МИНОБРНАУКИ РОССИИ

Передовые инженерные школы

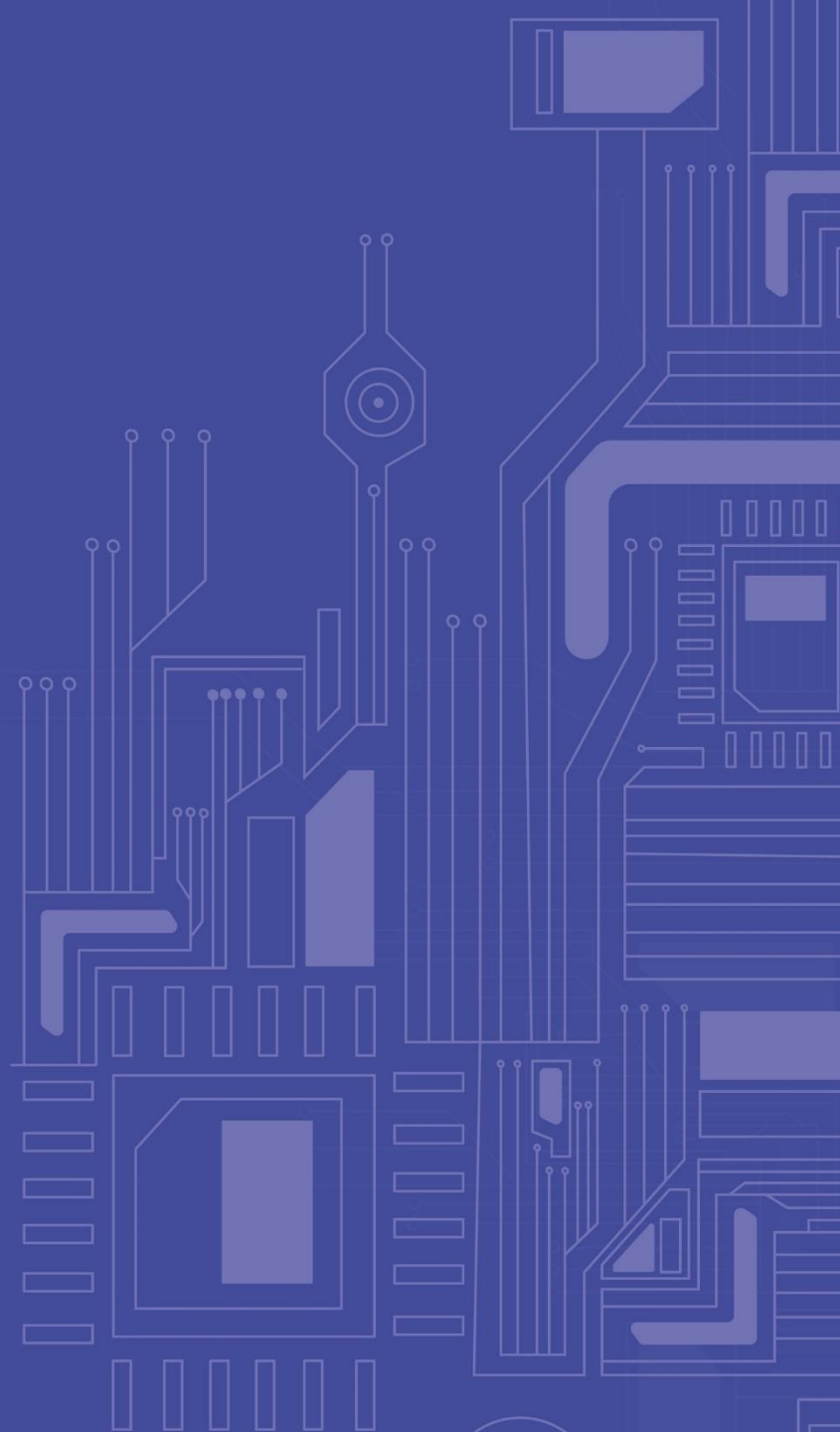
СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Лекция 9

