У даному документі приведено найскладніші приклади запитив, які використовуються у програмі

Відображення предметів

```
SELECT i."ID", u."Username" AS "Owner", i."Price", i."\Nametag\", it.\"Description\" AS
\"ItemDescription\",
w.\"Weapon\" AS \"WeaponDescription\",
s.\"Category\" AS \"SpecialDescription\",
\"OwnerID\", p.\"Preview\" AS  \"Preview\",
 \"Type\"  FROM \"Items\" i
    LEFT JOIN \"Users\" u ON i.\"OwnerID\" = u.\"ID\"
    LEFT JOIN \"ItemTypes\" it ON i.\"Type\" = it.\"TypeID\"
    LEFT JOIN \"WeaponIDToWeapon\" w  ON i.\"Weapon\" = w.\"WeaponID\"
    LEFT JOIN \"SpecialToCategory\" s ON i.\"Special\" = s.\"SpecialID\"
    LEFT JOIN \"ItemPreviews\" p ON i.\"ID\" = p.\"ItemID\"
```

Запит вибирає записи з таблиці **Items** — це головна таблиця, де зберігаються предмети маркетплейсу.

Після вибірки даних із таблиці **Items**, запит за допомогою серії LEFT JOIN під'єднує довідкову та додаткову інформацію про предмет із пов’язаних таблиць.
Кожен з цих JOIN-ів забезпечує перетворення технічних числових ідентифікаторів у зрозумілі людині текстові описи, а також додає візуальні дані.

Ось що роблять ці об’єднання:

- За допомогою JOIN "Users" до кожного предмета додається ім’я його власника (Username), яке відповідає значенню OwnerID у Items.
- JOIN "ItemTypes" повертає текстовий опис типу предмета (наприклад «Скін», «Ніж», «Сувенір»), підставляючи замість числового поля Type людськозрозуміле значення.
- JOIN "WeaponIDToWeapon" забезпечує отримання назви зброї, якщо предмет є зброєю; інакше повертається NULL.
- JOIN "SpecialToCategory" додає інформацію про категорію special-параметра предмета (наприклад StatTrak, Souvenir тощо), що теж зберігається окремо в lookup-таблиці.
- JOIN "ItemPreviews" додає прев’ю-файл (bytea) — зображення або іконку предмета, яке використовується для відображення в інтерфейсі.

Усі об’єднання виконані через **LEFT JOIN**, що дозволяє:

- повертати предмет навіть тоді, коли деякі пов’язані довідкові дані відсутні;
- уникати втрати рядків у тих випадках, коли певний довідник не містить відповідного запису (наприклад, немає прев’ю або Special = NULL).

Таким чином, група JOIN-ів формує **повноцінний, збагачений набір даних**, де числові ідентифікатори перетворюються на повні текстові описи, а предмет отримує усю інформацію, необхідну для коректного відображення у списку або картці предмета.
Запит повертає наступну інформацію про кожен предмет:

- **ID** — унікальний ідентифікатор предмета.
- **Owner** — ім'я власника предмета (Username).

- **Price** — поточна ціна предмета.
- **Nametag** — кастомний напис, встановлений користувачем.
- **ItemDescription** — текстовий опис типу предмета (з таблиці ItemTypes).
- **WeaponDescription** — назва зброї (якщо предметом є зброя).
- **SpecialDescription** — опис special-категорії предмета.
- **OwnerID** — числовий ID власника (використовується програмною логікою).
- **Preview** — зображення (bytea), яке використовується для показу предмета.

Type — числовий тип предмета, який може використовуватися для фільтрації або додаткової логіки.

Приклад виводу:

Owner	Name	Price	Type	Weapon	Special
User_6	survival_#12	131	Weapon	survival	Souvenir
User_13	sawedoff_#13	33	Weapon	sawedoff	Highlight
User_12	nomad_#14	109	Weapon	nomad	Highlight
User_2	r8_#15	103	Weapon	r8	Highlight
User_4	g3sg1_#17	144	Weapon	g3sg1	Stattrak
User_11	glock18_#18	45	Weapon	glock18	Normal
User_5	fiveseven_#19	99	Weapon	fiveseven	Souvenir
User_15	famas_#20	70	Weapon	famas	Highlight
User_6	Katto_2023_#101	24	Sticker		
User_13	device_sig_#102	31	Sticker		
User_10	s1mple_sig_#103	16	Sticker		
User_17	m0NESY_sig_#104	45	Sticker		
User_19	Flame_#105	47	Sticker		
User_17	FaZe_#106	31	Sticker		
User_7	G2_#107	42	Sticker		
User_18	DreamHack_#108	36	Sticker		

