Основы РСУБД

структура, запросы, транзакции

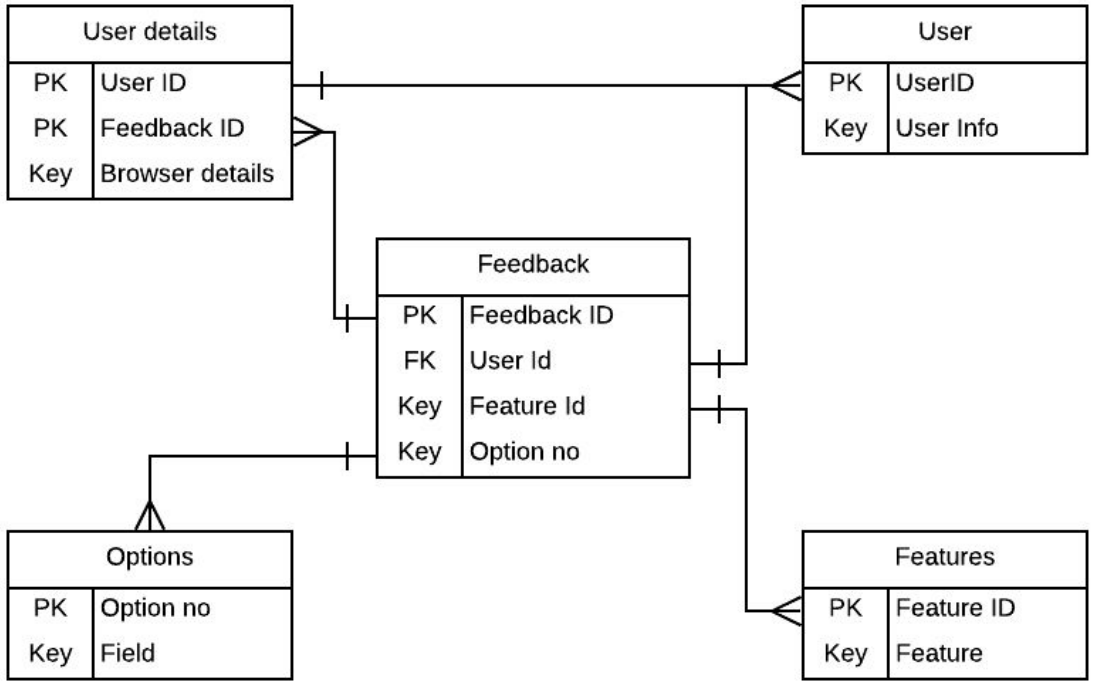
Реляционная база данных

Реляционная БД представляет собой совокупность именованных двумерных таблиц данных, логически связанных (находящихся в отношении) между собой

Таблицы и строки

Таблицы состоят из строк и именованных столбцов, **строки** представляют собой экземпляры информационного объекта, **столбцы** – атрибуты объекта. Строки иногда называют **записями**, а столбцы **полями записи**.

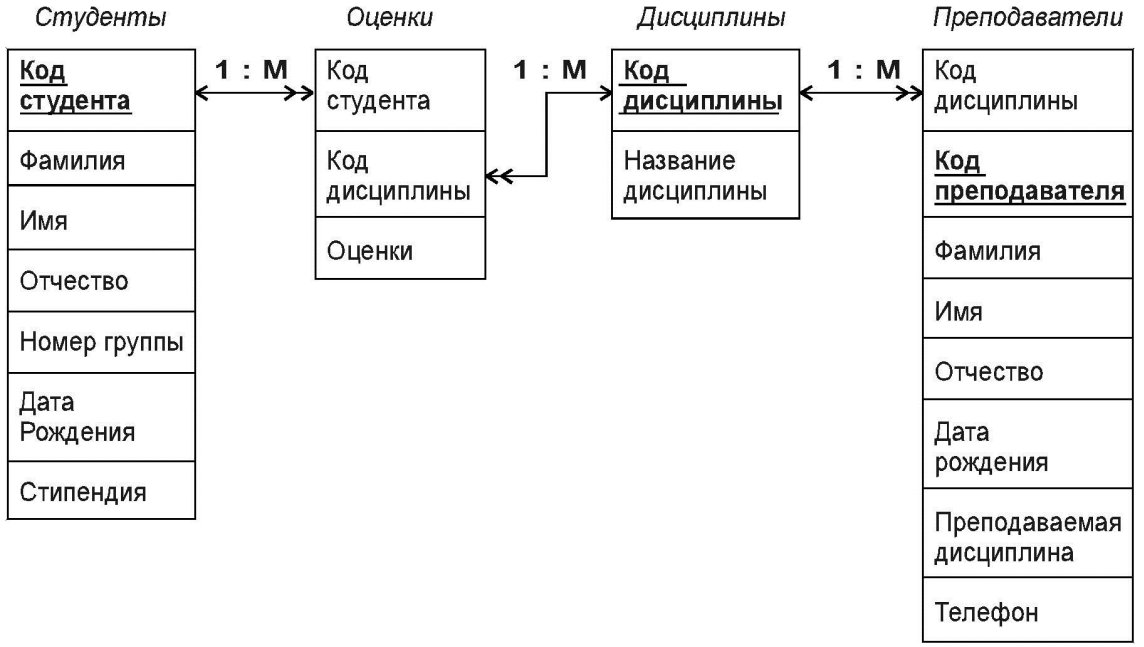
Таким образом, в реляционной модели все данные представлены для пользователя в виде таблиц значений данных, и все операции над базой сводятся к манипулированию таблицами.



Ключи

Первичный ключ (главный ключ, primary key, PK) – столбец или совокупность столбцов, значения которых однозначно идентифицируют строки.

Вторичный ключ (внешний, foreign key, FK) – столбец или совокупность столбцов, которые в данной таблице не являются первичными ключами, но являются первичными ключами в другой таблице.

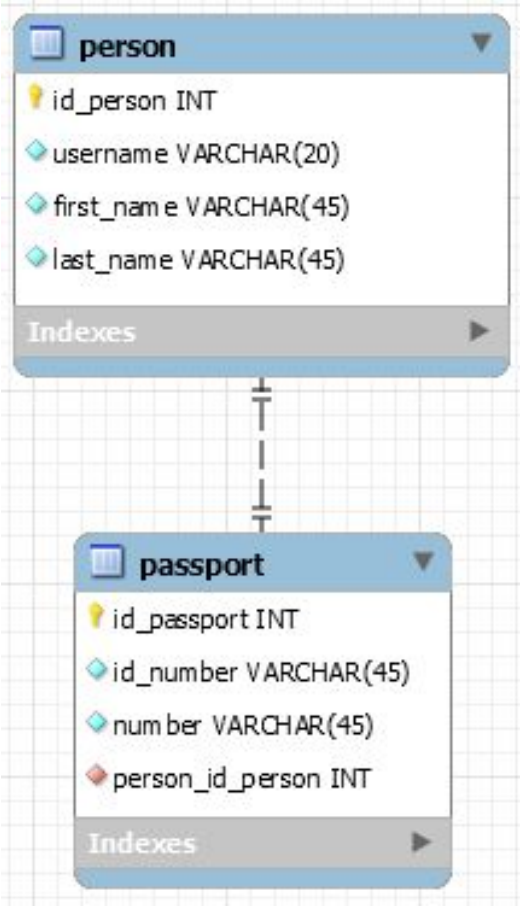


Типы связей между таблицами

Связь **один к одному** (1:1) – это когда одной записи в таблице отвечает только одна запись из другой таблицы.