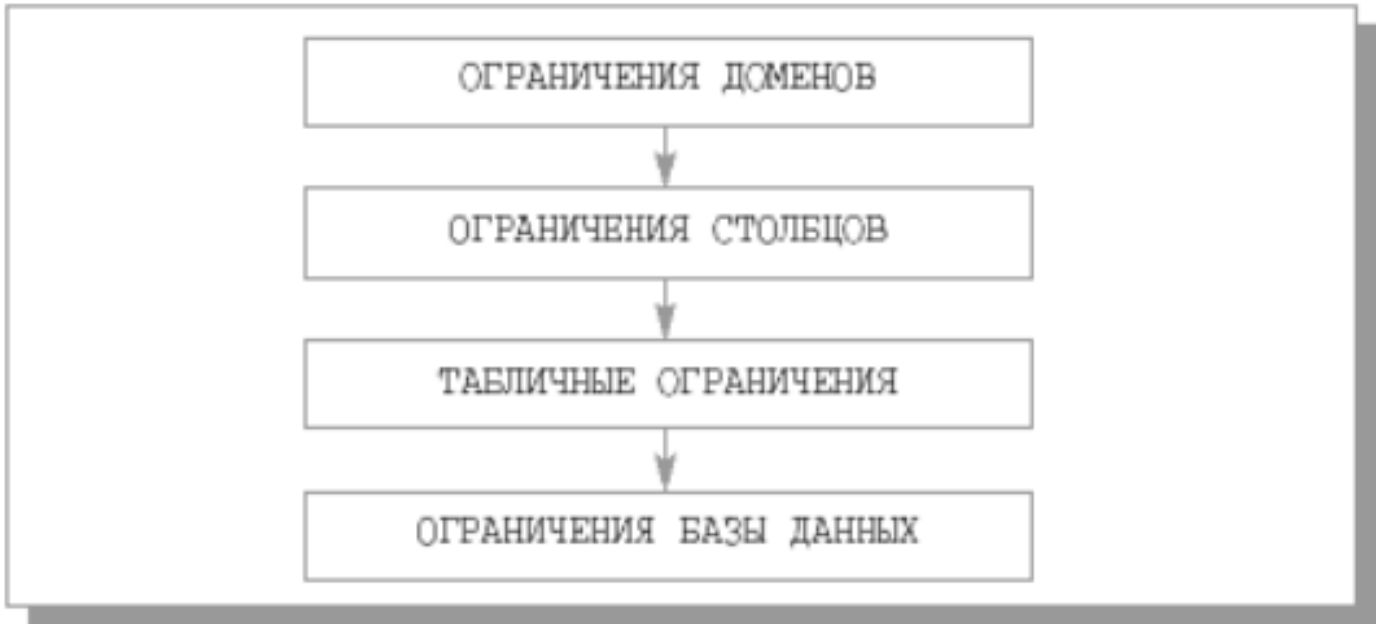
ПЛАН ЛЕКЦИИ

- Общие ограничения целостности
- Скалярные выражения
- Общая структура оператора выборки
- Ссылки на таблицы раздела FROM
- Примеры

