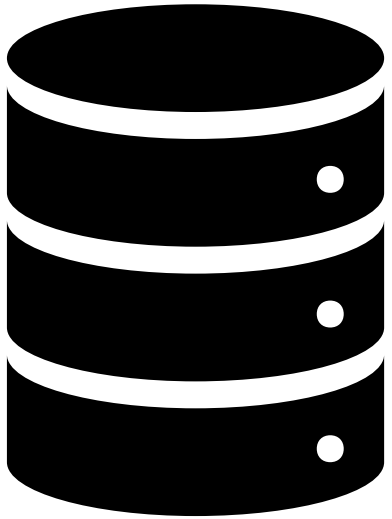


Базы данных

Лекция 3. СУБД. Транзакции. Троичная логика

Меркурьева Надежда
✉ merkurievanad@gmail.com
@merkurievanad

ФИВТ МФТИ, 2019

Комбинирование нескольких запросов

- UNION
- INTERSECT
- EXCEPT

```
Query_A
{UNION | INTERSECT | EXCEPT}
Query_B
```

UNION

- Позволяет объединять результаты двух и более запросов:
  - С одинаковым числом столбцов
  - Порядок столбцов в каждом запросе одинаков
  - Типы данных соответствующих столбцов совместимы
- Результатом является таблица, состоящая из объединения результатов запросов
- Если в Query_A есть строка X и в Query_B есть идентичная строка X, то в результате объединения останется только 1 строка X

- Если нужно, чтобы при объединении строка X была записана дважды (или более раз), можно использовать UNION ALL

UNION (ALL)

Результат выполнения Query_A

Фамилия	Имя
Роздухова	Нина
Житлухин	Дмитрий
Халяпов	Александр

Результат выполнения Query_B

Фамилия	Имя
Меркурьева	Надежда
Медведева	Анастасия
Житлухин	Дмитрий
Тюрюмина	Элла
Черняева	Надежда
Мазлов	Владимир
Халяпов	Александр

```
Query_A
UNION (ALL)
Query_B
```

Объединение результатов запросов

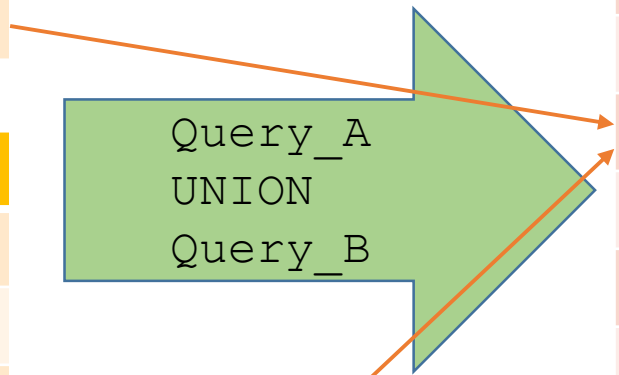
UNION

Результат выполнения Query_A

Фамилия	Имя
Роздухова	Нина
Житлухин	Дмитрий
Халяпов	Александр

Результат выполнения Query_B

Фамилия	Имя
Меркурьева	Надежда
Медведева	Анастасия
Житлухин	Дмитрий
Тюрюмина	Элла
Черняева	Надежда
Мазлов	Владимир
Халяпов	Александр



Объединение результатов запросов

Фамилия	Имя
Роздухова	Нина
Житлухин	Дмитрий
Халяпов	**Александр**
Меркурьева	Надежда
Медведева	Анастасия
Тюрюмина	Элла
Черняева	Надежда
Мазлов	Владимир