Связь один к одному образуется, когда ключевой столбец (идентификатор) присутствует в другой таблице, в которой тоже является ключом либо свойствами столбца задана его уникальность (одно и тоже значение не может повторяться в разных строках).

Пример: человек-паспорт

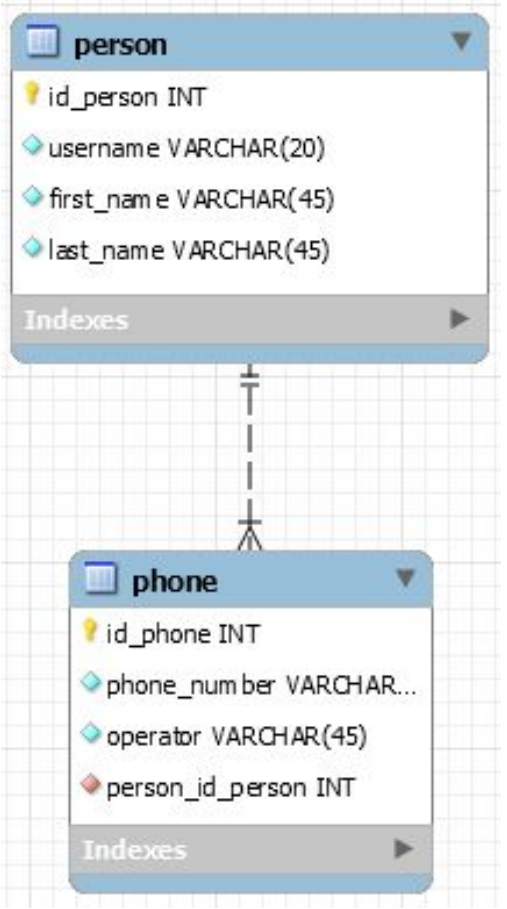


Типы связей между таблицами

Связь **один ко многим** (1:M) – одной записи в таблице может соответствовать множество записей другой таблицы.

Самая распространенная связь между таблицами.

Пример: человек-телефон

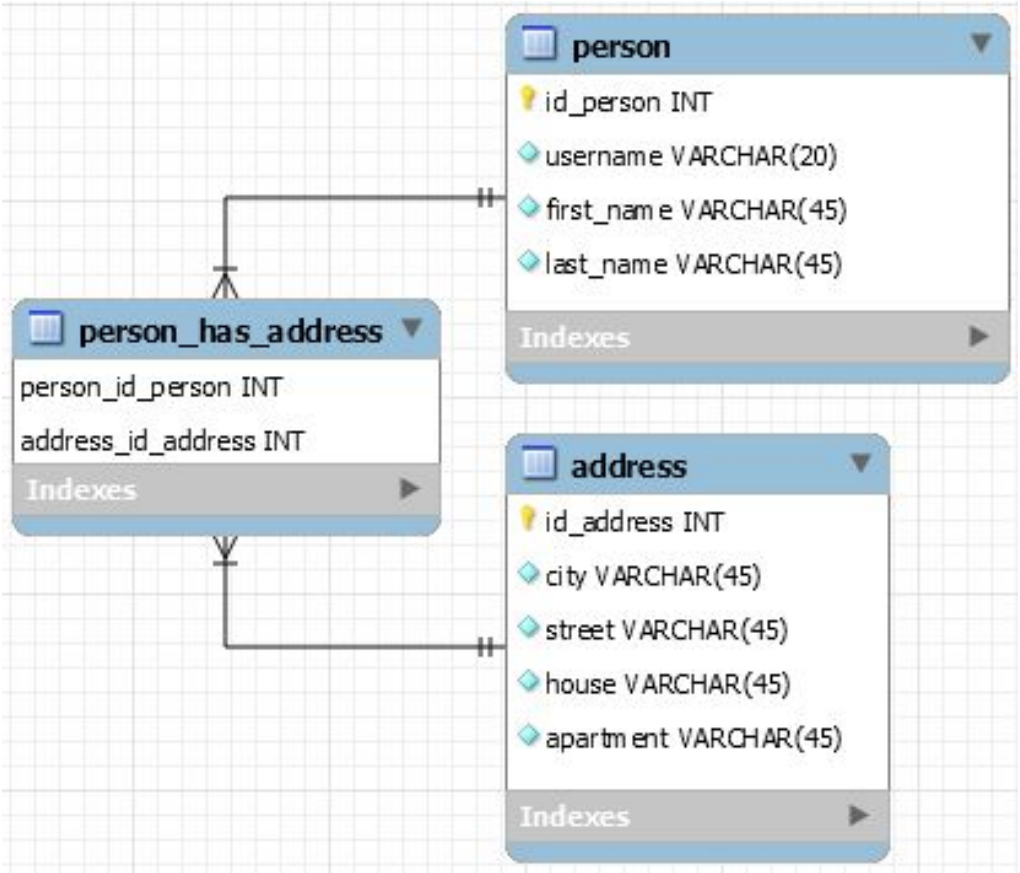


Типы связей между таблицами

Связь **многие ко многим** (M:M) – нескольким записям из одной таблицы соответствует несколько записей из другой таблицы.

Организовывается посредством связывающей таблицы.

Пример: человек-адрес



Вопрос-ответ

? Обязательно ли мне задавать связи между таблицами?