ОБЩИЕ ОГРАНИЧЕНИЯ ЦЕЛОСТНОСТИ



ВИДЫ ОГРАНИЧЕНИЙ ЦЕЛОСТНОСТИ



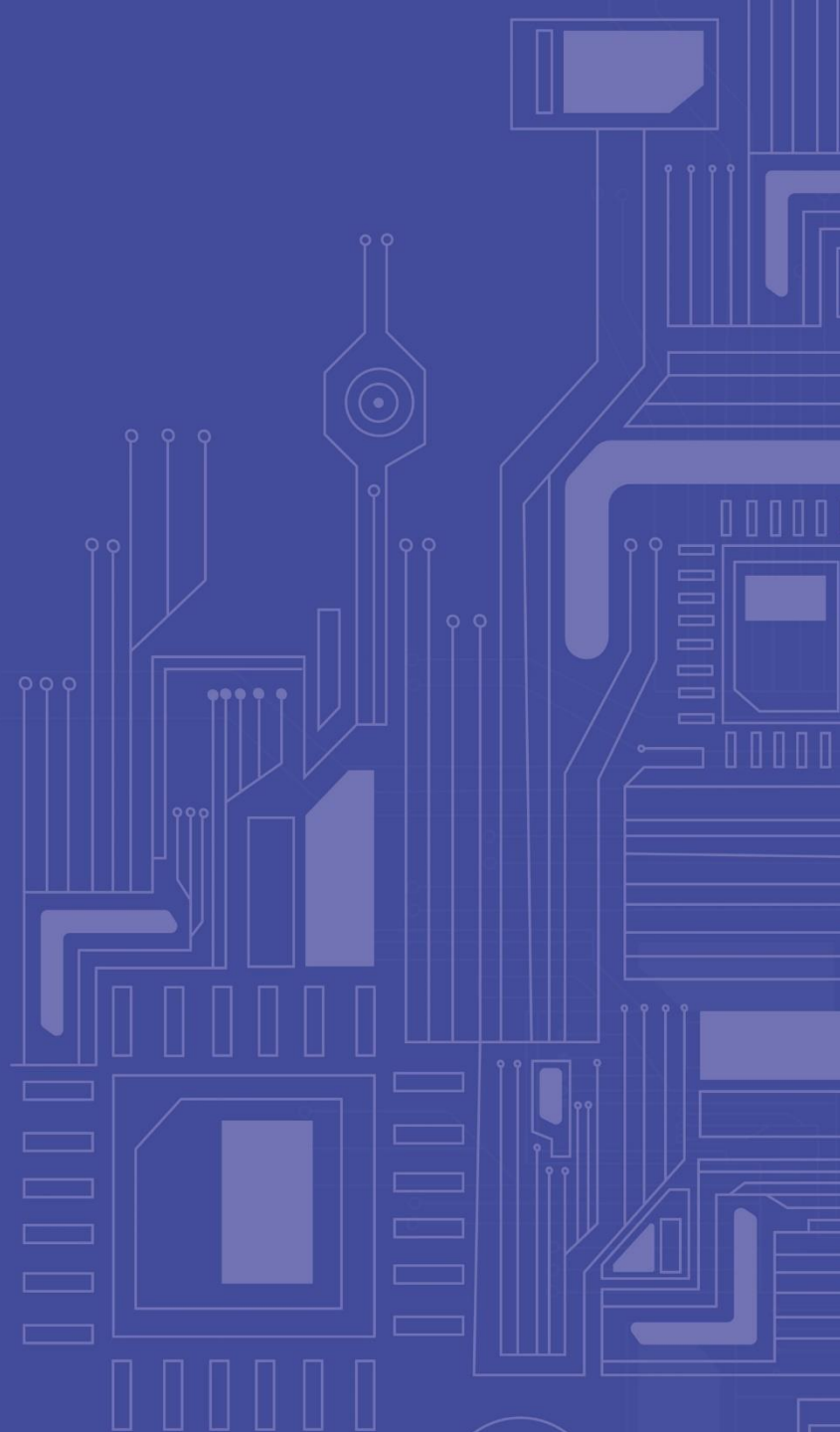
ОПРЕДЕЛЕНИЕ ОБЩИХ ОГРАНИЧЕНИЙ ЦЕЛОСТНОСТИ (1/2)

1. Оператор, синтаксис
2. Ограничения столбцов и таблиц
3. Вычисляемые и логические ограничения
4. Ограничения внешнего ключа

ОПРЕДЕЛЕНИЕ ОБЩИХ ОГРАНИЧЕНИЙ ЦЕЛОСТНОСТИ (2/2)

1. Отмена определения общего ограничения целостности
2. Немедленная и откладываемая поверка ограничений

СКАЛЯРНЫЕ ВЫРАЖЕНИЯ



ОБЩИЕ СИНТАКСИЧЕСКИЕ ПРАВИЛА

```
value_expression ::=
    numeric_value_expression
  | string_value_expression
  | datetime_value_expression
  | interval_value_expression
  | boolean_value_expression
  | array_value_expression
  | multiset_value_expression
  | row_value_expression
  | user_defined_value_expression
  | reference_value_expression
```

```
value_expression_primary ::=
    unsigned_value_specification
  | column_reference
  | set_function_specification
  | scalar_subquery
  | case_expression
  | (value_expression)
  | cast_specification
```

