🔹 1. Consultas básicas

Obtener todos los objetos

python

```python
Concurso.objects.all()
```

`SELECT * FROM concurso;`

Filtrar resultados

python

```python
Concurso.objects.filter(activo=True)
```

`WHERE activo = TRUE`

Excluir registros

python

```python
Concurso.objects.exclude(activo=True)
```

`WHERE NOT activo = TRUE`

Buscar por múltiples condiciones (`Q`)

python

```python
from django.db.models import Q

Concurso.objects.filter(Q(activo=True) | Q(nombre__contains="Premio"))
```

`WHERE activo = TRUE OR nombre LIKE '%Premio%'`

Ordenar resultados

python

```python
Concurso.objects.order_by("fecha_inicio")        # ASC
Concurso.objects.order_by("-fecha_inicio")       # DESC
```

`ORDER BY fecha_inicio ASC / DESC`

Limitar resultados

python

```python
Participante.objects.order_by("-fecha_inscripcion")[:1]
```

`ORDER BY fecha_inscripcion DESC LIMIT 1`

🔹 2. Búsquedas por texto y fecha

python

```python
Concurso.objects.filter(descripcion__contains="texto")
Concurso.objects.filter(fecha_inicio__year=2025, fecha_inicio__month=11)
```

🟰

sql

```sql
WHERE descripcion LIKE '%texto%'
AND EXTRACT(YEAR FROM fecha_inicio) = 2025
```

🔹 3. Consultas con relaciones

`select_related()` **(JOINs 1:1 o ForeignKey)**

Optimiza relaciones *uno a uno* o *muchos a uno*.

python

```python
Concurso.objects.select_related("creador__usuario", "ganador__usuario")
```

🟰 `JOIN` en SQL.

Ideal para evitar consultas extras.

`prefetch_related()` (relaciones inversas o N:N)

Optimiza relaciones *uno a muchos* o *muchos a muchos*.

```python
Concurso.objects.prefetch_related("participantes__usuario")
```

🟰 Ejecuta una segunda query y las une en memoria (evita el N+1 problem).

🔹 7. Anotaciones y HAVING (`annotate()` + `filter()`)

Para calcular valores **por cada grupo**.

```python
from django.db.models import Avg, Count

Concurso.objects.annotate(
    media_votos=Avg("votar_movil__puntuacion")
).filter(media_votos__lt=3)
```

🟰

```sql
SELECT concurso.*, AVG(puntuacion) AS media_votos
FROM votar
GROUP BY concurso.id
HAVING AVG(puntuacion) < 3;
```

🔹 8. Comparaciones de campos (`F()`)

```python
from django.db.models import F

Concurso.objects.filter(fecha_fin__lt=F("fecha_inicio"))
```

