КСиТ

5. Язык манипулирования данными ч.2

Базы данных

Хисамутдинов М.А.
кафедра №12 НИЯУ МИФИ
2026

План занятия

1. Контекст и схема «Остатки»
2. Обычный SELECT и WHERE
3. JOIN, подзапросы, EXISTS
4. DISTINCT, UNION, EXCEPT
5. COALESCE и работа с NULL
6. Агрегатные функции и HAVING

Контекст данных

Учебная схема «Остатки»:

- склады, товары, текущие остатки, перемещения
- есть NULL и нулевые остатки

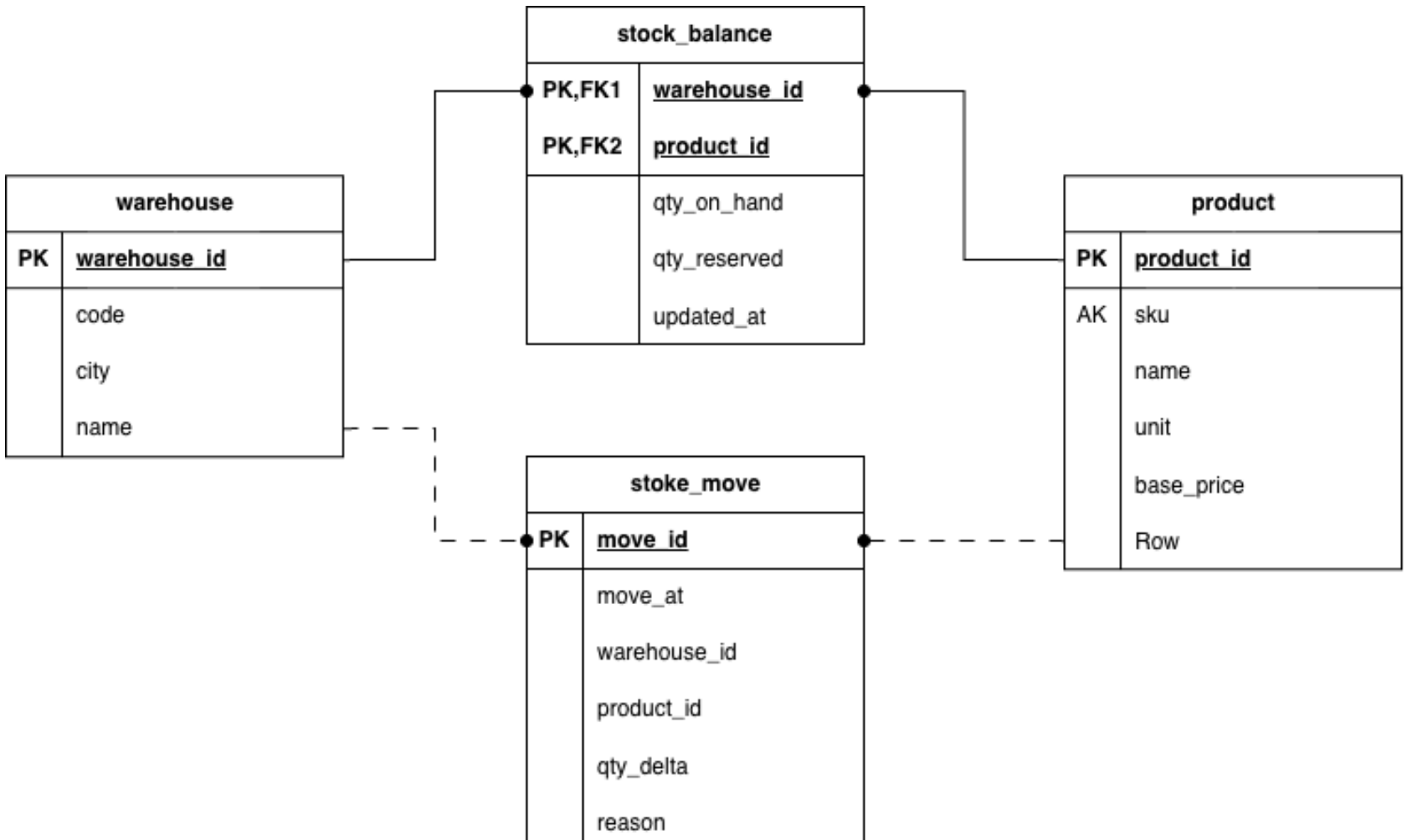
Схема «Остатки»

- `warehouse(warehouse_id, code, city, name)`
- `product(product_id, sku, name, category, base_price, discontinued)`
- `stock_balance(warehouse_id, product_id, qty_on_hand, qty_reserved, updated_at)`
- `stock_move(move_id, moved_at, warehouse_id, product_id, qty_delta, reason)`

Логика предметной области:

- `stock_balance` содержит «срез» остатков по складам
- `stock_move` хранит приход/расход
- `qty_on_hand` может быть NULL (неизвестно)
- `qty_reserved` может быть NULL (неизвестно)

Схема «Остатки»



Пример данных (warehouse)

warehouse_id	code	city	name
1	WH-A	Алматы	Склад Алматы (центральный)
2	WH-N	Астана	Склад Астана
3	WH-K	Караганда	Склад Караганда

Пример данных (product)

product_id	sku	name	category	base_price	discontinued
1	SKU-001	Гайка M8	Крепёж	5.00	TRUE
2	SKU-002	Болт M8x30	Крепёж	7.50	FALSE
3	SKU-010	Кабель USB-C 1м	Кабели	900.00	FALSE
4	SKU-011	Кабель USB-C 2м	Кабели	1200.00	FALSE
5	SKU-020	Маркер перманент.	Канц	250.00	FALSE