Відображення транзакцій

```
SELECT t.\"TransactionID\", t.\"TransactionID\", t.\"SellerID\",  t.\"BuyerID\", t.\"ItemID\",
        s.\"Username\" AS \"SellerName\",
        b.\"Username\" AS \"BuyerName\",
        i.\"Nametag\" AS \"ItemNametag\",
        it.\"Description\" AS \"ItemType\",
        t.\"Price\",
        t.\"Date\",
        t.\"Archive\",
        t.\"Canceled\"
    FROM \"Transactions\" t
    LEFT JOIN \"Users\" s ON t.\"SellerID\" = s.\"ID\"
    LEFT JOIN \"Users\" b ON t.\"BuyerID\"  = b.\"ID\"
    LEFT JOIN \"Items\" i ON t.\"ItemID\"   = i.\"ID\"
    LEFT JOIN \"ItemTypes\" it ON i.\"Type\" = it.\"TypeID\"
```

Запит формує повний набір даних про кожну транзакцію, беручи інформацію з таблиці **Transactions** та доповнюючи її пов'язаними полями з інших таблиць.

LEFT JOIN-и та їх роль

Усі об’єднання виконані через **LEFT JOIN**, щоб транзакції відображалися навіть у випадках, коли пов’язані дані були змінені, видалені або недоступні (наприклад, користувача деактивовано, предмет архівовано, тип змінено).

JOIN-и виконують таке:

- **LEFT JOIN Users s ON SellerID = ID**
  → додає ім’я продавця (SellerName).
  Якщо продавець був видалений, транзакція все одно повертається, але ім’я буде NULL.
- **LEFT JOIN Users b ON BuyerID = ID**
  → додає ім’я покупця (BuyerName).
  Аналогічно, покупець може бути відсутній у системі.
- **LEFT JOIN Items i ON ItemID = ID**
  → дозволяє отримати назву предмета (ItemNametag) і його тип.
  Якщо предмет переміщений або видалений, транзакція зберігається в історії.
- **LEFT JOIN ItemTypes it ON i.Type = TypeID**
  → додає текстовий опис типу предмета (ItemType).
  Цей опис використовується у UI для зрозумілого відображення виду предмета.

Призначення вибраних полів

Запит повертає наступну інформацію:

- **TransactionID** — унікальний ідентифікатор операції.
- **SellerID / BuyerID** — числові ID сторін транзакції (потрібно для логіки програми).
- **ItemID** — предмет, який був проданий.
- **SellerName / BuyerName** — текстові імена користувачів (для відображення).
- **ItemNametag** — кастомний напис на предметі, якщо був встановлений.
- **ItemType** — зрозумілий опис типу предмета (наприклад, “Knife”, “Sticker”, “Skin”).
- **Price** — вартість продажу.
- **Date** — коли операція була проведена.
- **Archive** — чи прихована транзакція з основного відображення.
- **Canceled** — чи була транзакція скасована.

Приклад виводу:

Seller	Buyer	Item	Price	Date	Type	Archived	Cancelled
User_16	User_19	MiniKnife_#437	324	11.01.2025 23:56:(	Keychain		False
User_19	User_10	MiniKnife_#437	405	09.12.2025 23:56:(	Keychain		False
User_10	User_20	MiniKnife_#437	198	10.08.2024 22:56:(	Keychain		False
User_14	User_6	CombatHook_#438	162	13.03.2025 23:56:(	Keychain		False
User_6	User_10	CombatHook_#438	219	07.12.2025 23:56:(	Keychain		False
User_10	User_6	CombatHook_#438	494	04.07.2025 22:56:(	Keychain		False
User_6	User_11	CombatHook_#438	89	15.12.2024 23:56:(	Keychain		False
Game Owner	User_13	HoloStar_#439	338	06.12.2025 23:56:(	Keychain		False
User_13	User_19	HoloStar_#439	165	09.05.2025 22:56:(	Keychain		False
User_7	User_6	LuckyCoin_#440	489	21.05.2024 22:56:(	Keychain		False
User_6	User_12	LuckyCoin_#440	202	21.02.2024 23:56:(	Keychain		False
User_12	User_6	LuckyCoin_#440	133	08.10.2025 22:56:(	Keychain		False
User_8	User_11	Liquid_#118	178	08.09.2025 22:56:(	Sticker		True
User_8	User_12	falchion_#16	81	03.05.2025 22:56:(	Weapon		True
User_2	User_12	Katto_2023_#125	319	27.09.2025 22:56:(	Sticker		True