✔ Связи между отдельными таблицами в реляционной модели в явном виде могут не описываться

✔ Они устанавливаются пользователем при написании запроса на выборку данных и представляют собой условия равенства значений соответствующих полей

SQL

SQL – декларативный язык, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

Ограничения полей

Для полей (атрибутов) используются следующие виды ограничений:

- Тип и формат поля
- Задание диапазона значений
- Недопустимость пустого поля
- Проверка на уникальность

Ограничения таблицы

PRIMARY KEY (имя столбца, …)

UNIQUE (имя столбца, …)

FOREIGN KEY (имя столбца, …) **REFERENCES** имя таблицы [имя столбца, …][ссылочная спецификация]

CHECK предикат

DEFAULT = значение по умолчанию

NOT NULL

Ссылочная спецификация:

[ON UPDATE {CASCADE | SET NULL | SET DEFAULT | RESTRICTED | NO ACTION}]

[ON DELETE {CASCADE | SET NULL | SET DEFAULT | RESTRICTED | NO ACTION}]

Операторы SQL

Операторы определения данных (Data Definition Language, DDL):

- **CREATE** создаёт объект базы данных (саму базу, таблицу, представление, пользователя и так далее)
- **ALTER** изменяет объект
- **DROP** удаляет объект

Операторы SQL

Операторы манипуляции данными (Data Manipulation Language, DML):

- **SELECT** выбирает данные, удовлетворяющие заданным условиям
- **INSERT** добавляет новые данные
- **UPDATE** изменяет существующие данные
- **DELETE** удаляет данные

Операторы SQL

Операторы определения доступа к данным (Data Control Language, DCL):

- **GRANT** предоставляет пользователю (группе) разрешения на определённые операции с объектом
- **REVOKE** отзывает ранее выданные разрешения
- **DENY** задаёт запрет, имеющий приоритет над разрешением

Операторы SQL

Операторы управления транзакциями (Transaction Control Language, TCL):

- **COMMIT** применяет транзакцию
- **ROLLBACK** откатывает все изменения, сделанные в контексте текущей транзакции
- **SAVEPOINT** делит транзакцию на более мелкие участки

Создание базы данных

Для создания БД используются команды:

- **CREATE DATABASE** <имя>;
- **CREATE SCHEMA** <имя>;

```
1 CREATE DATABASE db_name
2 CHARACTER SET utf8
3 COLLATE utf8_general_ci;
```

```
1 CREATE SCHEMA db_name
2 CHARACTER SET utf8
3 COLLATE utf8_general_ci;
```

Удаление базы данных

Для удаления БД используются команды:

- **DROP DATABASE** <имя>;
- **DROP SCHEMA** <имя>;

```
1 DROP DATABASE db_name;
```

```
1 DROP SCHEMA db_name;
```

Создание таблицы

Для создания таблицы используется команда:

- **CREATE TABLE** <имя> (<имя столбца> <тип> <атрибуты>, …);

```
CREATE TABLE books(
    id   INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    title     VARCHAR(50) NOT NULL,
    author    VARCHAR(50) NOT NULL,
    price     DECIMAL(15,2),
    edition   TINYINT UNSIGNED DEFAULT '1' NOT NULL,
    page_num  SMALLINT UNSIGNED,
    publish_year   SMALLINT(4),
    creation_date  TIMESTAMP
);
```

Удаление таблицы

Для удаления таблицы используется команда:

- **DROP TABLE** <имя>;

```
1 DROP TABLE table_name;
```

Изменение таблицы

Команда ALTER TABLE позволяет добавлять новые столбцы и ограничения целостности, удалять их, менять типы столбцов, переименовывать столбцы.

```
1
2 ALTER TABLE books ADD discount TINYINT UNSIGNED, ADD amount SMALLINT UNSIGNED;
3 ALTER TABLE books ADD shelf_position VARCHAR(20) AFTER Price;
4
5 ALTER TABLE books DROP edition;
6
7 ALTER TABLE books MODIFY COLUMN Price DECIMAL(15,2) AFTER Author;
8
9 ALTER TABLE books CHANGE COLUMN creation_date entry_date DATE;
10
11 ALTER TABLE books ALTER discount SET DEFAULT 0;
12 ALTER TABLE books ALTER dicount DROP DEFAULT;
13
14 ALTER TABLE books RENAME TO books_collection;
15 ALTER TABLE books TYPE = MYISAM;
16 ALTER TABLE books COMMENT = 'Список продаваемых книг.';
```

Выборка данных

SELECT [**DISTINCT**] <список столбцов>

FROM <имя таблицы>

[**JOIN** <имя таблицы> **ON** <условия связывания>]

[**WHERE** <условия выборки>]

[**GROUP BY** <столбцы для группировки>]

[**HAVING** <условия выборки групп>]

[**ORDER BY** <список столбцов для сортировки>]

[**LIMIT** <число>]

Алиасы

Алиасы - временные имена, которые можно задать таблицам и колонкам, на момент выполнения запроса.

Алиасы позволяют задавать более читабельные и простые имена.

Чаще всего алиасы применяются когда:

- нужно упростить имя таблицы или колонки (вплоть до одной буквы).
- в запросе используются функции.
- имя колонки очень большое или плохо читаемое.
- несколько колонок скомбинированы вместе.

```sql
SELECT column1, ... , columnN FROM table_name AS alias;

SELECT column1 AS 'alias', ... , columnN AS 'aliasN' FROM table_name;
```

Пример select

```
mysql> SELECT title, author, price, status FROM books, orders WHERE books.id=book_id;
+---------------------+--------------------+--------+-----------------+
| title               | author             | price  | status          |
+---------------------+--------------------+--------+-----------------+
| Мастер и Маргарита  | Михаил Булгаков    | 263.00 | Новый           |
| Дубровский (Акция)  | Александр Пушкин   | 230.00 | Обрабатывается  |
+---------------------+--------------------+--------+-----------------+
2 rows in set (0.00 sec)
```

Условия

WHERE [**NOT**] <условие 1> [**AND** | **OR**] <условие 2>]

Условия:

- <столбец, константа или выражение> <оператор сравнения> <столбец, константа или выражение>
- **IS** [**NOT**] **NULL**
- [**NOT**] **LIKE** <шаблон>
- [**NOT**] **IN** (<список значений>)
- [**NOT**] **BETWEEN** <нижняя граница> **AND** <верхняя граница>

Операторы сравнения

Оператор	Значение
=	Равенство
<>, !=	Неравенство
>	Больше
<	Меньше
>=	Больше либо равно
<=	Меньше либо равно
BETWEEN	Между двумя значениями
LIKE	Соответствует шаблону
IN	Соответствует набору значений

Логические операторы

Операторы **AND**, **OR**, **NOT**, **XOR** используются для создания нескольких условий (фильтров) вывода и обработки записей таблиц базы данных

Оператор	Значение
AND	Правда, если оба условия принимают истинное значение
OR	Правда, если одно из условий принимает истинное значение
NOT	Вывод всех значений, которые не соответствуют условию
XOR	Правда, если одно из условий принимает истинное значение. Неправда, если оба условия истинны

Пример where

```
mysql> SELECT id, title, author, price, amount FROM books WHERE amount=3;
+----+---------------------------+--------------------+--------+--------+
| id | title                     | author             | price  | amount |
+----+---------------------------+--------------------+--------+--------+
|  4 | Мёртвые души (Акция)      | Николай Гоголь     | 173.00 |      3 |
|  5 | Преступление и наказание  | Фёдор Достоевский  | 245.00 |      3 |
|  8 | Отцы и дети               | Иван Тургенев      | 371.00 |      3 |
+----+---------------------------+--------------------+--------+--------+
3 rows in set (0.00 sec)
```

Like

LIKE используется для поиска записей, данные которых совпадают с заданным шаблоном.