Пример данных (stock_balance)

warehouse_id	product_id	qty_on_hand	qty_reserved
1	1	120	10
1	2	80	0
1	3	NULL	NULL
1	4	25	5
1	5	0	0

Пример данных (stock_move)

move_id	moved_at	warehouse_id	product_id	qty_delta	reason
M1	2026-01-10 10:00:00	1	1	+50	RECEIPT
M2	2026-01-10 12:00:00	1	1	-20	SHIPMENT
M3	2026-01-11 09:30:00	2	2	+60	RECEIPT
M4	2026-01-13 14:00:00	1	4	-3	ADJUST

Доступный остаток

Чтобы принять заказ, нужно посчитать «доступный остаток»:

```
qty_available = qty_on_hand - qty_reserved
```

Если `qty_on_hand` или `qty_reserved` = NULL, результат неизвестен.

Команда SELECT

```
SELECT [DISTINCT] список_вывода
FROM источники
[INNER | LEFT | RIGHT | FULL] JOIN таблица_соединения
  ON условие_соединения
WHERE условие_отбора_строк
GROUP BY список_для_группирования
HAVING условие_отбора_групп
ORDER BY список_для_упорядочивания
LIMIT количество_строк OFFSET смещение;
```

- Каждому столбцу можно выдать новое имя (alias).
- Для каждой используемой таблицы можно указать короткий alias.
- Описываем, что хотим получить
- WHERE фильтрует строки до группировки
- HAVING фильтрует группы после GROUP BY

Логический порядок выполнения

1. FROM / JOIN
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY
7. LIMIT / OFFSET

ORDER BY и LIMIT

```sql
SELECT p.sku, p.name, p.base_price
FROM product p
WHERE p.category = 'Кабели'
ORDER BY p.base_price DESC, p.sku
FETCH FIRST 2 ROWS ONLY;
```

sku	name	base_price
SKU-011	Кабель USB-C 2м	1200.00
SKU-010	Кабель USB-C 1м	900.00

Сначала сортируем, затем ограничиваем число строк.

ORDER BY: NULLS FIRST/LAST

```sql
SELECT p.sku, sb.qty_on_hand
FROM product p
LEFT JOIN stock_balance sb
  ON sb.product_id = p.product_id
 AND sb.warehouse_id = 1
ORDER BY sb.qty_on_hand NULLS LAST;
```

`NULLS LAST` переносит строки с NULL в конец сортировки.

sku	qty_on_hand
SKU-020	0
SKU-011	25
SKU-002	80
SKU-001	120
SKU-010	NULL

LIMIT / OFFSET

```sql
SELECT sku, name
FROM product
ORDER BY sku
LIMIT 3 OFFSET 2;
```

LIMIT ограничивает число строк, OFFSET пропускает первые строки.

sku	name
SKU-010	Кабель USB-C 1м
SKU-011	Кабель USB-C 2м
SKU-020	Маркер перманент.

Псевдонимы и выражения

- алиасы (псевдонимы) упрощают чтение и запись
- выводим только нужные колонки

```
SELECT p.sku AS code,
       p.name AS title,
       p.base_price * 1.12 AS price_with_tax
FROM product p
WHERE p.base_price >= 100;
```

code	title	price_with_tax
SKU-010	Кабель USB-C 1м	1008.00
SKU-011	Кабель USB-C 2м	1344.00
SKU-020	Маркер перманент.	280.00

Тип запроса: Обычный JOIN

```sql
SELECT p.sku, p.name, sb.qty_on_hand
FROM stock_balance sb
JOIN product p ON p.product_id = sb.product_id
JOIN warehouse w ON w.warehouse_id = sb.warehouse_id
WHERE w.code = 'WH-A';
```

sku	name	qty_on_hand
SKU-001	Гайка M8	120
SKU-002	Болт M8x30	80
SKU-010	Кабель USB-C 1м	25

Пример обычного JOIN с фильтром по складу.

Тип запроса: WHERE

Вход:

```
SELECT p.sku, p.name,
       (sb.qty_on_hand - sb.qty_reserved) AS qty_available
FROM stock_balance sb
JOIN product p ON p.product_id = sb.product_id
JOIN warehouse w ON w.warehouse_id = sb.warehouse_id
WHERE w.code = 'WH-A'
  AND sb.qty_on_hand IS NOT NULL
  AND sb.qty_reserved IS NOT NULL
  AND (sb.qty_on_hand - sb.qty_reserved) < 20;
```

sku	name	qty_available
SKU-011	Кабель USB-C 2м	0

WHERE фильтрует строки по условиям и может использовать вычисления.

WHERE: операторы

- `IN`, `BETWEEN`, `LIKE`
- `IS NULL` вместо `= NULL`
- правильные скобки для `AND`/`OR`

```sql
SELECT sku, name
FROM product
WHERE category IN ('Кабели', 'Крепёж')
  AND base_price BETWEEN 5 AND 1000
  AND discontinued = FALSE;
```

sku	name
SKU-002	Болт M8x30
SKU-010	Кабель USB-C 1м

WHERE: LIKE

```
SELECT sku, name
FROM product
WHERE name LIKE 'Кабель%';
```

sku	name
SKU-010	Кабель USB-C 1м
SKU-011	Кабель USB-C 2м

Шаблоны:

- `%` — любая строка,
- `_` — один символ.

Примеры:

- **`Кабель%`** — начинается с «Кабель»,
- **`%USB%`** — содержит «USB»,
- **`Кабель _м`** — один символ перед «м».