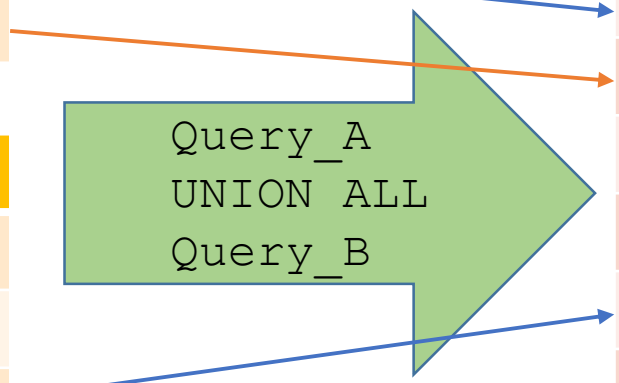
UNION ALL

Результат выполнения Query_A

Фамилия	Имя
Роздухова	Нина
Житлухин	Дмитрий
Халяпов	Александр

Результат выполнения Query_B

Фамилия	Имя
Меркурьева	Надежда
Медведева	Анастасия
Житлухин	Дмитрий
Тюрюмина	Элла
Черняева	Надежда
Мазлов	Владимир
Халяпов	Александр



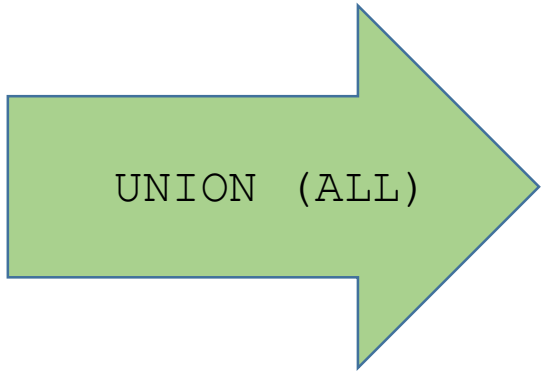
Объединение результатов запросов

Фамилия	Имя
Роздухова	Нина
Житлухин	Дмитрий
Халяпов	**Александр**
Меркурьева	Надежда
Медведева	Анастасия
Житлухин	Дмитрий
Тюрюмина	Элла
Черняева	Надежда
Мазлов	Владимир
Халяпов	**Александр**

UNION (ALL)

