?" **You:** "Querysets are lazy - they don't hit the database until evaluated."

```
# This doesn't execute any SQL yet
qs = Memory.objects.filter(user=user)
qs = qs.filter(analysis_status='done')
qs = qs.order_by('-created_at')
```

```
qs = qs[:20]

# SQL only executed when:
list(qs)           # Convert to list
for m in qs:       # Iterate
print(qs)          # Print
qs.exists()        # Check existence
len(qs)            # Get length
```

Deferred Fields

Interviewer: "How do you optimize large field loading?" **You:** "Using only() to load specific fields or defer() to skip fields."

```
# only() - Only load these fields
Memory.objects.only('id', 'title')
# SELECT id, title FROM memories

# defer() - Load all except these
Memory.objects.defer('metrics_json', 'embedding')
# Loads everything except heavy JSON/vector fields

# Chaining with other methods
Memory.objects.filter(user=user)\
              .only('id', 'title', 'created_at')\
              .order_by('-created_at')[:10]
```

In VibeVault: "I defer embedding field when listing memories - it's a large vector."

Database Functions

Interviewer: "Can you use database functions in ORM?" **You:** "Yes, Django provides database functions and you can create custom ones."

```
from django.db.models.functions import (
    Lower, Upper, Length, Concat, Coalesce,
    TruncMonth, ExtractYear
)

# String functions
Memory.objects.annotate(
    title_lower=Lower('title'),
    title_length=Length('title')
)

# Date functions
Memory.objects.annotate(
    month=TruncMonth('created_at'),
    year=ExtractYear('created_at')
```

```
).values('month').annotate(count=Count('id'))

# Coalesce - first non-null value
Memory.objects.annotate(
    display_text=Coalesce('text', 'title', Value('Untitled'))
)
```

Expression Wrappers

Interviewer: "How do you do complex expressions?" **You:** "Using ExpressionWrapper for type casting and complex calculations."

```
from django.db.models import ExpressionWrapper, F, DurationField

# Calculate age in days
Memory.objects.annotate(
    age=ExpressionWrapper(
        Now() - F('created_at'),
        output_field=DurationField()
    )
)

# Combined calculations
Memory.objects.annotate(
    engagement_score=ExpressionWrapper(
        F('view_count') * 2 + F('like_count') * 5,
        output_field=IntegerField()
    )
).order_by('-engagement_score')
```

---

4. PERFORMANCE TIPS

Query Optimization Checklist

```
# 1. Use select_related/prefetch_related
Memory.objects.select_related('user')

# 2. Use only()/defer() for large fields
Memory.objects.defer('embedding')

# 3. Use values()/values_list() when you don't need objects
Memory.objects.values('id', 'title')

# 4. Use exists() instead of count() > 0
if Memory.objects.filter(user=user).exists():  # Better
if Memory.objects.filter(user=user).count() > 0:  # Worse
```

```python
# 5. Use update() for bulk updates
Memory.objects.filter(status='old').update(status='new')  # One query
# vs
for m in Memory.objects.filter(status='old'):
    m.status = 'new'
    m.save()  # N queries!

# 6. Use bulk_create() for many inserts
Memory.objects.bulk_create(memories_list)

# 7. Index frequently filtered columns
class Memory(models.Model):
    user = models.ForeignKey(...)  # Auto-indexed
    analysis_status = models.CharField(... db_index=True)  # Add index

    class Meta:
        indexes = [
            models.Index(fields=['user', 'created_at']),
        ]

# 8. Use iterator() for large querysets
for memory in Memory.objects.all().iterator():
    process(memory)  # Doesn't load all into memory
```

Debugging Queries

Interviewer: "How do you debug slow queries?" **You:** "Using query logging, explain, and django-debug-toolbar."

```python
# Print actual SQL
qs = Memory.objects.filter(user=user)
print(qs.query)

# Enable query logging in settings
LOGGING = {
    'loggers': {
        'django.db.backends': {
            'level': 'DEBUG',
        }
    }
}

# Count queries in view
from django.db import connection
def my_view(request):
    # ... do stuff ...
    print(len(connection.queries))

# EXPLAIN query
from django.db import connection
with connection.cursor() as cursor:
```

```
    cursor.execute("EXPLAIN SELECT * FROM memories WHERE user_id = 1")
    print(cursor.fetchall())
```

QUICK REFERENCE

SQL ↔ ORM Mapping

SQL	Django ORM
`SELECT * FROM memories`	`Memory.objects.all()`
`WHERE id = 5`	`.get(id=5)` or `.filter(id=5)`
`WHERE title LIKE '%beach%'`	`.filter(title__icontains='beach')`
`WHERE status IN (...)`	`.filter(status__in=[...])`
`ORDER BY created_at DESC`	`.order_by('-created_at')`
`LIMIT 10`	`[:10]`
`COUNT(*)`	`.count()`
`GROUP BY status`	`.values('status').annotate(...)`
`INNER JOIN`	`.select_related()`
`INSERT INTO`	`.create()`
`UPDATE ... SET`	`.update()`
`DELETE FROM`	`.delete()`