WHERE: BETWEEN

```sql
SELECT sku, name, base_price
FROM product
WHERE base_price BETWEEN 5 AND 10
ORDER BY base_price;
```

sku	name	base_price
SKU-001	Гайка M8	5.00
SKU-002	Болт M8x30	7.50

BETWEEN включает границы. Эквивалентно `base_price >= 5 AND base_price <= 10`.

WHERE: IN

```sql
SELECT sku, name
FROM product
WHERE sku IN ('SKU-001', 'SKU-010', 'SKU-020')
ORDER BY sku;
```

sku	name
SKU-001	Гайка M8
SKU-010	Кабель USB-C 1м
SKU-020	Маркер перманент.

`IN` удобно, когда список значений известен заранее и небольшой.

WHERE: IN vs EXISTS

- `IN` удобен для явных списков
- `EXISTS` удобен для коррелированных условий

Коррелированный подзапрос использует значения из внешней строки. То есть для каждой строки «снаружи» подзапрос проверяется отдельно.

```
SELECT p.sku
FROM product p
WHERE p.product_id IN (
  SELECT sb.product_id
  FROM stock_balance sb
  WHERE sb.qty_on_hand > 0
);
```

sku
SKU-001
SKU-002
SKU-011

WHERE: работа со временем

Функция	Описание
`EXTRACT(YEAR FROM d)`	Год из даты
`EXTRACT(MONTH FROM d)`	Месяц (1–12)
`EXTRACT(DAY FROM d)`	День месяца
`EXTRACT(HOUR FROM d)`	Час (0–23)
`DATE_TRUNC('month', d)`	Усечь до начала месяца
`DATE(d)`	Извлечь только дату без времени
`NOW() / CURRENT_DATE`	Текущая дата и время

WHERE: работа со временем

Найти все движения товаров за конкретный день:

```sql
SELECT move_id, product_id, qty_delta, move_at
FROM stoke_move
WHERE move_at = '2024-03-15';

-- Движения с 1 января 2024 года:
WHERE move_at >= '2024-01-01'

-- Движения до конца 2023 года:
WHERE move_at < '2024-01-01'
```

Найти все движения за первый квартал 2024 года:

```sql
SELECT move_id, warehouse_id, product_id, qty_delta, move_at
FROM stoke_move
WHERE move_at BETWEEN '2024-01-01' AND '2024-03-31 23:59:59';
```

Ловушка BETWEEN с датами

WHERE: работа со временем

```
BETWEEN '2024-01-01' AND '2024-03-31'
```

не захватит записи 31 марта, у которых время HH:MM:SS > 00:00:00

Безопаснее писать:

```
WHERE move_at >= '2024-01-01' AND move_at < '2024-04-01'
```

WHERE: работа со временем

Все движения в марте 2024 года:

```
SELECT move_id, product_id, qty_delta, move_at
FROM stoke_move
WHERE EXTRACT(YEAR  FROM move_at) = 2024
  AND EXTRACT(MONTH FROM move_at) = 3;
```

Только понедельники (DOW = 1 в PostgreSQL):

```
SELECT move_id, qty_delta, move_at
FROM stoke_move
WHERE EXTRACT(DOW FROM move_at) = 1;  -- 0=воскресенье, 1=пн, ... 6=сб
```

WHERE: работа со временем

Очень часто нужно фильтровать по «последним N дням» относительно текущего момента:

```sql
SELECT move_id, product_id, qty_delta
FROM stoke_move
WHERE move_at >= NOW() - INTERVAL '7 days';

-- Остатки, обновлённые сегодня:
SELECT warehouse_id, product_id, qty_on_hand
FROM stock_balance
WHERE DATE(updated_at) = CURRENT_DATE;

-- Движения за текущий месяц:
WHERE DATE_TRUNC('month', move_at) = DATE_TRUNC('month', CURRENT_DATE)
```

WHERE: NOT IN и NULL

Предположим есть такие данные:

warehouse_id	product_id	qty_on_hand	qty_reserved
1	1	120	10
1	2	80	0
1	NULL	50	0
1	4	25	5

Если в подзапросе есть `NULL`, `NOT IN` может вернуть пустой результат.

Безопаснее: `NOT EXISTS`.

WHERE: NOT IN и NULL

```sql
SELECT p.sku
FROM product p
WHERE p.product_id NOT IN (
  SELECT sb.product_id
  FROM stock_balance sb
);
```

sku

```sql
SELECT p.sku
FROM product p
WHERE NOT EXISTS (
  SELECT 1
  FROM stock_balance sb
  WHERE sb.product_id = p.product_id
);
```

sku
SKU-010
SKU-020

WHERE: логика NULL

- `= NULL` всегда UNKNOWN
- используем `IS NULL` / `IS NOT NULL`

```sql
SELECT sku, name
FROM product
WHERE base_price IS NULL;
```

NULL: сравнения

`NULL` не равно ничему, даже самому себе. Для безопасного сравнения есть `IS DISTINCT FROM`.

```
SELECT 1 WHERE NULL IS DISTINCT FROM 0;
```

Вывод: одна строка со значением `1`, потому что `NULL` считается отличающимся от `0`.

NULL: IS DISTINCT FROM

```sql
SELECT p.sku, sb.qty_on_hand
FROM product p
LEFT JOIN stock_balance sb ON sb.product_id = p.product_id
WHERE sb.qty_on_hand IS DISTINCT FROM 0;
```

sku	qty_on_hand
SKU-001	120
SKU-002	80
SKU-010	NULL
SKU-011	25

Вывод: все строки, где `qty_on_hand` не равно 0, плюс строки с `qty_on_hand = NULL`.

