`rm db.sqlite3` Fresh dev DB
`source venv/bin/activate`

`python manage.py makemigrations contracts` - keep the python model definitions and the underlying tables in sync / Django records that change in a new migration

`python manage.py migrate` so your database picks up the new schema.

`python manage.py loaddata base_contract` tells django to lead the fixture

SERVICES

```
 amendment_rates = AmendmentRate.objects.filter(
        contract=contract, effective_date__lte=query_date
    ).order_by("-effective_date", "-id")
```

```
    # Walk amendments from newest to oldest until we find the
service rate
    if amendment_rates:
        for amendment_rate in amendment_rates:
            amendment_rate_map = {
                k.strip().lower(): v for k, v in
amendment_rate.rates.items()
            }
            if normalized_service_type in amendment_rate_map:
                return
Decimal(str(amendment_rate_map[normalized_service_type]))
```

TESTS

```
def setUp(self):
        self.contract = Contract.objects.create(
            contract_id="CTR-001", effective_date=date(2024, 1,
1)
```

```
        )
        ContractServiceRates.objects.create(
            contract=self.contract,
            rates={"office_visit": "100.00", "lab_work":
"50.00"},
        )
```

```
    def _create_amendment(self, effective_date, updates):
        return AmendmentRate.objects.create(
            contract=self.contract,
            effective_date=effective_date,
            rates={st: rate for st, rate in updates},
        )
```

TESTS & SERVICES

TEST

```
    def
test_other_service_uses_base_rate_when_not_amended(self):
        self._create_amendment(date(2024, 6, 1),
[("office_visit", "120.00")])
```

```
        rate = get_service_rate_on_date("CTR-001", "lab_work",
date(2024, 7, 1))
        self.assertEqual(rate, Decimal("50.00"))
```

SERVICE

```
base_rates: ContractServiceRates = contract.base_rate_set
if normalized_service_type in base_rates.rates:
        return
Decimal(str(base_rates.rates[normalized_service_type]))
```

SERVICE

```
    try:
        base_rates: ContractServiceRates =
contract.base_rate_set
    except ContractServiceRates.DoesNotExist as exc:
        raise ValueError(f"No base rates found for contract
'{contract_id}'") from exc
```

NO TEST BUT SHOULD BE

```
    def test_missing_base_rates_raises_error(self):
        contract = Contract.objects.create(
```

```
            contract_id="CTR-NO-RATES",
effective_date=date(2024, 1, 1)
        )
        with self.assertRaisesRegex(ValueError, "No base rates
found for contract 'CTR-NO-RATES'"):
            get_service_rate_on_date("CTR-NO-RATES",
"office_visit", date(2024, 2, 1))
```

THEN AMENDMENTS

SERVICE

```
    amendment_rates = (
        AmendmentRate.objects.filter(contract=contract,
effective_date__lte=query_date)
        .order_by("-effective_date", "-id")
        .first()
    )

    if amendment_rates:
        if normalized_service_type in amendment_rates.rates:
            return
Decimal(str(amendment_rates.rates[normalized_service_type]))
```

TEST
AMENDMENT ONES

```
def test_service_type_matching_is_case_insensitive
```

Will catch you out so

```
amendment_rate_map = {k.strip().lower(): v for k, v in
amendment_rates.rates.items()}
        if normalized_service_type in amendment_rate_map:
            return
Decimal(str(amendment_rate_map[normalized_service_type]))
```

Should fix all