Фамилия
Иванов
Петров
Иванов
Сидоров

Фамилия
Иванов
Петров

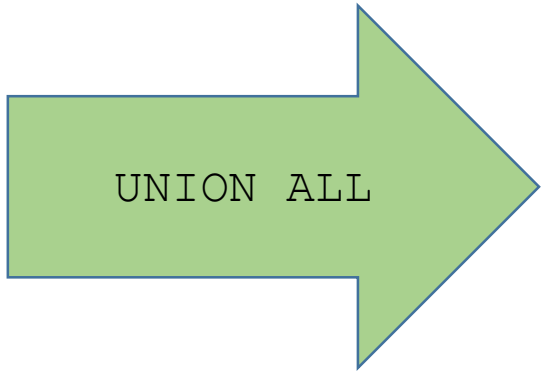


Объединение результатов запросов

UNION ALL

Фамилия
Иванов
Петров
Иванов
Сидоров

Фамилия
Иванов
Петров

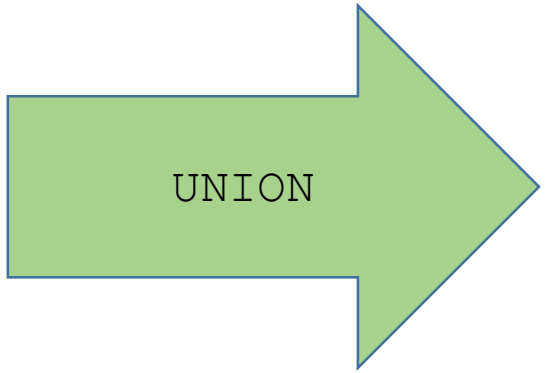


Фамилия
Иванов
Петров
Иванов
Сидоров
Иванов
Петров

UNION

Фамилия
Иванов
Петров
Иванов
Сидоров

Фамилия
Иванов
Петров



Фамилия
Иванов
Петров
Сидоров

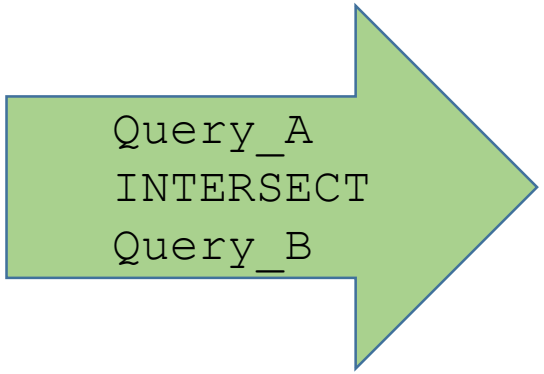
INTERSECT

Результат выполнения Query_A

Фамилия	Имя
Роздухова	Нина
Житлухин	Дмитрий
Халяпов	Александр

Результат выполнения Query_B

Фамилия	Имя
Меркурьева	Надежда
Медведева	Анастасия
Житлухин	Дмитрий
Тюрюмина	Элла
Черняева	Надежда
Мазлов	Владимир
Халяпов	Александр



Пересечение результатов запросов

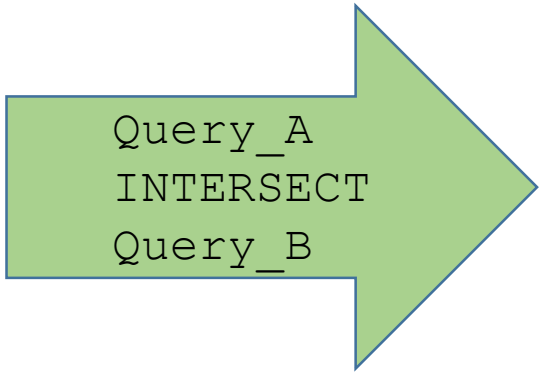
INTERSECT

Результат выполнения Query_A

Фамилия	Имя
Роздухова	Нина
Житлухин	Дмитрий
Халяпов	Александр

Результат выполнения Query_B

Фамилия	Имя
Меркурьева	Надежда
Медведева	Анастасия
Житлухин	Дмитрий
Тюрюмина	Элла
Черняева	Надежда
Мазлов	Владимир
Халяпов	Александр



Пересечение результатов запросов

Фамилия	Имя
Житлухин	Дмитрий
Халяпов	Александр

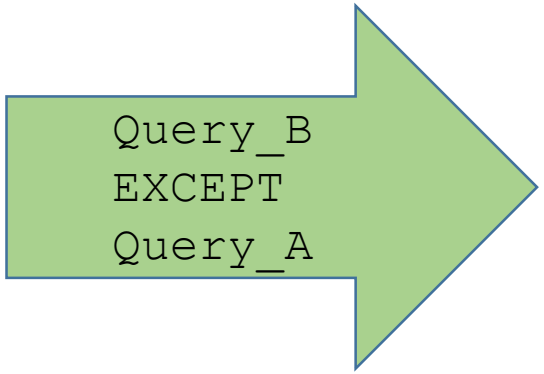
EXCEPT

Результат выполнения Query_A

Фамилия	Имя
Роздухова	Нина
Житлухин	Дмитрий
Халяпов	Александр

Результат выполнения Query_B

Фамилия	Имя
Меркурьева	Надежда
Медведева	Анастасия
Житлухин	Дмитрий
Тюрюмина	Элла
Черняева	Надежда
Мазлов	Владимир
Халяпов	Александр



Разность результатов запросов

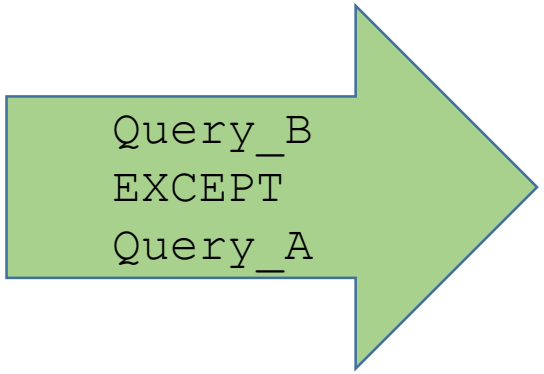
EXCEPT

Результат выполнения Query_A

Фамилия	Имя
Роздухова	Нина
Житлухин	Дмитрий
Халяпов	Александр

Результат выполнения Query_B

Фамилия	Имя
Меркурьева	Надежда
Медведева	Анастасия
Житлухин	Дмитрий
Тюрюмина	Элла
Черняева	Надежда
Мазлов	Владимир
Халяпов	Александр