Если бы использовали `WHERE qty_on_hand != 0`, то отфильтровали бы строку с NULL.

Типовая ошибка: забыли скобки

```sql
SELECT sku, name
FROM product
WHERE category = 'Кабели'
  OR category = 'Крепёж'
  AND base_price > 100;
```

Правильно:

```sql
SELECT sku, name
FROM product
WHERE (category = 'Кабели' OR category = 'Крепёж')
  AND base_price > 100;
```

Типовая ошибка: неявные джоины

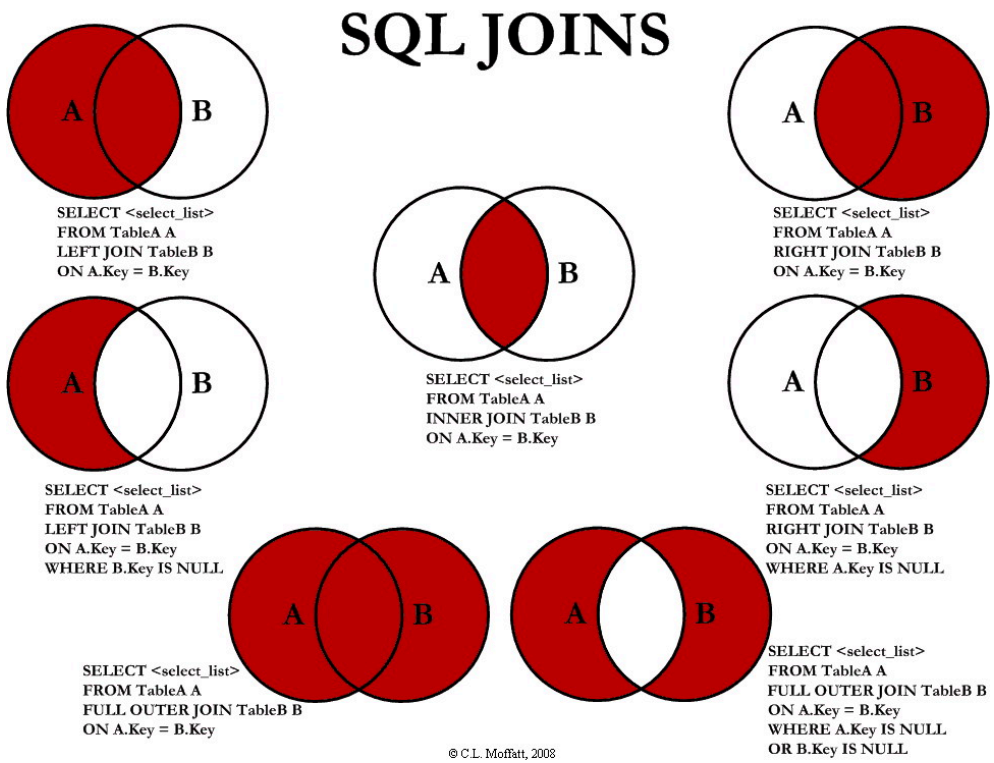
Плохо:

```
SELECT *
FROM product p, stock_balance sb
WHERE p.product_id = sb.product_id;
```

Хорошо:

```
SELECT *
FROM product p
JOIN stock_balance sb ON sb.product_id = p.product_id;
```

JOIN: диаграммы множеств



JOIN: типы соединений

- INNER JOIN: только совпавшие
- LEFT JOIN: все слева
- RIGHT JOIN: все справа
- FULL JOIN: все слева и справа
- CROSS JOIN: декартово произведение

Тип запроса: JOIN (INNER)

```
SELECT w.code, p.sku, sb.qty_on_hand
FROM warehouse w
JOIN stock_balance sb ON sb.warehouse_id = w.warehouse_id
JOIN product p ON p.product_id = sb.product_id
ORDER BY w.code, p.sku;
```

code	sku	qty_on_hand
WH-A	SKU-001	120
WH-A	SKU-002	80
WH-A	SKU-003	NULL

Тип запроса: JOIN (LEFT)

```
SELECT p.sku, p.name, sm.qty_delta
FROM product p
LEFT JOIN stock_move sm ON sm.product_id = p.product_id
ORDER BY p.sku, sm.moved_at;
```

sku	name	qty_delta
SKU-001	Гайка M8	+50
SKU-001	Гайка M8	-20
SKU-002	Болт M8x30	+60
SKU-011	Кабель USB-C 1м	NULL
SKU-010	Кабель USB-C 2м	-3
SKU-011	Маркер перманент.	NULL

LEFT JOIN сохраняет все строки слева, даже если справа нет совпадений.

JOIN: где писать условия

- условия соединения в `ON`
- фильтрацию оставляем в `WHERE`

```
SELECT p.sku, p.name, sm.qty_delta
FROM product p
LEFT JOIN stock_move sm ON sm.product_id = p.product_id
WHERE sm.warehouse_id = 1
```

sku	name	qty_delta
SKU-001	Гайка M8	+50
SKU-001	Гайка M8	-20
SKU-010	Кабель USB-C 2м	-3

JOIN: типичная ошибка

`WHERE sb.warehouse_id = 1` превращает LEFT JOIN в INNER JOIN.