Для создания шаблона используются два специальных символа:

- «%» - представляет собой неопределенное количество любых символов
- «_» - представляет только один любой символ

Like

Шаблон	Описание
LIKE 'a%'	Значение начинается с символа "a"
LIKE '%a'	Значение заканчивается на символ "a"
LIKE '%a%'	Значение имеет символ "a" в любом месте
LIKE '_a%'	Значение имеет символ "a" на втором месте
LIKE 'a__'	Значение состоит из 3 символов, первый "a"
LIKE 'a%d'	Значение начинается с символа "a" и заканчивается на символ "d"

Пример Like

```
mysql> SELECT id, title, author, price, shelf_position
    -> FROM books
    -> WHERE title LIKE '%Акция%';
+----+--------------------------+--------------------+--------+----------------+
| id | title                    | author             | price  | shelf_position |
+----+--------------------------+--------------------+--------+----------------+
|  1 | Дубровский (Акция)       | Александр Пушкин   | 230.00 | 023A12         |
|  2 | Нос (Акция)              | Николай Гоголь     | 255.20 | 003C05         |
|  4 | Мёртвые души (Акция)     | Николай Гоголь     | 173.00 | 007A15         |
+----+--------------------------+--------------------+--------+----------------+
3 rows in set (0.00 sec)
```

Regexp

Поиск значений по шаблонам также возможен с помощью регулярных выражений, используя оператор **REGEXP**.

RLIKE является синонимом для оператора REGEXP.

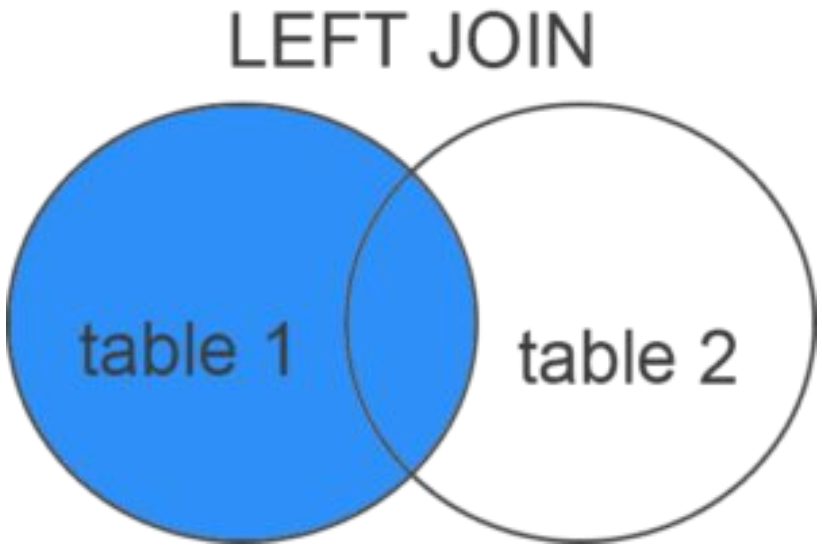
```
mysql> SELECT id, title, author, price, shelf_position
    -> FROM books
    -> WHERE shelf_position REGEXP '^004.*';
+----+---------------------------+--------------------+--------+----------------+
| id | title                     | author             | price  | shelf_position |
+----+---------------------------+--------------------+--------+----------------+
|  3 | Мастер и Маргарита        | Михаил Булгаков    | 263.00 | 004D11         |
|  5 | Преступление и наказание  | Фёдор Достоевский  | 245.00 | 004E08         |
|  7 | Анна Каренина             | Лев Толстой        | 346.00 | 004D05         |
+----+---------------------------+--------------------+--------+----------------+
3 rows in set (0.00 sec)
```

JOIN

Тип	Результат
JOIN	Записи, значения полей в которых совпадают
LEFT JOIN	Все записи из таблицы A и соответствующие им записи из таблицы B. Если соответствия нет, поля из B будут пустыми
RIGHT JOIN	Все записи из таблицы B и соответствующие им записи из таблицы A. Если соответствия нет, поля из A будут пустыми
FULL JOIN	Все записи из A и соответствующие им записи из B. Если соответствия нет, то поля из B будут пустыми. Записи из B, которым не нашлось пары в A, тоже будут присутствовать в результирующем наборе. В этом случае поля из A будут пустыми.
CROSS JOIN	Результирующий набор содержит все варианты комбинаций строк из A и B. Условное соединение при этом не указывается

LEFT JOIN

LEFT JOIN возвращает все записи из левой (первой) таблицы и записи из правой (второй) таблицы, в которых были найдены совпадения.



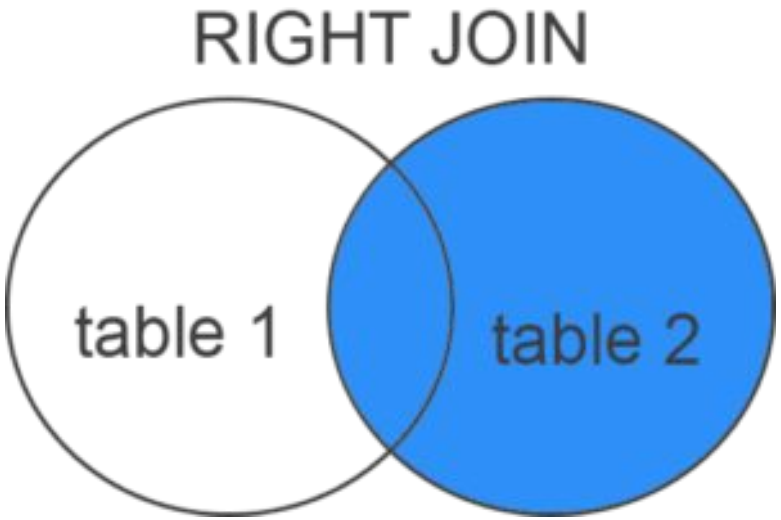
Пример LEFT JOIN

```
mysql> SELECT id, first_name, last_name FROM customers;
+----+----------------+----------------+
| id | first_name     | last_name      |
+----+----------------+----------------+
|  1 | Олег           | Пальшин        |
|  2 | Jane           | Doherty        |
|  3 | Евгений        | Серов          |
|  4 | София          | Молина         |
|  5 | John           | Doe            |
+----+----------------+----------------+
5 rows in set (0.00 sec)

mysql> SELECT id, customer_id, employer_id, status FROM orders;
+----+-------------+-------------+------------------+
| id | customer_id | employer_id | status           |
+----+-------------+-------------+------------------+
|  3 |           1 |           3 | Готов к отправке |
|  6 |           3 |           2 | Готов к отправке |
|  7 |           5 |           2 | Завершен         |
|  9 |           3 |           3 | Обрабатывается   |
| 10 |           5 |           3 | Готов к отправке |
+----+-------------+-------------+------------------+
5 rows in set (0.00 sec)
```

```
mysql> SELECT orders.id, customers.first_name, customers.last_name, orders.employer_id, orders.status
    -> FROM customers
    -> LEFT JOIN orders ON orders.customer_id = customers.id;
+------+------------+-----------+-------------+------------------+
| id   | first_name | last_name | employer_id | status           |
+------+------------+-----------+-------------+------------------+
| NULL | София      | Молина    |        NULL | NULL             |
| NULL | Jane       | Doherty   |        NULL | NULL             |
|    3 | Олег       | Пальшин   |           3 | Готов к отправке |
|    6 | Евгений    | Серов     |           2 | Готов к отправке |
|    7 | John       | Doe       |           2 | Завершен         |
|    9 | Евгений    | Серов     |           3 | Обрабатывается   |
|   10 | John       | Doe       |           3 | Готов к отправке |
+------+------------+-----------+-------------+------------------+
7 rows in set (0.00 sec)
```