🟰 `WHERE fecha_fin < fecha_inicio`

Operador	Django ORM	SQL equivalente
=	field=value	=
!=	~Q(field=value)	<>
<	field__lt=value	<
<=	field__lte=value	<=
>	field__gt=value	>
>=	field__gte=value	>=
IN	field__in=[1,2,3]	IN (...)
IS NULL	field=None	IS NULL

```
Usuario.objets
.all()
.filter( edad__gt = 18 )
```

```python
# Una url que me muestre todos los Concursos y sus datos, incluido los relacionados.
def concursos_listar(request):

    concursos = (
        Concurso.objects
        .select_related(
            "creador__usuario",  # anidado
            "ganador__usuario",  # Asi también accedes a su usuario
        )
        .prefetch_related("participantes__usuario")  # Relacion N:N en participantes
    )
    concursos.all()

    """
    concursos = (Concurso.objects.raw(
    "SELECT * FROM Concursos_Online_concurso co "
    + " JOIN Concursos_Online_administrador ad ON co.creador_id = ad.id "
    + " JOIN Concursos_Online_participante pa ON co.ganador_id = pa.id "
    + " JOIN Concursos_Online_inscribe ins ON co.id = ins.concurso_id "
    + " JOIN Concursos_Online_participante p ON p.id = ins.participante_id"
    ))
    """

    return render(request,'Concursos_Online/lista_Concurso.html',{'Concursos_Mostrar':concursos})
```

```python
# Una url que me muestre un Concurso y sus datos, incluido los relacionados.
def dame_concurso(request, id_concurso):

    concurso = (
        Concurso.objects
        .select_related(
            "creador__usuario",  # anidado
            "ganador__usuario",  # Asi también accedes a su usuario
        )
        .prefetch_related("participantes__usuario")  # Relacion N:N en participantes
        .get(id=id_concurso) # Obtiene el concurso con el ID especificado
    )

    """
    concurso = (Concurso.objects.raw(
    "SELECT * FROM Concursos_Online_concurso co "
    + " JOIN Concursos_Online_administrador ad ON co.creador_id = ad.id "
    + " JOIN Concursos_Online_participante pa ON co.ganador_id = pa.id "
    + " JOIN Concursos_Online_inscribe ins ON co.id = ins.concurso_id "
    + " JOIN Concursos_Online_participante p ON p.id = ins.participante_id"
    + " WHERE co.id = %s",[id_concurso])[0]
    )
    """

    return render(request,'Concursos_Online/Concurso.html',{'Concurso_Mostrar':concurso})
```

```python
# Una url que muestre los Concursos que comienzan en un año y mes concreto
def dame_concursos_fecha(request, anyo_concurso, mes_concurso):

    concursos = (
        Concurso.objects
        .select_related(
            "creador__usuario",  # anidado
            "ganador__usuario",  # Asi también accedes a su usuario
        )
        .prefetch_related("participantes__usuario")  # Relacion N:N en participantes
    )
    concursos = concursos.filter(fecha_inicio__year=anyo_concurso, fecha_inicio__month=mes_concurso)
    concursos.all()

    """
    # Convertir el mes a cadena y asegurar que tenga 2 digitos con relleno de cero
    mes_formato_sql = str(mes_concurso).zfill(2)

    concursos = (Concurso.objects.raw(
    "SELECT * FROM Concursos_Online_concurso co "
    + " JOIN Concursos_Online_administrador ad ON co.creador_id = ad.id "
    + " JOIN Concursos_Online_participante pa ON co.ganador_id = pa.id "
    + " JOIN Concursos_Online_inscribe ins ON co.id = ins.concurso_id "
    + " JOIN Concursos_Online_participante p ON p.id = ins.participante_id"
    + " WHERE strftime('%%Y', co.fecha_inicio) = %s "
    + " AND strftime('%%m', co.fecha_inicio) = %s "
    ,[str(anyo_concurso),mes_formato_sql] # Usamos la variable formateada
    ))
    """

    return render(request,'Concursos_Online/lista_Concurso.html',{'Concursos_Mostrar':concursos})
```

```python
# Una url que:
# Si pones "true" en la URL, solo ves los concursos activos;
# Si pones "false", ves todos los concursos (activos e inactivos);
# Y siempre están ordenados por fecha de inicio.
def dame_concurso_activo(request, activo):

    # Variable para transformar el str de la url a bolean
    is_active = (str(activo).lower() == 'true')

    concursos = (