ЧИСЛЕННЫЕ ВЫРАЖЕНИЯ

```
numeric_value_expression> ::= numeric_term
   | numeric_value_expression + term
   | numeric_value_expression – term
numeric_term ::= numeric_factor
   | numeric_term * numeric_factor
   | numeric_term / numeric_factor
numeric_factor ::= [ { + | – } ] numeric_primary
numeric_primary ::= value_expression_primary
   | numeric_value_function
```

```
numeric_value_function ::=
   POSITION (character_value_expression
   IN character_value_expression)
   |{CHAR_LENGTH | CHARACTER_LENGTH }
   (string_value_expression)
   | OCTET_LENGTH (string_value_expression)
   | BIT_LENGTH (string_value_expression)
   | EXTRACT ({ datetime_field | time_zone field }
     FROM { datetime_value_expression
       | interval_value_expression })
   | CARDINALITY (array_value_expression
         | multiset_value_expression)
   | ABS (numeric_value_expression)
   | MOD (numeric_value_expression)
```

СИМВОЛЬНЫЕ ИЛИ БИТОВЫЕ СТРОКИ

1. Конкатенация

```
string_value_function ::= character_value_function
   | bit_value_function
character _value_function ::= SUBSTRING
   (character _value_expression
     FROM start_position
       [ FOR string_length ])
     | SUBSTRING (character _value_expression
       SIMILAR character _value_expression
         ESCAPE character_value_expression)
     | { UPPER | LOWER }
       (character_value_expression)
     | CONVERT (character_value_expression
       USING conversion_name)
     | TRANSLATE (character_value_expression)
       USING translation_name)
     | TRIM ([ {LEADING | TRAILING | BOTH} ]
       [ character_value_expression ]
       [ character_value_expression ])
     | OVERLAY (character_value_expression
       PLACING character_value_expression
         FROM start_position
           [ FOR string_length ])
bit _value_function ::= SUBSTRING (bit_value_expression
       FROM start_position
         [ FOR string_length ])
start_position ::= numeric_value_expression
string length ::= numeric_value_expression
```

ВЫРАЖЕНИЯ ДАТЫ - ВРЕМЕНИ

```
datetime_value_expression ::=
   datetime_term
     | interval_value_expression + datetime term
     | datetime_value_expression + interval term
     | datetime value expression – interval term
datetime_term ::= datetime_primary
   [ AT { LOCAL | TIME ZONE interval_value_expression } ]
datetime_primary ::= value_expression_primary
     | datetime_value_function
```

```
datetime_value_function ::= CURRENT_DATE
   | CURRENT_TIME [ (precision) ]
   | LOCALTIME [ (precision) ]
   | CURRENT_TIMESTAMP [ (precision) ]
   | LOCALTIMESTAMP [ (precision) ]
```

```
interval_value_expression ::=
     interval_term
   | interval_value_expression + interval term
   | interval_value_expression – interval term
   | (datetime value expression – datetime term)
     interval_qualifier
interval_term ::= interval_factor
   | interval_term * numeric_factor
   | interval_term / numeric_factor
   | numeric_term * interval_factor

interval_factor ::= [ { + | – } ]
     interval_primary [ <interval qualifier> ]
interval_primary ::= value_expression_primary
   | interval_value_function
```

БУЛЕВСКИЕ ВЫРАЖЕНИЯ

```
boolean_value_expression ::= boolean_term
   | boolean_value_expression OR boolean_term
boolean_term ::= boolean_factor
   | boolean_term AND boolean_factor
boolean_factor ::= [ NOT ] boolean_test
boolean_test ::= boolean_primary [ IS [ NOT ] truth_value ]
truth_value ::= TRUE | FALSE | UNKNOWN
boolean_primary ::= predicate
   | (boolean_value_expression)
   | value_expression_primary
```

1. Приоритет операций
2. Таблицы истинности IS и IS NOT

IS	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	FALSE
FALSE	FALSE	TRUE	FALSE
UNKNOWN	FALSE	FALSE	TRUE

IS NOT	TRUE	FALSE	UNKNOWN
TRUE	FALSE	TRUE	TRUE
FALSE	TRUE	FALSE	TRUE
UNKNOWN	TRUE	TRUE	FALSE

ВЫРАЖЕНИЯ С ПЕРКЛЮЧАТЕЛЕМ (1/2)