Пересечение результатов запросов

Фамилия	Имя
Меркурьева	Надежда
Медведева	Анастасия
Тюрюмина	Элла
Черняева	Надежда
Мазлов	Владимир

SQL запрос: будет ли работать?

```
SELECT T1.attr1_1, …, T1.attr1_N, T2.attr2_1, …, T2.attr2_M
  FROM T1
 INNER|LEFT|RIGHT|FULL JOIN T2
    ON T1.attr1_1 = T2.attr2_1 AND T1.attr1_N > T2.attr2_M
 WHERE (T1.attr1_k BETWEEN X AND Y) OR (T2.attr2_l LIKE
'a%')
 GROUP BY T1.attr1_2, T2.attr2_2
HAVING SUM(T1.attr1_3) > 0
 UNION
SELECT T3.attr3_1, …, T3.attr3_N, T4.attr4_1, …, T4.attr4_M
  FROM T3
 INNER|LEFT|RIGHT|FULL JOIN T4
    ON T3.attr3_1 > T4.attr4_2;
```

Связь реляционной алгебры с SQL

Реляционная алгебра	SQL
Теоретико-множественные операции	
Объединение	UNION
Пересечение	INTERSECT
Разность	EXCEPT
Реляционные операции	
Ограничение	WHERE
Проекция	SELECT
Соединение	JOIN
Деление	-

Примеры запросов

Реляционная алгебра	SQL
Из таблицы Product(product_id, product_name, product_type, product_price) выделить все наименования продуктов с ценой не менее 150 рублей	
P150 = Product[product_price >= 150] R = P150[product_name] Ответ: R.	**SELECT DISTINCT** product_name **FROM** Product **WHERE** product_price >= 150;
Пусть даны таблицы Product(product_id, product_name, product_type, product_price) и Shop(shop_id, shop_name, product_id). Найти все магазины, в которых продается мыло с ценой менее 150 рублей	
PS = Product[product_type = 'Мыло'][product_price < 150] SS = PS[PS[product_id] = Shop[product_id]]Shop SSN = SS[shop_name] Ответ: SSN.	**SELECT DISTINCT** shop_name **FROM** Product P **INNER JOIN** Shop Sh **ON** P.product_id = Sh.product_id **WHERE** product_type = 'Мыло' **AND** product_price < 150;

Система управления базами данных

- совокупность программных и лингвистических средств, обеспечивающих управление созданием и использованием баз данных

Преимущества

- Совместный доступ
  - Экономия на обслуживании
- Непротиворечивость
  - Единая версия истины
- Контроль целостности
  - Соответствие данных внутренней логике системы
- Транзакционность
  - Надежный механизм обновления
- Независимость от данных
  - Упрощение сопровождения

Основные существующие СУБД

- PostgreSQL
  - Open source and free
  - Наиболее соответствует стандарту SQL
- Oracle
  - Дорого-богато
  - Первыми создали коммерческое решение (1979 г.)
  - Поддерживает много языков программирования
- MS SQL
  - Оригинальная идея получена от Sysbase
  - Хорошая интеграция с Microsoft решениями
  - Разнообразие процедурных расширений
- MySQL
  - Тоже open source
  - Тоже производится Oracle

PostgreSQL

- Open source решение
- Разработан в University of California, Berkley
- Произошел от Ingres (Post Ingres)
- Написан на C
- Поддерживает много возможностей из SQL: 2011
- Является базой по умолчанию в MacOS Server
- Используется в таких компаниях как Yandex, Reddit, Skype, Instagram, TripAdviser, The Guardian

Oracle DB

- Имеет много дополнительных возможностей
- Придерживается стандарта слабее, чем PostgreSQL
- Поддерживает наибольшее число языков программирования
- Является наиболее широко распространенным коммерческим решением