        Concurso.objects
        .select_related(
            "creador__usuario",  # anidado
            "ganador__usuario",  # Asi también accedes a su usuario
        )
        .prefetch_related("participantes__usuario")  # Relacion N:N en participantes
    )
    concursos = concursos.filter(Q(activo=is_active)|Q(activo=True)).order_by("fecha_inicio")
    concursos.all()

    """
    concursos = (Concurso.objects.raw(
    "SELECT * FROM Concursos_Online_concurso co "
    + " JOIN Concursos_Online_administrador ad ON co.creador_id = ad.id "
    + " JOIN Concursos_Online_participante pa ON co.ganador_id = pa.id "
    + " JOIN Concursos_Online_inscribe ins ON co.id = ins.concurso_id "
    + " JOIN Concursos_Online_participante p ON p.id = ins.participante_id"
    + " WHERE co.activo = %s "
    + " OR co.activo = True "
    + " ORDER BY co.fecha_inicio "
    ,[is_active] # Usamos la variable activo
    ))
    """

    return render(request,'Concursos_Online/lista_Concurso.html',{'Concursos_Mostrar':concursos})
```

```python
# Una url que:
# Lista los concursos que tienen el texto especificado en su descripción.
# Los concursos resultantes están ordenados de forma descendente (de Z a A) según el nombre.
def dame_concurso_texto(request, texto):

    concursos = (
        Concurso.objects
        .select_related(
            "creador__usuario",  # anidado
            "ganador__usuario",  # Asi también accedes a su usuario
        )
        .prefetch_related("participantes__usuario")  # Relacion N:N en participantes
    )
    concursos = concursos.filter(descripcion__contains=texto).order_by("-nombre")
    concursos.all()

    """
    # 1. Prepara el parámetro para la búsqueda de subcadena (LIKE '%%')
    texto_con_comodines = '%' + texto + '%'

    concursos = (Concurso.objects.raw(
    "SELECT * FROM Concursos_Online_concurso co "
    + " JOIN Concursos_Online_administrador ad ON co.creador_id = ad.id "
    + " JOIN Concursos_Online_participante pa ON co.ganador_id = pa.id "
    + " JOIN Concursos_Online_inscribe ins ON co.id = ins.concurso_id "
    + " JOIN Concursos_Online_participante p ON p.id = ins.participante_id "
    + " WHERE co.descripcion LIKE %s "
    + " ORDER BY co.nombre DESC "
    ,[texto_con_comodines] # Usamos la variable texto_con_comodines
    ))
    """

    return render(request,'Concursos_Online/lista_Concurso.html',{'Concursos_Mostrar':concursos})
```

```python
# Una url que permite ver el participante que se inscribió más recientemente en un concurso concreto, utilizando el ID del concurso.
# Muestra únicamente la información de ese último inscrito, limitando la consulta a un solo registro
def dame_ultimo_participante(request, id_concurso):

    participante_a_mostrar = (
        Participante.objects
        .filter(inscribe_participante__concurso_id = id_concurso)
        .order_by("-inscribe_participante__fecha_inscripcion")[:1].get()
    )

    """
    participante_raw = (Participante.objects.raw(
    "SELECT * FROM Concursos_Online_participante pa "
    + " JOIN Concursos_Online_inscribe ins ON pa.id = ins.participante_id "
    + " WHERE ins.concurso_id = %s "
    + " ORDER BY ins.fecha_inscripcion DESC "
    + " LIMIT 1 "
    ,[id_concurso]
    )[0])

    """
    return render(request,'Concursos_Online/Participante.html',{'Participante_Mostrar':participante_a_mostrar})
```

```python
# Una url que permite obtener información sobre un Participante en concreto, buscando por su alias.
def detalle_participante_alias(request, alias_participante):

    participante_a_mostrar = (
        Participante.objects
        .get(alias=alias_participante)
    )

    """
    participante_a_mostrar = (Participante.objects.raw(
    "SELECT * FROM Concursos_Online_participante pa "
    + " WHERE pa.alias = %s "
    ,[alias_participante]
    )[0])
    """

    return render(request,'Concursos_Online/Participante.html',{'Participante_Mostrar':participante_a_mostrar})
```

```python
# Una url que obtiene todos los Usuarios que nunca han recibido una Notificación.
def usuarios_sin_notificar(request):