1. Синтаксис
2. Эквивалентность выражений с простым и поисковым переключателем

```
case_expression ::= case_abbreviation
   | case_specification

case_abbreviation ::= NULLIF (value_expression , value_expression)
   | COALESCE (value_expression_comma_list)
case specification ::= simple_case | searched_case

simple_case ::= CASE value_expression simple_when_clause_list
     [ ELSE value_expression ] END

searched_case ::= CASE searched_when_clause_list
     [ ELSE value_expression ] END
simple_when_clause ::= WHEN value_expression
lue_expression
n_clause ::= WHEN conditional_expression
lue_expression
```

```
CASE CO WHEN WO1 THEN result1
   WHEN WO2 THEN result2
   . . . . . .
   WHEN WOn THEN resultn
   ELSE result
END
```

эквивалентно выражению с поисковым переключателем

```
CASE WHEN CO = WO1 THEN result1
   WHEN CO = WO2 THEN result2
   . . . . . .
   WHEN CO = WOn THEN resultn
   ELSE result
END
```

ВЫРАЖЕНИЯ С ПЕРКЛЮЧАТЕЛЕМ (2/2)

Выражение `NULLIF (V1, V2)` эквивалентно следующему выражению с переключателем:

```
CASE WHEN V1 = V2 THEN NULL ELSE V1 END.
```

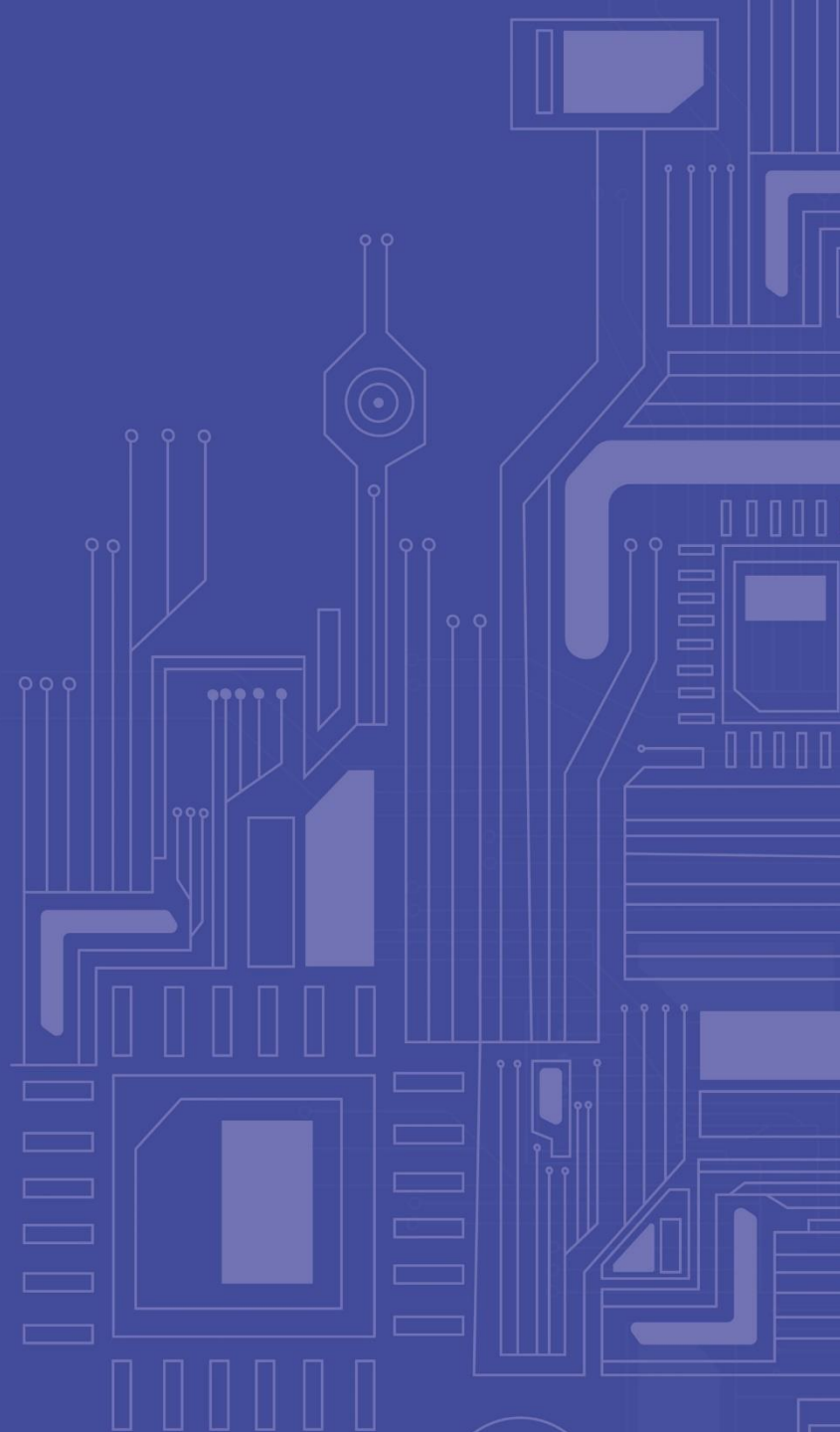
Выражение `COALESCE (V1, V2)` эквивалентно следующему выражению с переключателем:

```
CASE WHEN V1 IS NOT NULL THEN V1 ELSE V2 END.
```