MySQL

- Open source решение
- Поддерживает несколько движков СУБД
- Сейчас находится под Oracle
- Имеет несколько «копий» MariaDB, Percona и т.д., которые появились после покупки Oracle
- До определенного момента по функционалу сильно проигрывала PostgreSQL
- Используется, например, в Google, Facebook, Twitter, YouTube
- Входит в LAMP – стэк приложений для веб-сервисов (Linux, Apache HTTP Server, MySQL, PHP)

Процедурные расширения SQL

- PL/SQL — Oracle
- Transact SQL (T-SQL) — MS SQL
- PL/pgSQL — PostgreSQL

Транзакция

Транзакция – это группа последовательных операций с базой данных, которая представляет собой логическую единицу работы с данными, гарантированно переводящая БД из одного непротиворечивого состояния в другое.

TCL (Transaction Control Language)

- `COMMIT`
  - Применяет транзакцию, т.е. сохраняет изменения, произведенные в процессе выполнения транзакции
- `ROLLBACK`
  - Откатывает все изменения, произведенные в процессе выполнения транзакции
- `SAVEPOINT`
  - Создает так называемую точку останова

Точка останова

- Точка останова – промежуточный участок в транзакции, на который можно откатиться в случае необходимости
  - Позволяет дробить транзакцию на части
  - Позволяет реализовать «вложенные» транзакции

Свойства транзакции (ACID)

- **A**tomicity (Атомарность):
  - Выполнены либо все подоперации, либо никакие
- **C**onsistency (Согласованность)
  - Каждая успешная транзакция фиксирует только допустимые результаты
- **I**solation (Изолированность)
  - Параллельные транзакции не влияют на результаты друг друга
- **D**urability (Устойчивость)
  - Вне зависимости от сбоев системы результаты успешных транзакций сохранятся в системе

Atomicity (атомарность)

- Никакая транзакция не будет зафиксирована в системе частично
- Выполнены либо все подоперации, либо никакая

- На практике одновременное и атомарное выполнение транзакций невозможно
- На практике «атомарность» реализуется с использованием «отката» (rollback)
- В процессе отката отменяются все уже произведенные операции

Consistency (согласованность)

- Фиксируются только допустимые результаты операций
- Согласованность применяется не только в контексте БД.

Пример: банковские транзакции

- В ходе выполнения операции согласованность не требуется
- Вследствие атомарности промежуточная несогласованность остается скрытой

Isolation (изолированность)

- Параллельные транзакции не оказывают влияния на друг друга

- В реальных БД полная изолированность не поддерживается
- Уровень изолированности – характеристика соответствия БД свойству изолированности

Проблемы поддержки изолированности

- Потерянное обновление:
  - Изменение одного блока данных несколькими транзакциями
- «Грязное» чтение:
  - Чтение данных, измененных впоследствии откатившейся транзакцией
- Неповторяющееся чтение
  - Повторное чтение измененных данных одной и той же транзакцией
- Чтение «фантомов»
  - Взаимосвязанные критерии изменения данных двумя транзакциями

Изолированность: потерянное обновление

Транзакция 1	Транзакция 2

`UPDATE table_1`
`  SET attr_2 = attr_2 + 20`
`WHERE attr_1 = 1;`
	`UPDATE table_1`
	`  SET attr_2 = attr_2 + 25`
	`WHERE attr_1 = 1;`

Что получим в итоге?

1. `attr_2 = attr_2 + 20`
2. `attr_2 = attr_2 + 25`
3. `attr_2 = attr_2 + 45`

Изолированность: потерянное обновление

Транзакция 1	Транзакция 2

`UPDATE table_1`
`SET attr_2 = attr_2 + 20`
`WHERE attr_1 = 1;`
	`UPDATE table_1`
	`SET attr_2 = attr_2 + 25`
	`WHERE attr_1 = 1;`

Что получим в итоге?