    usuarios_no_notificados = (
        Usuario.objects
        .filter(recibos=None)
    )
    usuarios_no_notificados.all()

    """
    usuarios_no_notificados = (Usuario.objects.raw(
    "SELECT * FROM Concursos_Online_usuario user "
    + " LEFT JOIN Concursos_Online_recibe re ON re.usuario_id = user.id "
    + " WHERE re.usuario_id IS NULL "
    ))
    """

    return render(request,
'Concursos_Online/no_notificados.html',{'Usuario_Mostrar':usuarios_no_notificados})
```

```python
# Una url que obtiene todos los objetos Jurado.
def dame_jurados(request):

    jurados = (
        Jurado.objects
        .select_related('usuario')
        .prefetch_related(Prefetch('concursos'))
    )
    jurados.all()

    """
    jurados = (Jurado.objects.raw(
    "SELECT * FROM Concursos_Online_jurado ju "
    + " JOIN Concursos_Online_usuario user ON user.id = ju.usuario_id"
    + " LEFT JOIN Concursos_Online_asigna asig ON asig.jurado_id = ju.id "
    + " LEFT JOIN Concursos_Online_concurso co ON asig.concurso_id = co.id "
    ))
    """
    return render(request, 'Concursos_Online/lista_Jurados.html', {'Jurados_Mostrar':jurados})
```

```python
# Una url que calcula y muestra las métricas de agregación (media, máximo y mínimo) del campo experiencia de todos los Jurados.
def metricas_experiencia_jurados(request):

   metricas_objeto = (
       Jurado.objects
       .aggregate(
           media_experiencia=Avg('experiencia'),
           max_experiencia=Max('experiencia'),
           min_experiencia=Min('experiencia')
       )
   )

   """
   metricas_queryset = (Jurado.objects.raw(
   "SELECT 1 AS id, AVG(experiencia) AS media_experiencia, MAX(experiencia) AS max_experiencia, MIN(experiencia) AS min_experiencia FROM Concursos_Online_jurado"
   )[0])

   """
   return render(request, 'Concursos_Online/metricas_Jurados.html', {'Metricas_Mostrar':metricas_objeto})
```

```python
"""
   2-Crea una URL que muestre una lista de todos los proyectos de la aplicación
   con sus datos correspondientes.
"""

def listar_proyectos(request):

   proyectos = (Proyecto.objects.select_related("creador")
                .prefetch_related("colaboradores",Prefetch("proyecto_tareas"))
               ).all()
   return render(request, "proyecto/lista.html", {"proyectos":proyectos})
```

```python
"""
   3-Crear una URL que muestre todas las tareas que están asociadas a un proyecto,
   ordenadas por fecha de creación descendente.
"""
def listar_tareas_proyecto(request,proyecto_id):
   proyecto_mostrar = Proyecto.objects.get(id=proyecto_id)

   tareas = (Tarea.objects.select_related("creador","proyecto").
             prefetch_related("usuarios_asignados",Prefetch("etiquetas_tareas"),
                              Prefetch("comentarios_tarea"),
                              Prefetch("comentarios_tarea__autor")
                              )
            )
   tareas = tareas.filter(proyecto=proyecto_id).order_by("-fecha_creacion").all()

   return render(request, "tarea/lista.html", {"tareas":tareas, "proyecto":proyecto_mostrar})
```

```python
"""
  4- Crear una URL que muestre todos los usuarios que están asignados a una tarea ordenados
  por la fecha de asignación de la tarea de forma ascendente.
"""
def listar_usuarios_tarea(request,tarea_id):
   tarea = Tarea.objects.get(id=tarea_id)
   #Version Corta
   """usuarios = (Usuario.objects
                      .filter(asignaciontarea__tarea=tarea_id)
                      .order_by("asignaciontarea__fecha_asignacion")
   ).all()"""

   #Version Larga

   usuarios = (Usuario.objects.prefetch_related(
                           Prefetch("creador_proyecto"),
                           Prefetch("colaboradores_proyecto"),
                           Prefetch("creador_tarea"),
                           Prefetch("colaboradores_tarea"),
                           Prefetch("comentarios_creador"),
                              )
                      .filter(asignaciontarea__tarea=tarea_id)