Аналітичні запити

Статистика транзакцій користувача

```
SELECT
 COUNT(*) FILTER (WHERE \"Canceled\"=FALSE) AS total_deals,
      SUM(\"Price\") FILTER (WHERE \"Canceled\"=FALSE) AS total_turnover,
       AVG(\"Price\") FILTER (WHERE \"Canceled\"=FALSE) AS avg_price
       FROM \"Transactions\" t
        WHERE t.\"Canceled\"=FALSE
      AND t.\"Date\" BETWEEN :d1 AND :d2 ;
      AND (t.\"SellerID\"=(SELECT \"ID\" FROM \"Users\" WHERE \"Username\"=:uname)
         OR  t.\"BuyerID\"=(SELECT \"ID\" FROM \"Users\" WHERE \"Username\"=:uname ) )
```

Запит обчислює статистику діяльності певного користувача на маркетплейсі за вибраний період часу.
У вибірку потрапляють лише успішні (не скасовані) транзакції, де користувач виступає або продавцем, або покупцем.
Результатом є три агреговані показники — кількість угод, загальний оборот та середня ціна угоди.

Фільтрація транзакцій і їх роль

Запит накладає серію умов у WHERE, які формують точну підмножину даних:

- t."Canceled" = FALSE
  у статистику включаються лише завершені угоди.
  Скасовані транзакції не враховуються.
- t."Date" BETWEEN :d1 AND :d2
  у розрахунок потрапляють лише угоди, виконані у вибраному часовому діапазоні.
  Дати передаються через параметри.
- користувач є учасником угоди
  t."SellerID" = (SELECT "ID" FROM "Users" WHERE "Username" = :uname)
  OR
  t."BuyerID" = (SELECT "ID" FROM "Users" WHERE "Username" = :uname)

  транзакція враховується лише тоді, коли користувач був продавцем або покупцем.
  Це дає повне й коректне уявлення про його торгову активність.

Таким чином, застосовується триступенева фільтрація:
по статусу → по часу → по учаснику угоди.

Призначення повернених полів

Запит повертає три агреговані метрики, які описують торгову активність користувача:

- **total_deals**
- COUNT(*) FILTER (WHERE "Canceled" = FALSE)

  кількість успішних угод, у яких брав участь користувач.
  Дає уявлення про активність і загальний обсяг операцій.

- **total_turnover**
- SUM("Price") FILTER (WHERE "Canceled" = FALSE)

  → сумарний оборот у грошах.
  Показує реальний обсяг торгівлі користувача за період.

- **avg_price**
- AVG("Price") FILTER (WHERE "Canceled" = FALSE)

  → середня ціна однієї угоди.
  Характеризує рівень цін, за якими користувач зазвичай торгує.

Фільтри в агрегатах дублюють логіку WHERE, підсилюючи надійність та коректність обчислень.

Аагрегована статистика за инвентарем користувачів

```
SELECT u.\"Username\", COUNT(i.\"ID\") AS items_count, SUM(i.\"Price\") AS total_value
  FROM \"Users\" u
LEFT JOIN \"Items\" i ON i.\"OwnerID\" = u.\"ID\"
WHERE 1=1  AND u.\"Username\" = :uname
GROUP BY u.\"Username\"
ORDER BY total_value DESC NULLS LAST
```

Запит обчислює зведену статистику щодо предметів, які належать певному користувачу. До користувача підключаються його предмети, після чого виконується підрахунок кількості цих предметів та сумування їхньої загальної вартості.

Запит починається з таблиці **Users**, після чого через LEFT JOIN підключаються предмети з таблиці **Items**, власником яких є відповідний користувач.

Призначення LEFT JOIN:

- забезпечити повернення користувача навіть тоді, коли у нього **немає жодного предмета** (items_count = 0, total_value = NULL);
- уникнути втрати користувача з результатів у випадку, коли інвентар порожній або тимчасово недоступний;
- гарантувати коректність агрегатів COUNT та SUM для кожного користувача.

Запит виконується **для одного конкретного користувача**, ім’я якого передається параметром.
Це дозволяє отримати персональну статистику по його інвентарю.

Запит повертає три ключові показники:

- **Username**
  ім’я користувача, для якого рахується статистика.
- **items_count**
- COUNT(i."ID")

→ кількість предметів у власності цього користувача.
Якщо предметів немає — повертається 0.