1. `attr_2 = attr_2 + 20`
2. `attr_2 = attr_2 + 25`
3. `attr_2 = attr_2 + 45`

Результат однозначно не определен

Изолированность: «грязное» чтение

Транзакция 1	Транзакция 2

`UPDATE table_1`
`  SET attr_2 = attr_2 + 20`
`WHERE attr_1 = 1;`	

	`SELECT attr_2`
	`  FROM table_1`
	`WHERE attr_1 = 1;`

`ROLLBACK WORK;`	

Изолированность: «грязное» чтение

Транзакция 1	Транзакция 2

`UPDATE table_1`
`  SET attr_2 = attr_2 + 20`
`WHERE attr_1 = 1;`	

	`SELECT attr_2`
	`  FROM table_1`
	`WHERE attr_1 = 1;`

`ROLLBACK WORK;`	

Значения, полученные второй транзакцией, будут отличаться от значений, хранимых в БД

Изолированность: неповторяющееся чтение

Транзакция 1	Транзакция 2

	`SELECT attr_2`
	`  FROM table_1`
	`WHERE attr_1 = 1;`

`UPDATE table_1`
`  SET attr_2 = attr_2 + 20`
`WHERE attr_1 = 1;`	

`COMMIT;`	

	`SELECT attr_2`
	`  FROM table_1`
	`WHERE attr_1 = 1;`

Изолированность: чтение фантомов

Транзакция 1	Транзакция 2
	`SELECT sum(attr_2) FROM table_1;`
`INSERT INTO table_1 (attr_1, attr_2) VALUES (15, 20);`	
`COMMIT;`	
	`SELECT sum(attr_2) FROM table_1;`

Уровни изолированности транзакций

- Read uncommitted (чтение незафиксированных данных)
- Read committed (чтение фиксированных данных)
- Repeatable read (повторяемость чтения)
- Serializable (упорядочиваемость)

Read uncommitted

- Первый уровень изоляции
- Гарантирует отсутствие потерянных обновлений
- Итоговое значение – результат выполнения каждой транзакции
- Возможно считывание незафиксированных изменений

- Данные блокируются на время внесения изменений
- На время чтения данных блокировка отсутствует

Read committed

- Второй уровень изоляции
- Используется в большей части СУБД
- Защита от «грязного» чтения
- В процессе выполнения одна из транзакций успешно завершается, тогда остальные работают с измененными данными

- Реализация IRL на усмотрение разработчиков СУБД:
  - Блокирование читаемых и изменяемых данных
  - Сохранение нескольких версий параллельно изменяемых строк

Read committed: блокировка

- Блокирование читаемых и изменяемых данных
- *Пишущая* транзакция блокирует данные для *читающих* транзакций уровня read commited или выше
- Отсутствует «грязное» чтение
- Возможно неповторяющееся чтение

Read committed: версионность

- Сохранение нескольких версий параллельно изменяемых строк
- СУБД создает новую версию строки, с которой работает «изменяющие данные» транзакция
- Для «читающей» транзакции доступна последняя **зафиксированная** версия

Repeatable read

- Третий уровень изоляции
- Читающая транзакция игнорирует изменения в данных, которые были ей ранее прочитаны
- Никакая транзакция не может изменять данные, читаемые текущей транзакцией, пока чтение не завершено
- Спасает от эффекта неповторяющегося чтения

Serializable

- Четвертый (самый высокий) уровень изоляции
- Транзакции **полностью** изолированы друг от друга
- Параллельных транзакций как будто бы не существует вовсе
- Транзакции не подвержены эффекту «фантомного чтения»

Уровни изоляции

	Read uncommitted	Read committed	Repeatable read	Serializable
Потерянное обновление	+	+	+	+
«Грязное» чтение	-	+	+	+
Неповторяющееся чтение	-	-	+	+
Фантомное чтение	-	-	-	+

Durability (устойчивость)

- Вне зависимости от сбоев системы результаты успешных транзакций сохранятся в системе
- Если пользователь получил подтверждение об успешности транзакции, гарантируется сохранность результатов

Свойства транзакции (ACID)

- **A**tomicity (Атомарность):
  - Выполнены либо все подоперации, либо никакие
- **C**onsistency (Согласованность)
  - Каждая успешная транзакция фиксирует только допустимые результаты
- **I**solation (Изолированность)
  - Параллельные транзакции не влияют на результаты друг друга
- **D**urability (Устойчивость)
  - Вне зависимости от сбоев системы результаты успешных транзакций сохранятся в системе