Правильно:

```
SELECT p.sku, p.name, sm.qty_delta
FROM product p
LEFT JOIN stock_move sm
  ON sm.product_id = p.product_id AND sm.warehouse_id = 1;
```

sku	name	qty_delta
SKU-001	Гайка M8	+50
SKU-001	Гайка M8	-20
SKU-002	Болт M8x30	NULL
SKU-011	Кабель USB-C 1м	NULL
SKU-010	Кабель USB-C 2м	-3
SKU-011	Маркер перманент.	NULL

JOIN: фильтрация справа

Плохо (теряем товары без остатков):

```sql
SELECT p.sku, sb.qty_on_hand
FROM product p
LEFT JOIN stock_balance sb ON sb.product_id = p.product_id
WHERE sb.qty_on_hand > 0;
```

Хорошо:

```sql
SELECT p.sku, sb.qty_on_hand
FROM product p
LEFT JOIN stock_balance sb
  ON sb.product_id = p.product_id
 AND sb.qty_on_hand > 0;
```


JOIN: CROSS JOIN

```sql
SELECT w.code, p.sku
FROM warehouse w
CROSS JOIN product p;
```

code	sku
WH-A	SKU-001
WH-A	SKU-002
WH-A	SKU-010
WH-A	SKU-011
WH-A	SKU-020
WH-N	SKU-001
...	...

Число строк = (число складов) x (число товаров).

JOIN: SELF JOIN

Пример: товары одной категории сравнить между собой.

```
SELECT p1.sku AS sku1, p2.sku AS sku2, p1.category
FROM product p1
JOIN product p2
  ON p1.category = p2.category
 AND p1.sku < p2.sku;
```

sku1	sku2	category
SKU-001	SKU-002	Крепёж
SKU-010	SKU-011	Кабели

SELF JOIN соединяет таблицу саму с собой, когда нужно сравнить строки внутри одной сущности.

Тип запроса: Подзапрос (IN)

```sql
SELECT p.sku, p.name
FROM product p
WHERE p.product_id IN (
  SELECT sb.product_id
  FROM stock_balance sb
  WHERE sb.qty_on_hand > 0
);
```

sku	name
SKU-001	Гайка M8
SKU-002	Болт M8x30
SKU-010	Кабель USB-C 1м

IN оставляет строки, чьи значения есть в подзапросе.

Тип запроса: Подзапрос (скалярный)

Вход:

```
SELECT p.sku, p.name,
       (sb.qty_on_hand - sb.qty_reserved) AS qty_available
FROM stock_balance sb
JOIN product p ON p.product_id = sb.product_id
JOIN warehouse w ON w.warehouse_id = sb.warehouse_id
WHERE w.code = 'WH-A'
  AND sb.qty_on_hand IS NOT NULL
  AND sb.qty_reserved IS NOT NULL
  AND (sb.qty_on_hand - sb.qty_reserved) =
      (SELECT MAX(sb2.qty_on_hand - sb2.qty_reserved)
       FROM stock_balance sb2
       WHERE sb2.warehouse_id = sb.warehouse_id
         AND sb2.qty_on_hand IS NOT NULL
         AND sb2.qty_reserved IS NOT NULL);
```

Тип запроса: Подзапрос (скалярный)

Вывод:

sku	name	qty_available
SKU-001	Гайка M8	110

Скалярный подзапрос возвращает одно значение, которое используется в сравнении.

EXISTS vs JOIN

EXISTS проверяет наличие строк, JOIN строит результат. Часто EXISTS читается проще, когда нужен ответ «да/нет».

```
SELECT p.sku
FROM product p
WHERE EXISTS (
  SELECT 1
  FROM stock_balance sb
  WHERE sb.product_id = p.product_id
    AND COALESCE(sb.qty_on_hand, 0) > 0
);
```


Дубликаты после JOIN и DISTINCT

DISTINCT убирает повторяющиеся строки результата.

До:

```
SELECT p.category
FROM product p
JOIN stock_balance sb ON sb.product_id = p.product_id;
```

category
Крепёж
Крепёж
Крепёж
Кабели
Кабели

Дубликаты после JOIN и DISTINCT

После:

```
SELECT DISTINCT p.category
FROM product p
JOIN stock_balance sb ON sb.product_id = p.product_id;
```

category
Крепёж
Кабели

DISTINCT vs GROUP BY

DISTINCT удаляет повторения целых строк. GROUP BY агрегирует данные и позволяет считать суммы/средние.

```
SELECT p.category
FROM product p
GROUP BY p.category;
```

category
Крепёж
Кабели

Тип запроса: COALESCE

```
SELECT p.sku,
       COALESCE(sb.qty_on_hand, 0) AS qty_on_hand_0
FROM product p
LEFT JOIN stock_balance sb
  ON sb.product_id = p.product_id
 AND sb.warehouse_id = 1
ORDER BY p.sku;
```

sku	qty_on_hand_0
SKU-001	120
SKU-002	80
SKU-010	0
SKU-011	25
SKU-020	0

COALESCE подставляет первое не-NULL значение.

COALESCE: зачем нужно

- NULL означает «неизвестно»
- для отчетов часто нужно считать NULL как 0

COALESCE и CASE

```
SELECT p.sku,
       CASE
         WHEN sb.qty_on_hand IS NULL THEN 0
         ELSE sb.qty_on_hand
       END AS qty_on_hand_0
FROM product p
LEFT JOIN stock_balance sb
  ON sb.product_id = p.product_id
 AND sb.warehouse_id = 1;
```

CASE позволяет задать правило замены NULL на нужное значение.

NULLIF

```
SELECT p.sku,
       NULLIF(sb.qty_on_hand, 0) AS qty_on_hand_nonzero
FROM product p
LEFT JOIN stock_balance sb
  ON sb.product_id = p.product_id
 AND sb.warehouse_id = 1
ORDER BY p.sku;
```

NULLIF возвращает NULL, если значения равны. Здесь 0 превращается в NULL.