Выражение `COALESCE (V1, V2, . . . Vn)` для $n \geq 3$ эквивалентно следующему выражению с переключателем:

```
CASE WHEN V1 IS NOT NULL THEN V1 ELSE COALESCE (V2,... n) END.
```

ОБЩАЯ СТРУКТУРА ОПЕРАТОРА ВЫБОРКИ



СИНТАКСИС. НАЧАЛО ОБРАБОТКИ.

```
SELECT [ ALL | DISTINCT ] select_item_commalist
   FROM table_reference_commalist
     [ WHERE conditional_expression ]
       [ GROUP BY column_name_commalist ]
         [ HAVING conditional_expression ]
           [ ORDER BY order_item_commalist ]
```

1. Этап FROM
2. Этап WHERE

СЛЕДУЮЩИЕ ЭТАПЫ

1. GROUP BY
2. HAVING
3. HAVING без GROUP BY
4. GROUP BY без HAVING

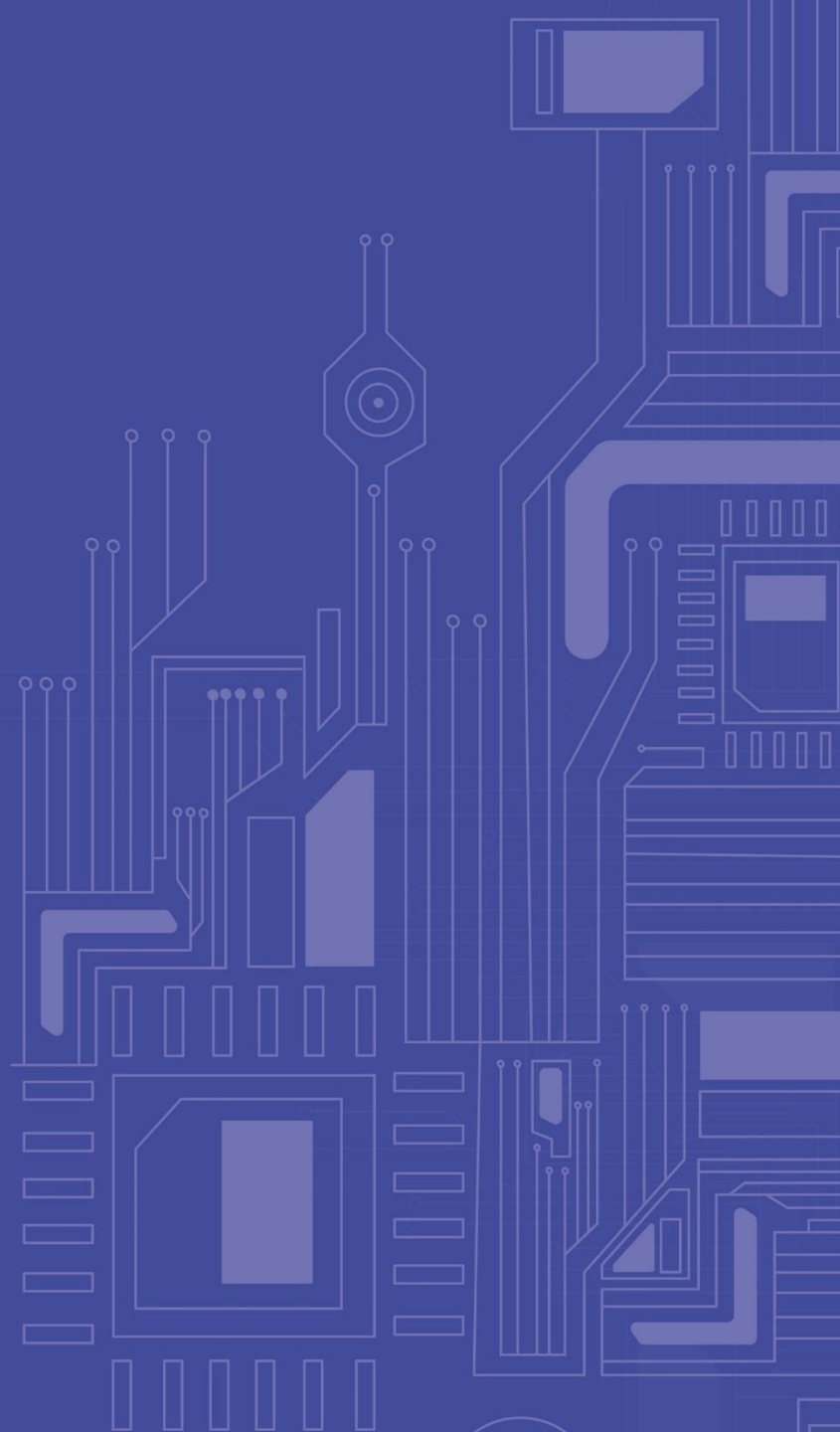
ФОРМИРОВАНИЕ РЕЗУЛЬТАТА (1/2)