Синтаксис

- SET TRANSACTION transaction_mode [, ...]
  - ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED }
  - READ WRITE | READ ONLY
  - [NOT] DEFERRABLE
- BEGIN
  - COMMIT
  - ROLLBACK
- SAVEPOINT name
  - ROLLBACK TO SAVEPOINT name
  - RELEASE SAVEPOINT name

Пример использования

```
BEGIN;
    INSERT INTO table1 VALUES (1);
    SAVEPOINT my_savepoint;
    INSERT INTO table1 VALUES (2);
    ROLLBACK TO SAVEPOINT my_savepoint;
    INSERT INTO table1 VALUES (3);
COMMIT;
```

Бинарная логика

- Допустимые значения логических выражений:
  - `TRUE`
  - `FALSE`

Бинарная логика

AND	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

OR	TRUE	FALSE
TRUE	TRUE	TRUE
FALSE	TRUE	FALSE

	NOT
TRUE	FALSE
FALSE	TRUE

Бинарная логика

- Допустимые значения логических выражений:
  - `TRUE`
  - `FALSE`

- Очень нужно абстрагироваться, иначе будет грустно и сложно

Тернарная (троичная) логика

- Что делать, если значение в ячейке неизвестно?
- Как проверить, что значение действительно неизвестно?
- Как учитывать в расчетах такие ячейки?
- Как правильно их агрегировать?

Тернарная (троичная) логика

- Допустимые значения логического выражения:
  - `TRUE`
  - `FALSE`
  - `UNKNOWN`

- Как оперировать со значениями только из булевой логики
- Как оперировать со значениями типа `UNKNOWN`
- В SQL используется обозначение `NULL`

Сравнение с NULL

- `NULL = 1`
- `NULL <> 1`
- `NULL > 1`
- `NULL = NULL`

Сравнение с NULL

- `NULL = 1` NULL
- `NULL <> 1` NULL
- `NULL > 1` NULL
- `NULL = NULL` NULL

NULL в выражениях с AND/OR

- (NULL = 1) OR (1 = 0)
- (NULL = 1) OR (1 = 1)
- (NULL = 1) AND (1 = 0)
- (NULL = 1) AND (1 = 1)
- NOT (NULL = NULL)

NULL в выражениях с AND/OR

- NULL OR FALSE
- NULL OR TRUE
- NULL AND FALSE
- NULL AND TRUE
- NOT NULL

NULL в выражениях с AND/OR

- X AND TRUE
- X AND FALSE
- X OR TRUE
- X OR FALSE

NULL в выражениях с AND/OR

- X AND TRUE — X
- X AND FALSE — FALSE
- X OR TRUE — TRUE
- X OR FALSE — X

NULL в выражениях с AND/OR

- `NULL OR FALSE` — `NULL`
- `NULL OR TRUE` — `TRUE`
- `NULL AND FALSE` — `FALSE`
- `NULL AND TRUE` — `NULL`
- `NOT NULL` — `NULL`

- Иными словами, если `NULL` как-то влияет на значение логического выражения, результат не определен

NULL в запросах

- Ключевые слова
  - WHERE
  - HAVING

строго требуют значение TRUE, поэтому условие NOT FALSE не является достаточным.

NULL в запросах

```
SELECT col
FROM T
WHERE col = NULL;
```

NULL в запросах

```
SELECT col
  FROM T
 WHERE col = NULL;
```

По такому запросу не будет отобрана ни одна запись

NULL в запросах

```
SELECT col
  FROM T
 WHERE col = NULL
    OR NOT (col = NULL);
```

NULL в запросах

```
SELECT col
  FROM T
 WHERE col = NULL
    OR NOT (col = NULL);
```

По такому запросу тоже не будет отобрана ни одна запись

NULL в запросах

- Предикат `IS NULL` позволяет отбирать строки со значениями `NULL`

```
SELECT col
  FROM T
 WHERE col IS NULL;
```