CAST и преобразования

```
SELECT sku,
       CAST(base_price AS DECIMAL(12,2)) AS price_dec
FROM product
WHERE base_price IS NOT NULL;
```

ИЛИ

```
SELECT sku,
       base_price::DECIMAL(12,2) AS price_dec
FROM product
WHERE base_price IS NOT NULL;
```

CAST приводит тип к нужному формату, чтобы избежать неявных преобразований.

CASE для категорий

```
SELECT sku, name,
       CASE
         WHEN base_price < 10 THEN 'cheap'
         WHEN base_price < 100 THEN 'mid'
         ELSE 'expensive'
       END AS price_bucket
FROM product
ORDER BY sku;
```

CASE удобно использовать для разметки строк по правилам.

Агрегатные функции

Основные: `SUM(col)`, `MAX(col)`, `MIN(col)`, `AVG(col)`, `COUNT(col)`, `COUNT(*)`.

Статистические: `STDDEV(col)`, `VARIANCE(col)`

Тип запроса: SUM

```
SELECT w.code,
       SUM(sb.qty_on_hand) AS total_on_hand
FROM warehouse w
JOIN stock_balance sb ON sb.warehouse_id = w.warehouse_id
WHERE sb.qty_on_hand IS NOT NULL
GROUP BY w.code
ORDER BY w.code;
```

code	total_on_hand
WH-A	225
WH-N	190
WH-K	15

SUM складывает значения в каждой группе.

Тип запроса: MAX

```sql
SELECT MAX(sb.qty_on_hand) AS max_qty
FROM stock_balance sb
WHERE sb.warehouse_id = 1;
```

max_qty
120

MAX возвращает максимальное значение.

Тип запроса: MIN

```
SELECT MIN(sb.qty_on_hand) AS min_qty
FROM stock_balance sb
WHERE sb.warehouse_id = 1
  AND sb.qty_on_hand IS NOT NULL;
```

min_qty
0

MIN возвращает минимальное значение.

Тип запроса: AVG

```
SELECT p.category,
       AVG(p.base_price) AS avg_price
FROM product p
GROUP BY p.category
ORDER BY p.category;
```

category	avg_price
Кабели	1050.00
Канц	250.00
Крепёж	6.25

AVG считает среднее, NULL не учитывается.

Тип запроса: COUNT

```sql
SELECT COUNT(sb.qty_on_hand) AS known_qty_rows
FROM stock_balance sb
WHERE sb.warehouse_id = 1;
```

known_qty_rows
4

COUNT(col) считает только строки, где `col` не NULL.

Тип запроса: `COUNT(*)`

```sql
SELECT COUNT(*) AS rows_total
FROM stock_balance sb
WHERE sb.warehouse_id = 1;
```

rows_total
5

`COUNT(*)` считает все строки, включая NULL в столбцах.

`COUNT(col)` vs `COUNT(*)`

- `COUNT(col)` не считает `NULL`
- `COUNT(*)` считает строки

GROUP BY: правило

Если в SELECT есть неагрегированный столбец, он должен быть в GROUP BY.

Построение запроса: шаг 1

Нужен отчет по складам: суммарный остаток и число SKU. Начинаем с базовых таблиц.

```
SELECT w.code, sb.product_id, sb.qty_on_hand
FROM warehouse w
LEFT JOIN stock_balance sb ON sb.warehouse_id = w.warehouse_id;
```

code	product_id	qty_on_hand
WH-A	1	120
WH-A	2	80
WH-A	3	NULL
WH-A	4	25
WH-A	5	0
WH-N	NULL	NULL
WH-K	NULL	NULL

Построение запроса: шаг 2

Добавляем агрегацию.

```
SELECT w.code,
       SUM(COALESCE(sb.qty_on_hand, 0)) AS total_on_hand
FROM warehouse w
LEFT JOIN stock_balance sb ON sb.warehouse_id = w.warehouse_id
GROUP BY w.code;
```

code	total_on_hand
WH-A	225
WH-N	0
WH-K	0

Построение запроса: шаг 3

Добавляем COUNT DISTINCT.

```
SELECT w.code,
       SUM(COALESCE(sb.qty_on_hand, 0)) AS total_on_hand,
       COUNT(DISTINCT sb.product_id) AS distinct_products
FROM warehouse w
LEFT JOIN stock_balance sb ON sb.warehouse_id = w.warehouse_id
GROUP BY w.code;
```

code	total_on_hand
distinct_products	
WH-A	225
5	WH-N
0	0
WH-K	0
0	

Построение запроса: шаг 4

Добавляем фильтр по суммарному остатку.

```
SELECT w.code,
       SUM(COALESCE(sb.qty_on_hand, 0)) AS total_on_hand,
       COUNT(DISTINCT sb.product_id) AS distinct_products
FROM warehouse w
LEFT JOIN stock_balance sb ON sb.warehouse_id = w.warehouse_id
GROUP BY w.code
HAVING SUM(COALESCE(sb.qty_on_hand, 0)) > 50;
```

code	total_on_hand	distinct_products
WH-A	225	5

Тип запроса: HAVING

Вход:

```
SELECT p.category,
       COUNT(*) AS products_count
FROM product p
GROUP BY p.category
HAVING COUNT(*) >= 2
ORDER BY p.category;
```

category	products_count
Кабели	2
Крепёж	2

HAVING: отличие от WHERE

- WHERE фильтрует строки
- HAVING фильтрует группы

Правильно (фильтруем группы):

```
SELECT p.category, COUNT(*) AS cnt
FROM product p
GROUP BY p.category
HAVING AVG(p.base_price) > 500;
```

category	cnt
Кабели	2

GROUP BY по нескольким столбцам

```
SELECT w.code, p.category,
       SUM(COALESCE(sb.qty_on_hand, 0)) AS total_on_hand
FROM warehouse w
LEFT JOIN stock_balance sb ON sb.warehouse_id = w.warehouse_id
LEFT JOIN product p ON p.product_id = sb.product_id
GROUP BY w.code, p.category
ORDER BY w.code, p.category;
```

code	category	total_on_hand
WH-A	Кабели	25
WH-A	Канц	0
WH-A	Крепёж	200
WH-N	NULL	NULL
WH-K	NULL	NULL