- **total_value**
- SUM(i."Price")

→ сумарна вартість усіх його предметів.
Відображає потенційний «капітал» у предметах.
Якщо предметів немає — значення буде NULL (або 0 — залежить від логіки обробки у застосунку).

Групування та сортування

```
GROUP BY u."Username"
ORDER BY total_value DESC NULLS LAST
```

- **GROUP BY** агрегує предмети користувача в один рядок із підсумковими значеннями.
- **ORDER BY total_value DESC NULLS LAST**
  упорядковує результат так, що:
  - користувачі з більшою сумарною вартістю інвентарю стоять вище,
  - користувачі без предметів (total_value = NULL) переміщуються в кінець списку.

У даному випадку фільтрація за Username дає лише один рядок, але ORDER BY зберігається як універсальна частина запиту, яка буде грати роль, коли запит буде виконуватись для декількох

Вивод графіку активності ринку за транзакціями

Запит використовується для виводу графіка за допомогою компонента **TChart** і формує щоденну статистику активності маркетплейсу, підраховуючи кількість успішних транзакцій по днях.
Він використовується для побудови графіків активності ринку, heatmap-діаграм, аналітичних панелей та моніторингу змін активності з часом.

Фільтрація транзакцій і їх роль

У запиті використовується проста, але важлива умова:

```
WHERE "Canceled" = FALSE
```

Це гарантує, що в статистику потрапляють **лише реальні завершені угоди**, без скасованих транзакцій, які могли б викривити дані.

Таким чином, аналіз відображає **фактичну активність ринку**, а не всі спроби продажу.

🔻 Групування по днях

Виділення дати з timestamp

```
DATE("Date") AS day
```

Поле Date містить дату та час транзакції (timestamptz).
Функція DATE() відкидає час і залишає лише календарну дату.

Це дозволяє будувати статистику:

- за календарними днями,
- незалежно від конкретного часу проведення угод.

Агрегація по днях

GROUP BY day

У результаті кожен рядок означає **один день**, а всі транзакції за цей день підсумовуються.

Призначення вибраних полів

Запит повертає два основні поля:

Поле	Значення
day	Дата, у яку відбулися транзакції (без часу)
deals	Кількість успішних угод у цей день

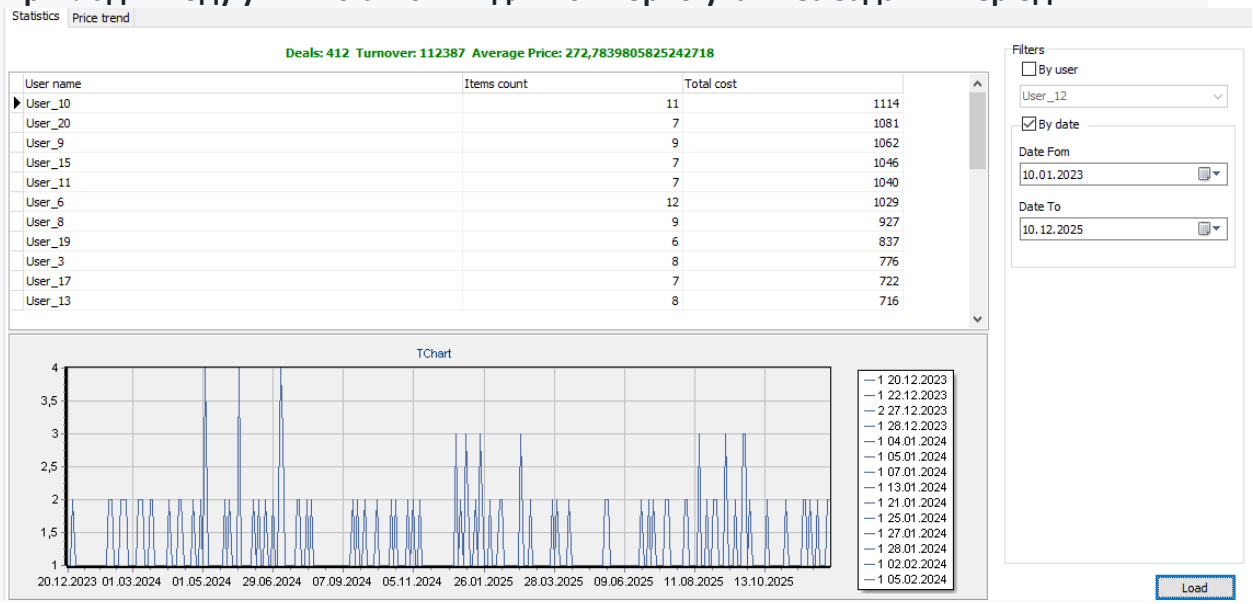
Ці дані дозволяють легко побудувати графік активності маркетплейсу.