NULL в запросах

- `1 NOT IN (0)`
- `1 NOT IN (1)`

NULL в запросах

- `1 NOT IN (0)`
- `1 NOT IN (1)`

- `1 NOT IN (NULL)`
- `1 NOT IN (2, NULL)`
- `1 NOT IN (NULL, 1)`

NULL в запросах

- `1 NOT IN (0)` — TRUE
- `1 NOT IN (1)` — FALSE

- `1 NOT IN (NULL)` — NULL
- `1 NOT IN (2, NULL)` — NULL
- `1 NOT IN (NULL, 1)` — FALSE

NULL в запросах

- Для теоретико-множественных операций
  - UNION
  - INTERSECT
  - EXCEPT

NULL = NULL

- НО! Для операции UNION ALL NULL <> NULL

NULL в запросах: IS DISTINCT FROM

- IS DISTINCT FROM
  - Аналогично операции ‘<>’
  - Относится к NULL как к известному значению
  - Oracle и MS SQL не умеют ☹
  - Зато PostgreSQL умеет ☺

NULL в запросах: IS DISTINCT FROM

attr1	attr2
1	1
1	2
1	NULL
2	1
2	2
2	NULL
NULL	1
NULL	2
NULL	NULL

```
SELECT attr1,
       attr2
  FROM table1
 WHERE attr1 IS DISCTINCT FROM attr2;
```

NULL в запросах: IS DISTINCT FROM

attr1	attr2	IS DISTINCT FROM
1	1	FALSE
1	2	TRUE
1	NULL	TRUE
2	1	TRUE
2	2	FALSE
2	NULL	TRUE
NULL	1	TRUE
NULL	2	TRUE
NULL	NULL	FALSE

NULL в запросах: IS DISTINCT FROM

attr1	attr2	IS DISTINCT FROM
1	1	FALSE
1	2	TRUE
1	NULL	TRUE
2	1	TRUE
2	2	FALSE
2	NULL	TRUE
NULL	1	TRUE
NULL	2	TRUE
NULL	NULL	FALSE

NULL в запросах: COALESCE

- `COALESCE(attr1, attr2,… , attrn )`
  - Функция возвращает первое отличное от NULL значение
  - Число аргументов ограничивается только вашей фантазией

NULL в запросах: COALESCE

attr1	attr2
1	1
1	2
1	NULL
2	1
2	2
2	NULL
NULL	1
NULL	2
NULL	NULL

```
SELECT coalesce(attr1, attr2) AS attr
  FROM table
 WHERE attr1 IS DISCTINCT FROM attr2;
```

NULL в запросах: COALESCE

attr1	attr2
1	1
1	2
1	NULL
2	1
2	2
2	NULL
NULL	1
NULL	2
NULL	NULL

attr
1
1
1
2
2
2
1
2
NULL

NULL в запросах: агрегирующие функции

- При подсчетах в агрегирующих функциях NULL не учитывается

```
SELECT sum(attr),
       min(attr),
       max(attr),
       avg(attr),
       count(attr)
  FROM table;
```

NULL в запросах: агрегирующие функции

attr
1
NULL
3
10
2
15
NULL
28
NULL
11
80

```
SELECT sum(attr),
       min(attr),
       max(attr),
       avg(attr),
       count(attr)
  FROM table;
```

NULL в запросах: агрегирующие функции

attr
1
NULL
3
10
2
15
NULL
28
NULL
11
80

sum(attr)	min(attr)	max(attr)	avg(attr)	count(attr)
150	1	80	18.75	8

F571: Boolean test

- Расширяет оператор `IS` на все значения логического типа данных: `TRUE, FALSE, NULL;`

- SQL: 1999
- Поддерживается не всеми СУБД, есть в:
  - PostgreSQL
  - MySQL

Резюме

- При написании запросов нужно помнить:
  - В какой СУБД вы работаете и на каком диалекте SQL пишете
  - В каких полях вашей таблицы допустимо присутствие NULL
  - Как именно работает троичная логика в разных операциях
  - Корректно ли отработает ваш запрос, если где-то встретится NULL

- Как написать хороший запрос:
  - Сначала подумать
  - Потом написать