RIGHT JOIN

RIGHT JOIN возвращает все записи из правой (второй) таблицы и записи из левой (первой) таблицы, в которых были найдены совпадения.



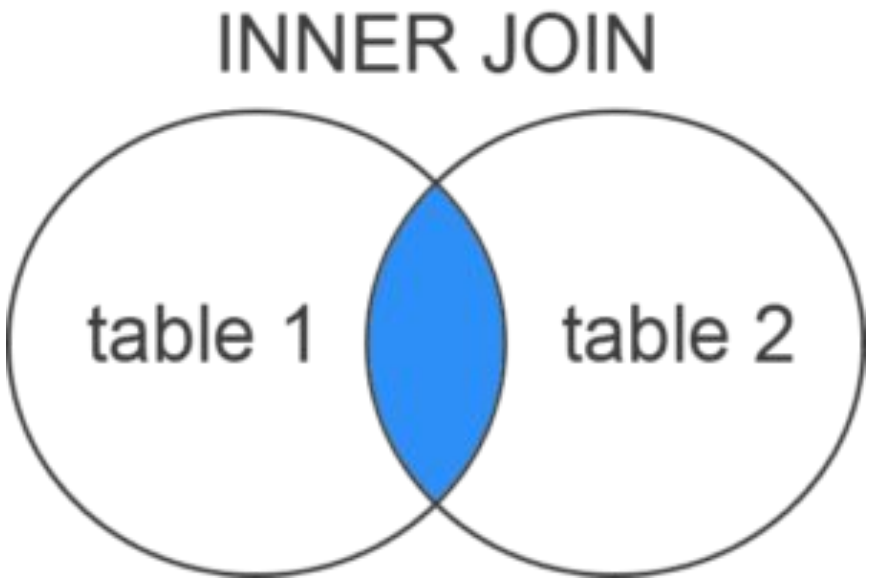
Пример RIGHT JOIN

```
mysql> SELECT id, first_name, last_name, position
    -> FROM employees;
+----+------------+-----------+------------------+
| id | first_name | last_name | position         |
+----+------------+-----------+------------------+
|  1 | Абросим    | Сумароков | Ген. Директор    |
|  2 | Александр  | Суматохин | Старший продавец |
|  3 | Петр       | Стропин   | Продавец         |
|  4 | Фёдор      | Телецкий  | Кладовщик        |
|  5 | Аркадий    | Прошин    | Продавец         |
+----+------------+-----------+------------------+
5 rows in set (0.00 sec)

mysql> SELECT id, customer_id, employee_id, status
    -> FROM orders;
+----+-------------+-------------+------------------+
| id | customer_id | employee_id | status           |
+----+-------------+-------------+------------------+
|  3 |           1 |           3 | Готов к отправке |
|  6 |           3 |           2 | Готов к отправке |
|  7 |           5 |           2 | Завершен         |
|  9 |           3 |           3 | Обрабатывается   |
| 10 |           5 |           3 | Готов к отправке |
+----+-------------+-------------+------------------+
5 rows in set (0.00 sec)
```

```
mysql> SELECT orders.id, employees.first_name, employees.last_name, employees.position, orders.status
    -> FROM orders
    -> RIGHT JOIN employees ON orders.employee_id = employees.id;
+------+------------+-----------+------------------+------------------+
| id   | first_name | last_name | position         | status           |
+------+------------+-----------+------------------+------------------+
| NULL | Фёдор      | Телецкий  | Кладовщик        | NULL             |
| NULL | Абросим    | Сумароков | Ген. Директор    | NULL             |
| NULL | Аркадий    | Прошин    | Продавец         | NULL             |
|    3 | Петр       | Стропин   | Продавец         | Готов к отправке |
|    6 | Александр  | Суматохин | Старший продавец | Готов к отправке |
|    7 | Александр  | Суматохин | Старший продавец | Завершен         |
|    9 | Петр       | Стропин   | Продавец         | Обрабатывается   |
+------+------------+-----------+------------------+------------------+
7 rows in set (0.00 sec)
```

INNER JOIN

INNER JOIN позволяет объединить несколько таблиц, которые имеют одинаковые данные в одной или нескольких колонках.



Пример INNER JOIN

```
mysql> SELECT id, first_name, last_name
    -> FROM customers;
+----+----------------+----------------+
| id | first_name     | last_name      |
+----+----------------+----------------+
|  1 | Олег           | Пальшин        |
|  2 | Jane           | Doherty        |
|  3 | Евгений        | Серов          |
|  4 | София          | Молина         |
|  5 | John           | Doe            |
+----+----------------+----------------+
5 rows in set (0.00 sec)

mysql> SELECT id, customer_id, employer_id, status, order_date
    -> FROM orders;
+----+-------------+-------------+------------------+---------------------+
| id | customer_id | employer_id | status           | order_date          |
+----+-------------+-------------+------------------+---------------------+
|  3 |           1 |           3 | Готов к отправке | 2019-01-05 04:55:58 |
|  6 |           3 |           3 | Готов к отправке | 2019-01-15 14:56:12 |
|  7 |           2 |           2 | Завершен         | 2019-01-11 20:59:40 |
|  9 |           4 |           3 | Обрабатывается   | 2018-12-22 21:16:16 |
| 10 |           2 |           3 | Готов к отправке | 2018-12-24 04:28:54 |
+----+-------------+-------------+------------------+---------------------+
5 rows in set (0.00 sec)
```

```
mysql> SELECT orders.id, customers.first_name, customers.last_name, orders.status, orders.order_date
    -> FROM orders
    -> INNER JOIN customers ON orders.customer_id = customers.id;
+----+------------+-----------+------------------+---------------------+
| id | first_name | last_name | status           | order_date          |
+----+------------+-----------+------------------+---------------------+
|  3 | Олег       | Пальшин   | Готов к отправке | 2019-01-05 04:55:58 |
|  6 | Евгений    | Серов     | Готов к отправке | 2019-01-15 14:56:12 |
|  7 | Jane       | Doherty   | Завершен         | 2019-01-11 20:59:40 |
|  9 | София      | Молина    | Обрабатывается   | 2018-12-22 21:16:16 |
| 10 | Jane       | Doherty   | Готов к отправке | 2018-12-24 04:28:54 |
+----+------------+-----------+------------------+---------------------+
5 rows in set (0.00 sec)
```