Сортування результатів

RDER BY day

Результати впорядковані від найстаріших до найновіших дат.
Це критично для правильного відображення графіка подій у часовому порядку.

Приклад виводу у вікні статистики для всіх користувачів за заданий період:



Аналітика змін цін предметів за часом

```
SELECT * FROM (
   SELECT i.\"ID\" AS ItemID,
          i.\"Nametag\",
          FIRST_VALUE(t.\"Price\") OVER w AS first_price,
          LAST_VALUE(t.\"Price\") OVER w AS last_price,
          AVG(t.\"Price\") OVER w AS avg_price,
          (LAST_VALUE(t.\"Price\") OVER w - FIRST_VALUE(t.\"Price\") OVER w) AS price_change,
          ROUND( (LAST_VALUE(t.\"Price\") OVER w - FIRST_VALUE(t.\"Price\") OVER w)
               * 100.0 / NULLIF(FIRST_VALUE(t.\"Price\") OVER w, 0), 2) AS change_percent
        FROM \"Transactions\" t
        JOIN \"Items\" i ON i.\"ID\" = t.\"ItemID\"
        WHERE t.\"Canceled\" = FALSE
        WINDOW w AS (
          PARTITION BY t.\"ItemID\"
          ORDER BY t.\"Date\"
          ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
        )
    ) sub   ORDER BY price_change DESC
```

Запит аналізує, як змінювалися ціни конкретних предметів у ході їхніх продажів.
Він використовує віконні функції для визначення першої, останньої та середньої ціни товару, а також розраховує абсолютну та відносну зміну ціни.

Цей запит є частиною аналітичного модуля маркетплейсу й дозволяє оцінювати динаміку ринку.

JOIN і його роль

```
JOIN "Items" i ON i."ID" = t."ItemID"
```

Запит працює з таблицею **Transactions**, де зберігаються фактичні дані про продаж.
Об'єднання з таблицею **Items** додає:

- назву предмета (**Nametag**),
- зв’язок між записами про продаж та самим предметом.

Після об’єднання кожен запис транзакції містить інформацію про конкретний предмет.

Фільтрація транзакцій

```
WHERE t."Canceled" = FALSE
```

У розрахунок включаються лише реальні успішні транзакції.
Скасовані угоди виключено, щоб не спотворювати аналітичні показники.

Віконні функції та їх призначення

Віконна рамка OVER w має вигляд:

```
WINDOW w AS (PARTITION BY i."ID" ORDER BY t."Date"
        ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
```

Це дозволяє аналізувати послідовність цін **для кожного предмета окремо**.

Ось які значення обчислюються:

Початкова ціна предмета (first_price)

FIRST_VALUE(t."Price") OVER w

- це ціна на **першій за часом транзакції**;
- характеризує початкову вартість предмета на ринку.

Остаточна ціна (last_price)

LAST_VALUE(t."Price") OVER w

- ціна на **останній транзакції**, доступній у вибірці;
- показує актуальну або найближчу до актуальної вартість.

Використання віконної рамки гарантує, що остання ціна дійсно є найпізнішою за датою продажу.

Середня ціна за весь період (avg_price)

AVG(t."Price") OVER w

- усереднює всі продажні ціни предмета;
- дозволяє бачити загальні ринкові тенденції.

Абсолютна зміна ціни (price_change)

LAST_VALUE(...) - FIRST_VALUE(...)

Показує, наскільки зросла або впала ціна предмета за весь період його продажів.

Відносна зміна у відсотках (change_percent)

```
ROUND(
  (last - first) * 100.0 / NULLIF(first, 0),  2 )
```

- виражає зміну у відсотках;
- NULLIF(first, 0) запобігає діленню на нуль.

Цей показник дозволяє швидко побачити зростання або падіння цін у процентах.