                      .order_by("asignaciontarea__fecha_asignacion")
   ).all()


   return render(request, "usuario/lista_completa.html", {"usuarios":usuarios,"tarea":tarea})
```

```python
"""
   5-Crear una URL que muestre todas las tareas que tengan un texto en concreto en las observaciones
   a la hora de asignarlas a un usuario.
"""
def listar_tareas_texto_usuario(request,usuario_id,texto):
   usuario = Usuario.objects.get(id=usuario_id)
   tareas = (Tarea.objects.select_related("creador","proyecto").
             prefetch_related("usuarios_asignados",Prefetch("etiquetas_tareas"),
                              Prefetch("comentarios_tarea"),
                              Prefetch("comentarios_tarea__autor")
                               )
   ).filter(asignaciontarea__observaciones__contains=texto,asignaciontarea__usuario=usuario_id).all()
   return render(request, "tarea/lista_filtro_usuario.html", {"tareas":tareas, "usuario": usuario})
```

```python
"""
   6-Crear una URL que muestre todos las tareas que se han creado entre dos años
   y el estado sea "Completada".
"""
def listar_tareas_anyos(request,anyo_desde,anyo_hasta):
   tareas = (Tarea.objects.select_related("creador","proyecto").
             prefetch_related("usuarios_asignados",Prefetch("etiquetas_tareas"),
                              Prefetch("comentarios_tarea"),
                              Prefetch("comentarios_tarea__autor")
                              )
   ).filter(fecha_creacion__year__gte=anyo_desde,fecha_creacion__year__lte=anyo_hasta,estado='Co')
   return render(request, "tarea/lista.html", {"tareas":tareas})
```

```python
"""
   7-Crear una URL que obtenga el último usuario que ha comentado en una tarea de un proyecto en concreto.
"""
def ultimo_comentario_proyecto(request,proyecto_id):
   comentario = (Comentario.objects.select_related("autor","tarea").
                  prefetch_related(Prefetch("tarea__proyecto"))
                 .filter(tarea__proyecto=proyecto_id)
                 .order_by("-fecha_comentario")[0:1].get()
   )
   usuario = comentario.autor


   """usuario = (Usuario.objects.filter(comentarios_creador__tarea__proyecto=proyecto_id).
               order_by("-comentarios_creador__fecha_comentario")[:1].get()
            )"""

   return render(request, "usuario/usuario.html", {"usuario":usuario})
```

```python
"""
   8-Crear una URL que obtenga todos los comentarios de una tarea que empiecen
   por la palabra que se pase en la URL y que el año del comentario sea uno en concreto.
"""
def listar_comentarios_filtro(request,tarea_id,anyo,texto):
   tarea = Tarea.objects.get(id=tarea_id)

   comentarios = (Comentario.objects.select_related("autor")
                  .filter(tarea=tarea_id)
                  .filter(fecha_comentario__year=anyo)
                  .filter(contenido__startswith=texto)
   ).all()

   return render(request, "comentario/lista.html", {"tarea":tarea,"comentarios":comentarios})
```

```python
"""
   9-Crear una URL que obtenga todas las etiquetas que se han
   usado en todas las tareas de un proyecto.
"""
def listar_etiquetas_proyecto(request,proyecto_id):
   proyecto = Proyecto.objects.get(id=proyecto_id)

   etiquetas = (Etiqueta.objects.prefetch_related("tarea")
                .filter(tarea__proyecto=proyecto_id)
   ).distinct().all()

   return render(request, "etiqueta/lista.html", {"proyecto":proyecto,"etiquetas":etiquetas})
```

```python
"""
   10-Crear una URL que muestre todos los usuarios que no están asignados a una tarea.
"""
def usuarios_no_asignados(request):
   usuarios = (Usuario.objects.prefetch_related(
                           Prefetch("creador_proyecto"),
                           Prefetch("colaboradores_proyecto"),
                           Prefetch("creador_tarea"),
                           Prefetch("comentarios_creador"),
                              )
                       .filter(asignaciontarea=None)
   ).all()
   return render(request, "usuario/lista_completa_no_asignados.html", {"usuarios":usuarios})
```