Сортировка

ORDER BY <столбец> [**ASC** | **DESC**] [, <столбец 2> [**ASC** | **DESC**] …]

Виды сортировок (по умолчанию ASC):

- **ASC** – по возрастанию
- **DESC** – по убыванию

Пример ORDER BY

```
mysql> SELECT id, title, author, price
    -> FROM books
    -> ORDER BY price DESC;
+----+---------------------------+--------------------+--------+
| id | title                     | author             | price  |
+----+---------------------------+--------------------+--------+
|  8 | Отцы и дети               | Иван Тургенев      | 371.00 |
|  7 | Анна Каренина             | Лев Толстой        | 346.00 |
|  6 | Война и мир               | Лев Толстой        | 341.00 |
|  3 | Мастер и Маргарита        | Михаил Булгаков    | 263.00 |
|  2 | Нос (Акция)               | Николай Гоголь     | 255.20 |
|  5 | Преступление и наказание  | Фёдор Достоевский  | 245.00 |
|  9 | Собачье сердце            | Михаил Булгаков    | 232.00 |
|  1 | Дубровский (Акция)        | Александр Пушкин   | 230.00 |
| 10 | Бесы                      | Фёдор Достоевский  | 212.00 |
|  4 | Мёртвые души (Акция)      | Николай Гоголь     | 173.00 |
+----+---------------------------+--------------------+--------+
10 rows in set (0.00 sec)
```

Вставка

INSERT INTO <имя>(<список столбцов>)

VALUES(<список значений>)

```sql
INSERT INTO books(title, author, publish_year, genre, price)
VALUES ('Дубровский', 'Александр Пушкин', 1855, 'Драма,Повесть', 125.50);
INSERT INTO books(title, author, publish_year, genre, price)
VALUES ('Нос','Николай Гоголь', 1836, 'Повесть', 150);
```

Обновление

UPDATE <имя>

SET <столбец> = <новое значение>[, <столбец> = <новое значение>]

[**WHERE** <условие>]

Пример update

```
mysql> SELECT id, title, author, price, discount, amount FROM books LIMIT 5;
+----+--------------------------+-------------------+--------+----------+--------+
| id | title                    | author            | price  | discount | amount |
+----+--------------------------+-------------------+--------+----------+--------+
|  1 | Дубровский               | Александр Пушкин  | 230.00 |        0 |      4 |
|  2 | Нос                      | Николай Гоголь    | 255.20 |        0 |      7 |
|  3 | Мастер и Маргарита       | Михаил Булгаков   | 240.50 |        0 |     10 |
|  4 | Мёртвые души             | Николай Гоголь    | 173.00 |        0 |      3 |
|  5 | Преступление и наказание | Фёдор Достоевский | 245.00 |        0 |      3 |
+----+--------------------------+-------------------+--------+----------+--------+
5 rows in set (0.00 sec)

mysql> UPDATE books SET price=263.00, discount=10, amount=amount-2 WHERE id=3;
```

```
mysql> SELECT id, title, author, price, discount, amount FROM books LIMIT 5;
+----+--------------------------+-------------------+--------+----------+--------+
| id | title                    | author            | price  | discount | amount |
+----+--------------------------+-------------------+--------+----------+--------+
|  1 | Дубровский               | Александр Пушкин  | 230.00 |        0 |      4 |
|  2 | Нос                      | Николай Гоголь    | 255.20 |        0 |      7 |
|  3 | Мастер и Маргарита       | Михаил Булгаков   | 263.00 |       10 |      8 |
|  4 | Мёртвые души             | Николай Гоголь    | 173.00 |        0 |      3 |
|  5 | Преступление и наказание | Фёдор Достоевский | 245.00 |        0 |      3 |
+----+--------------------------+-------------------+--------+----------+--------+
5 rows in set (0.00 sec)
```

Удаление

DELETE FROM <имя> [**WHERE** <условие>]

Пример delete

```
mysql> SELECT id, title, author, price, discount FROM books WHERE id BETWEEN 1 AND 5;
+----+----------------------+------------------------------+--------+----------+
| id | title                | author                       | price  | discount |
+----+----------------------+------------------------------+--------+----------+
|  1 | Капитанская дочка    | А.С.Пушкин                   | 151.20 |        0 |
|  2 | Мертвые души         | Н.В.Гоголь                   | 141.00 |        0 |
|  3 | Анна Каренина        | Л.Н.Толстой                  | 135.00 |       20 |
|  4 | Бесы                 | Ф.М.Достоевский              | 122.00 |        0 |
|  5 | Нос                  | Н.В.Гоголь                   | 105.00 |        0 |
+----+----------------------+------------------------------+--------+----------+
5 rows in set (0.00 sec)

mysql> DELETE FROM books WHERE author='Н.В.Гоголь';
Query OK, 2 rows affected (0.00 sec)

mysql> SELECT id, title, author, price, discount FROM books WHERE id BETWEEN 1 AND 5;
+----+----------------------+------------------------------+--------+----------+
| id | title                | author                       | price  | discount |
+----+----------------------+------------------------------+--------+----------+
|  1 | Капитанская дочка    | А.С.Пушкин                   | 151.20 |        0 |
|  3 | Анна Каренина        | Л.Н.Толстой                  | 135.00 |       20 |
|  4 | Бесы                 | Ф.М.Достоевский              | 122.00 |        0 |
+----+----------------------+------------------------------+--------+----------+
3 rows in set (0.00 sec)
```

GROUP BY

GROUP BY чаще всего используется совместно с собирательными функциями (COUNT(), MIN(), MAX(), SUM(), AVG()) для группировки результатов в одной или нескольких колонках

```sql
SELECT column1, column2, ... , columnN
FROM table_name
WHERE condition
GROUP BY column1, column2, ... , columnN
```

Групповые функции

Функция	Описание
SUM(Field)	Вычисляет сумму по указанному столбцу
MIN(Field)	Вычисляет минимальное значение в указанном столбце
MAX(Field)	Вычисляет максимальное значение в указанном столбце
AVG(Field)	Вычисляет среднее значение в указанном столбце
COUNT(*)	Вычисляет количество строк в результирующей выборке
COUNT(Field)	Вычисляет количество не пустых значений в столбце