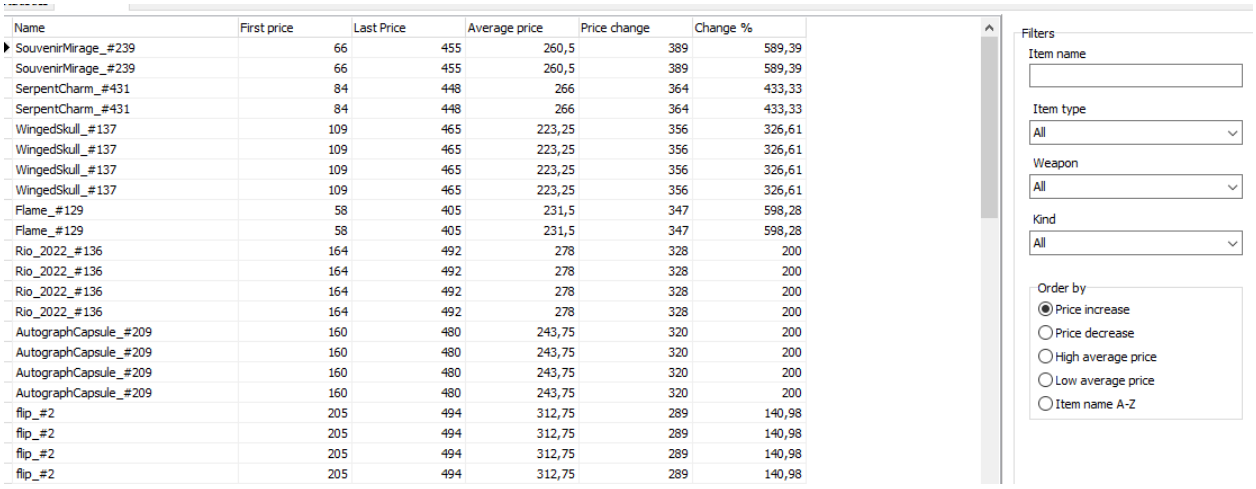
Призначення вибраних полів

Запит повертає:

Поле	Значення
ItemID	Унікальний ID предмета
Nametag	Назва предмета

Поле	Значення
first_price	Ціна під час першого продажу
last_price	Ціна під час останнього продажу
avg_price	Середня ціна за всі продажі
price_change	Абсолютна зміна ціни
change_percent	Відсоткова зміна ціни

Приклад виводу аналітики змін цін предметів за часом



Name	First price	Last Price	Average price	Price change	Change %
SouvenirMirage_#239	66	455	260,5	389	589,39
SouvenirMirage_#239	66	455	260,5	389	589,39
SerpentCharm_#431	84	448	266	364	433,33
SerpentCharm_#431	84	448	266	364	433,33
WingedSkull_#137	109	465	223,25	356	326,61
WingedSkull_#137	109	465	223,25	356	326,61
WingedSkull_#137	109	465	223,25	356	326,61
WingedSkull_#137	109	465	223,25	356	326,61
Flame_#129	58	405	231,5	347	598,28
Flame_#129	58	405	231,5	347	598,28
Rio_2022_#136	164	492	278	328	200
Rio_2022_#136	164	492	278	328	200
Rio_2022_#136	164	492	278	328	200
Rio_2022_#136	164	492	278	328	200
AutographCapsule_#209	160	480	243,75	320	200
AutographCapsule_#209	160	480	243,75	320	200
AutographCapsule_#209	160	480	243,75	320	200
AutographCapsule_#209	160	480	243,75	320	200
flip_#2	205	494	312,75	289	140,98
flip_#2	205	494	312,75	289	140,98
flip_#2	205	494	312,75	289	140,98
flip_#2	205	494	312,75	289	140,98

Оптимістичне та песимістичне блокування при оновлені бази

Реалізація оптимістичного блокування (Optimistic Locking)

Приклад: таблиця Users (PostgreSQL + FireDAC)

Що таке оптимістичне блокування

Оптимістичне блокування — це механізм захисту від конфліктів паралельного доступу, при якому:

- записи **не блокуються наперед**;
- кожен клієнт працює зі «своєю копією» даних;
- при спробі оновлення або видалення перевіряється, **чи не була змінена ця ж сама запис іншою транзакцією**.

Якщо запис було змінено — операція **UPDATE / DELETE не виконується**, і програма отримує повідомлення про конфлікт.

Структура таблиці Users

Вихідна таблиця:

```
CREATE TABLE IF NOT EXISTS "Users"
(
   "ID"              SERIAL PRIMARY KEY,
   "Username"        varchar(256) NOT NULL,
   "Balance"         integer NOT NULL DEFAULT 0,
   "DateJoined"      timestamptz NOT NULL DEFAULT now(),
   "RestrictionType"   integer REFERENCES "RestrictionType"("ID") DEFAULT 0,
   "RestrictedUntil"   timestamptz,
   "RestrictionReason" varchar(256),
   "Archive"         boolean DEFAULT FALSE
);
```

Додавання поля версії (обов’язковий крок)

Для реалізації оптимістичного блокування додається **поле версії**:

```
ALTER TABLE "Users"
ADD COLUMN "Version" integer NOT NULL DEFAULT 0;
```

Призначення поля Version

- зберігає номер версії запису;
- автоматично збільшується при кожному успішному UPDATE;
- використовується для перевірки конфлікту змін.