HAVING: пример с суммой

```
SELECT w.code,
       SUM(COALESCE(sb.qty_reserved, 0)) AS total_reserved
FROM warehouse w
LEFT JOIN stock_balance sb ON sb.warehouse_id = w.warehouse_id
GROUP BY w.code
HAVING SUM(COALESCE(sb.qty_reserved, 0)) > 10
ORDER BY total_reserved DESC;
```


Агрегация и NULL

- SUM/AVG игнорируют NULL
- для отчета часто нужен COALESCE

```
-- Данные: 10, 20, NULL, 30
SELECT SUM(price);   -- вернёт 60  (не 60+NULL)
SELECT AVG(price);   -- вернёт 20  (10+20+30) / 3 = 20
SELECT MAX(price);   -- вернёт 30
SELECT MIN(price);   -- вернёт 10
SELECT COUNT(price); -- вернёт 3   (не 4!)
SELECT COUNT(*);     -- вернёт 4, считает все строки включая NULL
SELECT MIN(COALESCE(price, 0)); --вернёт 0, NULL заменяется на 0
SELECT AVG(COALESCE(price, 0)) -- вернёт 15 (10 + 20 + 0 + 30) / 4 = 15
```

Что делает этот запрос?

```
SELECT w.code,
       COUNT(*) AS rows_in_balance,
       SUM(COALESCE(sb.qty_on_hand, 0)) AS total_on_hand
FROM warehouse w
LEFT JOIN stock_balance sb ON sb.warehouse_id = w.warehouse_id
GROUP BY w.code
ORDER BY w.code;
```

Отчетный пример: остатки по складам

```sql
SELECT w.code,
       COUNT(*) AS rows_in_balance,
       SUM(COALESCE(sb.qty_on_hand, 0)) AS total_on_hand
FROM warehouse w
LEFT JOIN stock_balance sb ON sb.warehouse_id = w.warehouse_id
GROUP BY w.code
ORDER BY w.code;
```

code	rows_in_balance	total_on_hand
WH-A	5	225
WH-N	1	0
WH-K	1	0

Что делает этот запрос?

```
SELECT p.sku, p.name,
       SUM(COALESCE(sb.qty_on_hand, 0)) AS total_on_hand
FROM product p
LEFT JOIN stock_balance sb ON sb.product_id = p.product_id
GROUP BY p.sku, p.name
ORDER BY total_on_hand DESC, p.sku
FETCH FIRST 2 ROWS ONLY;
```


Отчетный пример: топ товаров

```
SELECT p.sku, p.name,
       SUM(COALESCE(sb.qty_on_hand, 0)) AS total_on_hand
FROM product p
LEFT JOIN stock_balance sb ON sb.product_id = p.product_id
GROUP BY p.sku, p.name
ORDER BY total_on_hand DESC, p.sku
FETCH FIRST 2 ROWS ONLY;
```

sku	name	total_on_hand
SKU-001	Гайка M8	120
SKU-002	Болт M8x30	80

Что делает этот запрос?

```
SELECT w.code,
       SUM(sm.qty_delta) AS total_delta
FROM warehouse w
LEFT JOIN stock_move sm ON sm.warehouse_id = w.warehouse_id
GROUP BY w.code
ORDER BY w.code;
```


Перемещения по складам

```sql
SELECT w.code,
       SUM(sm.qty_delta) AS total_delta
FROM warehouse w
LEFT JOIN stock_move sm ON sm.warehouse_id = w.warehouse_id
GROUP BY w.code
ORDER BY w.code;
```

code	total_delta
WH-A	27
WH-N	60
WH-K	NULL

Что делает этот запрос?

```
SELECT sm.reason, COUNT(*) AS moves_count
FROM stock_move sm
WHERE sm.warehouse_id = 1
  AND sm.moved_at BETWEEN TIMESTAMP '2026-01-10 00:00:00'
                      AND TIMESTAMP '2026-01-12 23:59:59'
GROUP BY sm.reason
ORDER BY sm.reason;
```


Перемещения по складу с периодом

```sql
SELECT sm.reason, COUNT(*) AS moves_count
FROM stock_move sm
WHERE sm.warehouse_id = 1
  AND sm.moved_at BETWEEN TIMESTAMP '2026-01-10 00:00:00'
                      AND TIMESTAMP '2026-01-12 23:59:59'
GROUP BY sm.reason
ORDER BY sm.reason;
```

reason	moves_count
RECEIPT	1
SHIPMENT	1

Что делает этот запрос?

```
SELECT w.code,
       SUM(CASE WHEN sm.qty_delta > 0 THEN sm.qty_delta ELSE 0 END) AS
total_in,
       SUM(CASE WHEN sm.qty_delta < 0 THEN -sm.qty_delta ELSE 0 END) AS
total_out
FROM warehouse w
LEFT JOIN stock_move sm ON sm.warehouse_id = w.warehouse_id
GROUP BY w.code
ORDER BY w.code;
```