Пример GROUP BY

```
mysql> SELECT id, title, author, price, amount FROM books;
+----+---------------------------+--------------------+--------+--------+
| id | title                     | author             | price  | amount |
+----+---------------------------+--------------------+--------+--------+
|  1 | Дубровский                | Александр Пушкин   | 230.00 |     20 |
|  2 | Нос                       | Николай Гоголь     | 255.20 |      7 |
|  3 | Мастер и Маргарита        | Михаил Булгаков    | 263.00 |      8 |
|  4 | Мёртвые души              | Николай Гоголь     | 230.00 |      3 |
|  5 | Преступление и наказание  | Фёдор Достоевский  | 230.00 |      3 |
|  6 | Война и мир               | Лев Толстой        | 346.00 |      1 |
|  7 | Анна Каренина             | Лев Толстой        | 346.00 |      0 |
|  8 | Отцы и дети               | Иван Тургенев      | 371.00 |      3 |
|  9 | Собачье сердце            | Михаил Булгаков    | 232.00 |      0 |
| 10 | Бесы                      | Фёдор Достоевский  | 212.00 |      8 |
+----+---------------------------+--------------------+--------+--------+
10 rows in set (0.00 sec)
```

```
mysql> SELECT author, price, COUNT(id) AS 'Число книг'
    -> FROM books
    -> GROUP BY author, price;
+--------------------+--------+------------+
| author             | price  | Число книг |
+--------------------+--------+------------+
| Александр Пушкин   | 230.00 |          1 |
| Иван Тургенев      | 371.00 |          1 |
| Лев Толстой        | 346.00 |          2 |
| Михаил Булгаков    | 232.00 |          1 |
| Михаил Булгаков    | 263.00 |          1 |
| Николай Гоголь     | 230.00 |          1 |
| Николай Гоголь     | 255.20 |          1 |
| Фёдор Достоевский  | 212.00 |          1 |
| Фёдор Достоевский  | 230.00 |          1 |
+--------------------+--------+------------+
9 rows in set (0.00 sec)
```

HAVING

HAVING используется вместо оператора WHERE, т.к. WHERE не может быть применен совместно с собирательными функциями (COUNT(), MIN(), MAX(), SUM(), AVG()).

```
1 SELECT column1, column2, ... , columnN
2 FROM table_name
3 WHERE condition
4 GROUP BY column1, column2, ... , columnN
5 HAVING condition
```

Пример HAVING

```
mysql> SELECT id, title, author, price, amount FROM books;
+----+--------------------------+--------------------+--------+--------+
| id | title                    | author             | price  | amount |
+----+--------------------------+--------------------+--------+--------+
|  1 | Дубровский               | Александр Пушкин   | 230.00 |     20 |
|  2 | Нос                      | Николай Гоголь     | 255.20 |      7 |
|  3 | Мастер и Маргарита       | Михаил Булгаков    | 263.00 |      8 |
|  4 | Мёртвые души             | Николай Гоголь     | 230.00 |      3 |
|  5 | Преступление и наказание | Фёдор Достоевский  | 230.00 |      3 |
|  6 | Война и мир              | Лев Толстой        | 346.00 |      1 |
|  7 | Анна Каренина            | Лев Толстой        | 346.00 |      0 |
|  8 | Отцы и дети              | Иван Тургенев      | 371.00 |      3 |
|  9 | Собачье сердце           | Михаил Булгаков    | 232.00 |      0 |
| 10 | Театральный роман        | Михаил Булгаков    | 212.00 |      8 |
+----+--------------------------+--------------------+--------+--------+
10 rows in set (0.00 sec)
```

```
mysql> SELECT COUNT(id) AS 'Число книг', author FROM books
    -> GROUP BY author HAVING COUNT(id) <= 1
+-------------+------------------------------+
| Число книг  | author                       |
+-------------+------------------------------+
|           1 | Александр Пушкин             |
|           1 | Иван Тургенев                |
|           1 | Фёдор Достоевский            |
+-------------+------------------------------+
1 row in set (0.00 sec)
```

UNION

Оператор **UNION** используется для объединения данных из двух и более таблиц, выводимых с помощью команды SELECT.

Для успешного объединения необходимо выполнить несколько условий:

- Количество колонок должно быть одинаково для всех объединяемых таблиц.
- Колонки должны иметь схожие (одинаковые) типы данных.
- Все колонки должны идти в одном и том же порядке.

```
1 mysql> SELECT column1, column2, ... , columnN FROM table_name1
2     -> UNION
3     -> SELECT column1, column2, ... , columnN FROM table_name2
```

Пример UNION

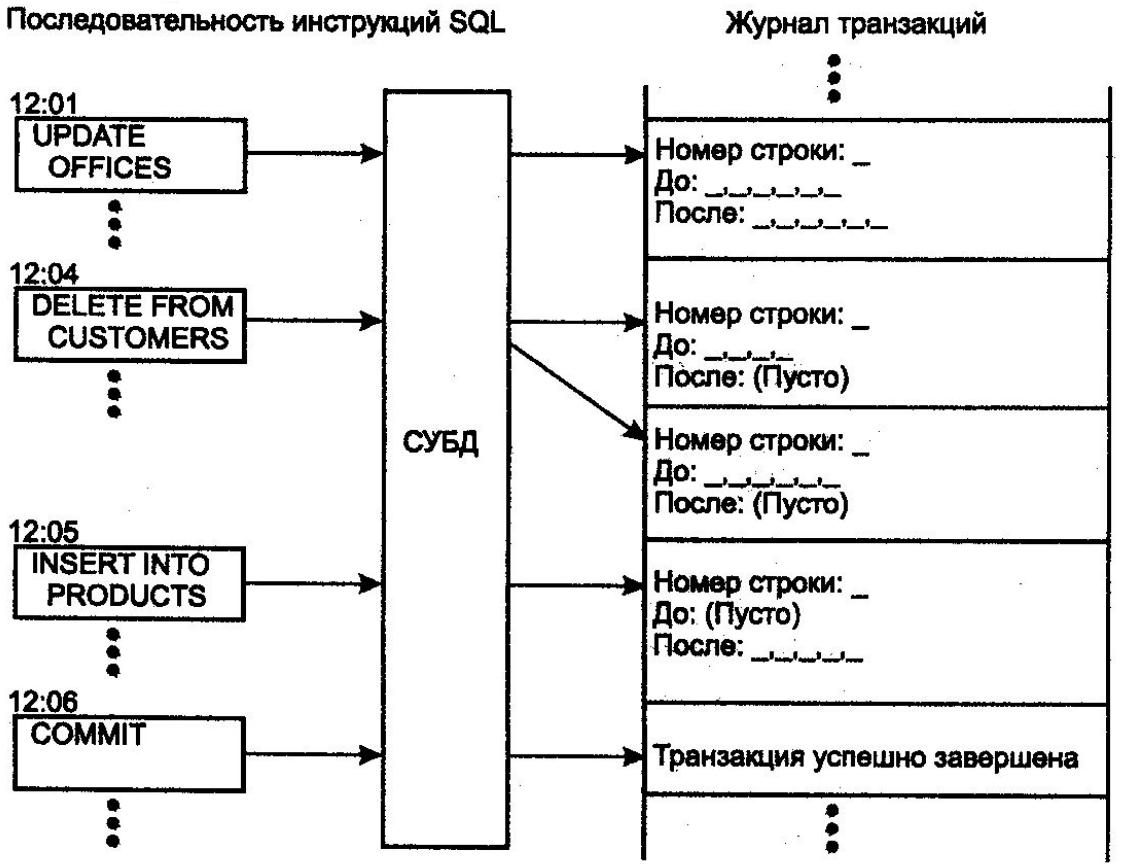
```
mysql> SELECT id, name, contract_expiration_date FROM suppliers;
+----+---------------------+--------------------------+
| id | name                | contract_expiration_date |
+----+---------------------+--------------------------+
|  1 | ООО "Книжный мир"   | 2021-02-23               |
|  2 | ООО "Все книги"     | 2025-01-01               |
|  3 | ООО "Книжный склад" | 2027-06-19               |
+----+---------------------+--------------------------+
3 rows in set (0.00 sec)

mysql> SELECT id, name, contract_expiration_date FROM carriers;
+----+-------------------------+--------------------------+
| id | name                    | contract_expiration_date |
+----+-------------------------+--------------------------+
|  1 | ООО "Скорость"          | 2020-12-31               |
|  2 | ООО "Грузовые перевозки" | 2021-05-05               |
+----+-------------------------+--------------------------+
2 rows in set (0.00 sec)
```

```
mysql> SELECT name, contract_expiration_date FROM suppliers
    -> UNION
    -> SELECT name, contract_expiration_date FROM carriers;
+-------------------------+--------------------------+
| name                    | contract_expiration_date |
+-------------------------+--------------------------+
| ООО "Книжный мир"       | 2021-02-23               |
| ООО "Все книги"         | 2025-01-01               |
| ООО "Книжный склад"     | 2027-06-19               |
| ООО "Скорость"          | 2020-12-31               |
| ООО "Грузовые перевозки" | 2021-05-05               |
+-------------------------+--------------------------+
5 rows in set (0.00 sec)
```