Основний SELECT-запит (TFDQuery)

Запит **обов'язково повинен містити поле Version**:

```
SELECT
  "ID",
  "Username",
  "Balance",
  "DateJoined",
  "RestrictionType",
  "RestrictedUntil",
  "RestrictionReason",
  "Archive",
  "Version"
FROM "Users"
ORDER BY "ID";
```

Після цього в **Fields Editor** для TFDQueryUsers потрібно створити persistent fields (Fields → Add All Fields).

Використання TFDUpdateSQL

Для керування UPDATE / DELETE використовується компонент **TFDUpdateSQL**, наприклад FDUpdateSQLUsers.

У TFDQueryUsers:
UpdateObject = FDUpdateSQLUsers

UPDATE з оптимістичним блокуванням

SQL для оновлення (ModifySQL)

```
UPDATE "Users"
SET
  "Username"       = :NEW_Username,
  "Balance"        = :NEW_Balance,
  "RestrictionType"   = :NEW_RestrictionType,
  "RestrictedUntil"   = :NEW_RestrictedUntil,
  "RestrictionReason" = :NEW_RestrictionReason,
  "Archive"        = :NEW_Archive,
  "Version"        = "Version" + 1
WHERE
  "ID" = :OLD_ID
  AND "Version" = :OLD_Version;
```

Пояснення

- :OLD_ID — ID запису **в момент завантаження**;
- :OLD_Version — версія запису **в момент завантаження**;
- якщо інший користувач змінив запис — Version вже інший;
- UPDATE не змінює жодного рядка → виникає конфлікт.

DELETE з оптимістичним блокуванням

SQL для видалення (DeleteSQL)

```
DELETE FROM "Users"
WHERE
  "ID" = :OLD_ID
  AND "Version" = :OLD_Version;
```

Пояснення

- запис буде видалений **тільки якщо його версія не змінилась**;
- якщо запис вже був змінений — DELETE не спрацює;
- FireDAC повідомить про конфлікт.

Налаштування UpdateOptions у TFDQueryUsers

Рекомендовані параметри:

```
UpdateOptions.KeyFields = 'ID'
UpdateOptions.UpdateMode = upWhereAll
UpdateOptions.CountUpdatedRecords = True
```

Це дозволяє FireDAC правильно працювати з OLD_* параметрами.

9. Обробка конфлікту (RAD Studio 10.3)

У RAD Studio 10.3 немає готових кодів er_FD_UpdRowNotFound, тому конфлікт визначається:

- за текстом повідомлення;

Обробник OnUpdateError

```
void __fastcall TMainForm::fdqryUsersUpdateError(
  TDataSet *ASender,
  EFDException *AException,
  TFDDatSRow *ARow,
  TFDUpdateRequest ARequest,
  TFDErrorAction &AAction)
{
  UnicodeString msg = AException->Message;

  // Оптимістичний конфлікт
  if (msg.Pos(L"Row not found") > 0 ||
    msg.Pos(L"Record changed") > 0)
  {
    ShowMessage(
      L"Запис був змінений іншим користувачем.\n"
      L"Оновіть дані та спробуйте ще раз."
    );
    AAction = eaSkip;   // або eaFail
    return;
  }
```

```
  // Інші помилки
  ShowMessage(L"Помилка оновлення: " + msg);
  AAction = eaFail;
}
```

Поведінка системи при конкурентному доступі

1. Користувач A завантажує запис (Version = 3);
2. Користувач B змінює той самий запис (Version → 4);
3. Користувач A намагається зберегти зміни;
4. UPDATE / DELETE не змінює жодного рядка;
5. FireDAC викликає OnUpdateError;
6. Користувач отримує повідомлення про конфлікт

Реалізація песимістичного блокування (Pessimistic Locking)

Приклад: таблиця Transactions (PostgreSQL + FireDAC)

Призначення песимістичного блокування

Песимістичне блокування застосовується у випадках, коли:

- паралельна зміна одного запису **неприпустима**;
- операція має фінансові або бізнес-критичні наслідки;
- необхідно гарантувати, що **лише один користувач** у конкретний момент часу може змінювати або видаляти запис.