1. Последовательное формирование результирующей таблицы
2. Отсутствие DISTINCT – на что влияет

ФОРМИРОВАНИЕ РЕЗУЛЬТАТА (2/2)

1. ORDER BY
2. Старый и новый синтаксис

ССЫЛКИ НА ТАБЛИЦЫ РАЗДЕЛА FROM



СИНТАКСИС

```
FROM table_reference_commalist
```

```
table_reference ::= table_primary | joined_table
table_primary ::= table_or_query_name [ [ AS ] correlation_name
     [ (derived_column_list) ] ]
   | derived_table [ [ AS ] correlation_name
     [ (derived_column_list) ] ]
   | lateral_derived_table [ [ AS ] correlation_name
     [ (derived_column_list) ] ]
   | collection_derived_table [ [ AS ] correlation_name
     [ (derived_column_list) ] ]
   | ONLY (table_or_query_name)[ [ AS ] correlation_name
     [ (derived_column_list) ] ]
   | (joined_table)
table_or_query_name ::= { table_name | query_name }
derived_table ::= (query_expression)
lateral_derived_table ::= LATERAL (query_expression)
collection_derived_table ::= UNNEST
   (collection_value_exression) [ WITH ORDINALITY ]
```

ТАБЛИЧНОЕ ВЫРАЖЕНИЕ, СПЕЦИФИКАЦИЯ ЗАПРОСОВ И ВЫРАЖЕНИЕ ЗАПРОСОВ

Табличным выражением (table_expression) называется конструкция

```
table_expression ::= FROM table_reference_commalist
   [ WHERE conditional_expression ]
     [ GROUP BY column_name_commalist ]
       [ HAVING conditional_expression ]
```

Спецификацией запроса (query_specification) называется конструкция

```
query_specification SELECT [ ALL | DISTINCT ]
   select_item_commalist table_expression
```

Наконец, *выражением запросов* (query_expression) называется конструкция

```
query_expression ::= [ with_clause ] query_expression_body
query_expression_body ::= { non_join_query_expression
   | joined_table }
non_join_query_expression ::= non_join_query_term
   | query_expression_body
     { UNION | EXCEPT }[ ALL | DISTINCT ]
       [ corresponding_spec ] query_term
query_term ::= non_join_query_term | joined_table
non_join_query_term ::= non_join_query_primary
   | query_term INTERSECT [ ALL | DISTINCT ]
     [ corresponding_spec ] query_primary
query_primary ::= non_join_query_primary | joined_table
non_join_query_primary ::= simple_table
   | (non_join_query_expression)
simple_table ::= query_specification
   | table_value_constructor
   | TABLE table_name
corresponding_spec ::= CORRESPONDING
   [ BY column_name_comma_list ]
```