Пример с CASE WHEN

```
SELECT w.code,
       SUM(CASE WHEN sm.qty_delta > 0 THEN sm.qty_delta ELSE 0 END) AS
total_in,
       SUM(CASE WHEN sm.qty_delta < 0 THEN -sm.qty_delta ELSE 0 END) AS
total_out
FROM warehouse w
LEFT JOIN stock_move sm ON sm.warehouse_id = w.warehouse_id
GROUP BY w.code
ORDER BY w.code;
```

code	total_in	total_out
WH-A	50	23
WH-N	60	0
WH-K	0	0

Современные возможности: LATERAL

```sql
SELECT w.code, t.sku, t.qty_on_hand
FROM warehouse w
LEFT JOIN LATERAL (
  SELECT p.sku, sb.qty_on_hand
  FROM stock_balance sb
  JOIN product p ON p.product_id = sb.product_id
  WHERE sb.warehouse_id = w.warehouse_id
  ORDER BY sb.qty_on_hand DESC NULLS LAST
  FETCH FIRST 1 ROW ONLY
) t ON TRUE
ORDER BY w.code;
```

LATERAL позволяет подзапросу использовать поля текущей строки из `w`. Типичный кейс: взять «лучший»/«первый» элемент для каждой строки слева.

Современные возможности: LATERAL

code	sku	qty_on_hand
WH-A	SKU-001	120
WH-N	NULL	NULL
WH-K	NULL	NULL

Современные возможности: FILTER для агрегатов


```
SELECT w.code,
       COUNT(*) AS rows_total,
       COUNT(*) FILTER (WHERE COALESCE(sb.qty_on_hand, 0) > 0) AS
rows_positive
FROM warehouse w
LEFT JOIN stock_balance sb ON sb.warehouse_id = w.warehouse_id
GROUP BY w.code
ORDER BY w.code;
```

FILTER задает условие только для конкретного агрегата, без дублирования запроса.

code	rows_total	rows_positive
WH-A	5	3
WH-N	1	0
WH-K	1	0

Современные возможности: GROUPING SETS

```
SELECT w.code, p.category,
       SUM(COALESCE(sb.qty_on_hand, 0)) AS total_on_hand
FROM warehouse w
LEFT JOIN stock_balance sb ON sb.warehouse_id = w.warehouse_id
LEFT JOIN product p ON p.product_id = sb.product_id
GROUP BY GROUPING SETS ((w.code), (p.category), ());
```

GROUPING SETS позволяет получить несколько уровней агрегации одним запросом.

Современные возможности: GROUPING SETS

code	category	total_on_hand
WH-A	NULL	225
WH-N	NULL	0
WH-K	NULL	0
NULL	Кабели	25
NULL	Канц	0
NULL	Крепёж	200
NULL	NULL	0
NULL	NULL	225

ROLLUP и CUBE: общая идея

ROLLUP и CUBE расширяют `GROUP BY` и добавляют строки-итоги.

Строки с `NULL` в группирующих колонках — это итоги:

- `code = NULL` означает «итог по всем складам»
- `category = NULL` означает «итог по всем категориям»

Базовая группировка (без итогов)

```
SELECT w.code, p.category,
       SUM(COALESCE(sb.qty_on_hand, 0)) AS total_on_hand
FROM warehouse w
LEFT JOIN stock_balance sb ON sb.warehouse_id = w.warehouse_id
LEFT JOIN product p ON p.product_id = sb.product_id
GROUP BY w.code, p.category
ORDER BY w.code, p.category;
```

code	category	total_on_hand
WH-A	Кабели	25
WH-A	Крепёж	200
WH-N	Кабели	10
WH-N	Крепёж	180
WH-K	Канц	15

ROLLUP: иерархические итоги

```sql
SELECT w.code, p.category,
       SUM(COALESCE(sb.qty_on_hand, 0)) AS total_on_hand
FROM warehouse w
LEFT JOIN stock_balance sb ON sb.warehouse_id = w.warehouse_id
LEFT JOIN product p ON p.product_id = sb.product_id
GROUP BY ROLLUP (w.code, p.category);
```

ROLLUP строит итоги слева направо:

1. `code + category`
2. только `code`
3. общий итог

ROLLUP: иерархические итоги

code	category	total_on_hand
WH-A	Кабели	25
WH-A	Крепёж	200
WH-A	NULL	225
WH-N	Кабели	10
WH-N	Крепёж	180
WH-N	NULL	190
WH-K	Канц	15
WH-K	NULL	15
NULL	NULL	430

CUBE: все комбинации

```sql
SELECT w.code, p.category,
       SUM(COALESCE(sb.qty_on_hand, 0)) AS total_on_hand
FROM warehouse w
LEFT JOIN stock_balance sb ON sb.warehouse_id = w.warehouse_id
LEFT JOIN product p ON p.product_id = sb.product_id
GROUP BY CUBE (w.code, p.category);
```

CUBE строит все комбинации:

1. code + category
2. только code
3. только category
4. общий итог

CUBE: все комбинации

code	category	total_on_hand
WH-A	Кабели	25
WH-A	Крепёж	200
WH-A	NULL	225
WH-N	Кабели	10
WH-N	Крепёж	180
WH-N	NULL	190
WH-K	Канц	15
WH-K	NULL	15
NULL	Кабели	35
NULL	Крепёж	380
NULL	Канц	15
NULL	NULL	430

Как выбирать ROLLUP или CUBE

- ROLLUP: нужны итоги по иерархии (склад → общий итог)
- CUBE: нужны все срезы (и по складу, и по категории отдельно)
- Если в `GROUP BY` один столбец, ROLLUP и CUBE дадут одинаковый результат