Для таблиці **Transactions** це особливо важливо, оскільки вона пов’язана з:

- ціною угоди;
- продавцем і покупцем;
- фінансовими операціями;
- архівацією та історією транзакцій.

Структура таблиці Transactions

```
CREATE TABLE IF NOT EXISTS "Transactions"
(
  "TransactionID"  SERIAL PRIMARY KEY,
  "SellerID"      integer REFERENCES "Users"("ID"),
  "BuyerID"       integer REFERENCES "Users"("ID"),
  "ItemID"        integer REFERENCES "Items"("ID"),
  "Price"         integer NOT NULL,
  "Date"          timestamptz NOT NULL,
  "Archive"       boolean DEFAULT FALSE
);
```

Принцип песимістичного блокування в PostgreSQL

*У PostgreSQL песимістичне блокування **окремого рядка** реалізується через:*

```
SELECT ... FOR UPDATE;
```

Для клієнтських (GUI) застосунків **рекомендовано** використовувати:

SELECT ... FOR UPDATE NOWAIT;

Поведінка FOR UPDATE NOWAIT

- якщо рядок **не заблокований** — він блокується поточною транзакцією;
- якщо рядок **вже заблокований** іншою транзакцією — PostgreSQL **одразу повертає помилку**, без очікування;
- програма може повідомити користувача, що запис тимчасово недоступний.

Архітектура FireDAC-компонентів

Для коректної реалізації песимістичного блокування необхідно **чітко розділити ролі компонентів**:

Компонент	Призначення
fdqryTransactions	SELECT для відображення списку транзакцій (грид)
fdqryUpdate	UPDATE / DELETE транзакцій
fdqryLock	Песимістичне блокування (SELECT … FOR UPDATE)
fdtrans1	Явна транзакція

fdqryLock обов'язково працює в тій самій транзакції, що і UPDATE / DELETE.

Налаштування fdqryLock

Щоб PostgreSQL приймав FOR UPDATE, необхідно вимкнути використання серверних курсорів FireDAC.

Обов'язкові властивості:

```
Transaction = fdtrans1
CursorKind = ckForwardOnly
FetchOptions.Unidirectional = True
FetchOptions.Mode = fmAll
ResourceOptions.CmdExecMode = amBlocking
```

Ці налаштування запобігають створенню
DECLARE CURSOR WITH HOLD, який **несумісний з FOR UPDATE у PostgreSQL**.

SQL-запит для блокування рядка Transactions

```
SELECT 1
FROM "Transactions"
WHERE "TransactionID" = :ID
FOR UPDATE NOWAIT;
```

Особливості:

- запит **не читає дані**;

- використовується виключно для блокування;
- блокування діє **до COMMIT або ROLLBACK**.

Загальна схема роботи

```
StartTransaction
  ↓
SELECT ... FOR UPDATE NOWAIT   ← блокування рядка
  ↓
UPDATE / DELETE Transactions
  ↓
Commit / Rollback          ← зняття блокування
```

Приклад оновлення транзакції з песимістичним блокуванням

```
void TMainForm::ArchiveTransaction(int transactionID, bool archive)
{
  try
  {
    // 1. Початок транзакції
    fdtrans1->StartTransaction();

    // 2. Блокування рядка
    fdqryLock->Close();
    fdqryLock->SQL->Text =
      "SELECT 1 FROM \"Transactions\" "
      "WHERE \"TransactionID\" = :ID "
      "FOR UPDATE NOWAIT";
    fdqryLock->ParamByName("ID")->AsInteger = transactionID;

    // Використовуємо ExecSQL(), а не Open()
    fdqryLock->ExecSQL();

    // 3. Оновлення поля Archive
    fdqryUpdate->Close();
    fdqryUpdate->SQL->Text =
      "UPDATE \"Transactions\" "
      "SET \"Archive\" = :A "
      "WHERE \"TransactionID\" = :ID";

    fdqryUpdate->ParamByName("A")->AsBoolean = archive;
    fdqryUpdate->ParamByName("ID")->AsInteger = transactionID;

    fdqryUpdate->ExecSQL();

    // 4. Фіксація змін
    fdtrans1->Commit();
  }
  catch (Exception &e)
  {
    // 5. Відкат транзакції та зняття блокування
    if (fdtrans1->Active)
      fdtrans1->Rollback();

    ShowMessage(
      L"Неможливо змінити транзакцію.\nЗапис наразі використовується іншим користувачем.\n\n" +
      e.Message
    );
```

```
    }

    // 6. Оновлення списку
    fdqryTransactions->Close();
    fdqryTransactions->Open();
}
```