Транзакции

Транзакция — это операция, состоящая из одного или нескольких запросов к базе данных. Суть транзакций — обеспечить корректное выполнение всех запросов в рамках одной транзакции, а так-же обеспечить механизм изоляции транзакций друг от друга для решения проблемы совместного доступа к данным.

Любая транзакция либо выполняется полностью, либо не выполняется вообще.



Транзакции

В транзакционной модели есть два фундаментальных понятия: COMMIT и ROLLBACK.

- COMMIT означает фиксацию всех изменений в транзакции
- ROLLBACK означает отмену (откат) изменений, произошедших в транзакции

```
SET AUTOCOMMIT=0;
START TRANSACTION;
SELECT balance FROM checking WHERE customer_id = 10233276;
UPDATE checking SET balance = balance - 200.00 WHERE customer_id = 10233276;
UPDATE savings SET balance = balance + 200.00 WHERE customer_id = 10233276;
COMMIT;


CREATE PROCEDURE prc_test()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    ROLLBACK; //Вот здесь откатываем транзакцию в случае ошибки
  END;

  START TRANSACTION;
    INSERT INTO tmp_table VALUES ('null');
  COMMIT;
END;

CALL prc_test();
```

Нельзя откатиться

Для некоторых операторов нельзя выполнить откат с помощью ROLLBACK. Это операторы языка определения данных (Data Definition Language — DDL). Сюда входят запросы:

- CREATE
- ALTER
- DROP
- TRUNCATE
- COMMENT
- RENAME

Неявное завершение транзакции

Следующие операторы неявно завершают транзакцию (как если бы перед их выполнением был выдан COMMIT):

- ALTER TABLE
- DROP DATABASE
- LOAD MASTER DATA
- SET AUTOCOMMIT = 1
- BEGIN
- DROP INDEX
- LOCK TABLES
- START TRANSACTION
- CREATE INDEX
- DROP TABLE
- RENAME TABLE
- TRUNCATE TABLE

Уровни изоляции транзакций

- **0 — Чтение неподтверждённых данных (грязное чтение)** (Read Uncommitted, Dirty Read) — самый низкий уровень изоляции. При этом уровне возможно чтение незафиксированных изменений параллельных транзакций. Как раз в этом случае второй пользователь увидит вставленную запись из первой незафиксированной транзакции. Нет гарантии, что незафиксированная транзакция будет в любой момент откачена, поэтому такое чтение является потенциальным источником ошибок.
- **1 — Чтение подтверждённых данных** (Read Committed) — здесь возможно чтение данных только зафиксированных транзакций. Но на этом уровне существуют две проблемы. В этом режиме строки, которые участвуют в выборке в рамках транзакции, для других параллельных транзакций не блокируются, из этого вытекает проблема № 1: «Неповторяемое чтение» (non-repeatable read) — это ситуация, когда в рамках транзакции происходит несколько выборок (SELECT) по одним и тем же критериям, и между этими выборками совершается параллельная транзакция, которая изменяет данные, участвующие в этих выборках. Так как параллельная транзакция изменила данные, результат при следующей выборке по тем же критериям в первой транзакции будет другой. Проблема № 2 — «Фантомное чтение» — этот случай рассмотрен ниже.
- **2 — Повторяемое чтение** (Repeatable Read, Snapshot) — на этом уровне изоляции так же возможно чтение данных только зафиксированных транзакций. Так же на этом уровне отсутствует проблема «Неповторяемого чтения», то есть строки, которые участвуют в выборке в рамках транзакции, блокируются и не могут быть изменены другими параллельными транзакциями. Но таблицы целиком не блокируются. Из-за этого остается проблема «фантомного чтения». «Фантомное чтение» — это когда за время выполнения одной транзакции результат одних и тех же выборок может меняться по причине того, что блокируется не вся таблица, а только те строки, которые участвуют в выборке. Это означает, что параллельные транзакции могут вставлять строки в таблицу, в которой совершается выборка, поэтому два запроса SELECT * FROM table могут дать разный результат в разное время при вставке данных параллельными транзакциями.
- **3 — Сериализуемый** (Serializable) — сериализуемые транзакции. Самый надежный уровень изоляции транзакций, но и при этом самый медленный. На этом уровне вообще отсутствуют какие либо проблемы параллельных транзакций, но за это придется платить быстродействием системы, а быстродействие в большинстве случаев крайне важно.

Уровни изоляции транзакций

Уровень изоляции	Возможность чернового чтения	Возможность неповторяющегося чтения	Возможность фантомного чтения	Блокировка чтения
READ UNCOMMITED	Да	Да	Да	Нет
READ COMMITED	Нет	Да	Да	Нет
REPEATABLE READ	Нет	Нет	Да	Нет
SERIALIZABLE	Нет	Нет	Нет	Да