ПРАВИЛА ВЧИСЛЕНИЯ ВЫРАЖЕНИЙ ЗАПРОСОВ (1/2)

1. Из чего строится выражение запросов
2. Правила вычисления выражений запросов:

- Если выражение запросов не включает ни одной теоретико-множественной операции, то результатом вычисления выражения запросов является результат вычисления простой или соединенной таблицы.
- Если в терме (non_join_query_term) или выражении запросов (non_join_query_expression) без соединения присутствует теоретико-множественная операция, то пусть T1, T2 и TR обозначают соответственно первый операнд, второй операнд и результат терма или выражения соответственно, а OP – используемую теоретико-множественную операцию.
- Если в операции присутствует спецификация CORRESPONDING, то:
  - если присутствует конструкция BY column_name_comma_list, то все имена в этом списке должны быть различны, и каждое имя должно являться одновременно именем некоторого столбца таблицы T1 и именем некоторого столбца таблицы T2, причем типы этих столбцов должны быть совместимыми; обозначим данный список имен через SL;
  - если список соответствия столбцов не задан, пусть SL обозначает список имен столбцов, являющихся именами столбцов и в T1, и в T2, в том порядке, в котором эти имена фигурируют в T1;
  - вычисляемые терм или выражение запросов без соединения эквивалентны выражению (SELECT SL FROM T1) OP (SELECT SL FROM T2), не включающему спецификацию CORRESPONDING.

ПРАВИЛА ВЧИСЛЕНИЯ ВЫРАЖЕНИЙ ЗАПРОСОВ (2/2)

- При отсутствии в операции спецификации CORRESPONDING операция выполняется таким образом, как если бы эта спецификация присутствовала и включала конструкцию BY column_name_comma_list, в которой были бы перечислены все столбцы таблицы T1.
- При выполнении операции OP две строки s1 с именами столбцов c1, c2, …, cn и s2 с именами столбцов d1, d2, …, dn считаются строками-дубликатами, если для каждого i (i = 1, 2, …, n) либо ci и di не содержат NULL, и (ci = di) = true, либо и ci, и di содержат NULL.
- Если в операции OP не задана спецификация ALL, то в TR строки-дубликаты удаляются.
- Если спецификация ALL задана, то пусть s – строка, являющаяся дубликатом некоторой строки T1, или некоторой строки T2, или обеих; пусть m – число дубликатов s в T1, а n – число дубликатов s в T2. Тогда:
  - если указана операция UNION, то число дубликатов s в TR равно m + n;
  - если указана операция EXCEPT, то число дубликатов s в TR равно max ((m-n),0);
  - если указана операция INTERSECT, то число дубликатов s в TR равно min (m,n).

РАЗДЕЛ WITH ВЫРАЖЕНИЯ ЗАПРОСОВ

```
with_clause ::= WITH [ RECURSIVE ] with_element_comma_list
with_element ::= query_name [ (column_name_list) ]
   AS (query_expression) [ search_or_cycle_clause ]
```

```
WITH query_name (c1, c2, … cn) AS (query_exp_1) query_exp_2
```

КОНСТРУКТОРЫ ЗНАЧЕНИЯ СТРОКИ И ТАБЛИЦЫ

```
row_value_constructor ::= row_value_constructor_element
   | [ ROW ] (row_value_constructor_element_comma_list)
   | row_subquery
row_value_constructor_element ::= value_expression | NULL | DEFAULT
```

```
table_value_constructor ::= VALUES
   row_value_constructor_comma_list
```

ССЫЛКИ НА ТАБЛИЦЫ

Ссылки на базовые, представляемые и порождаемые таблицы

```
table_reference ::= table_primary
table_primary ::= table_or_query_name [ [ AS ] correlation_name
     [ (derived_column_list) ] ]
   | derived_table [ AS ] correlation_name
     [ (derived_column_list) ]
table_or_query_name ::= { table_name | query_name }
derived_table ::= (query_expression)
```

ПРЕДСТАВЛЕНИЯ (VIEW) (1/2)

1. Зачем нужны
2. Синтаксис

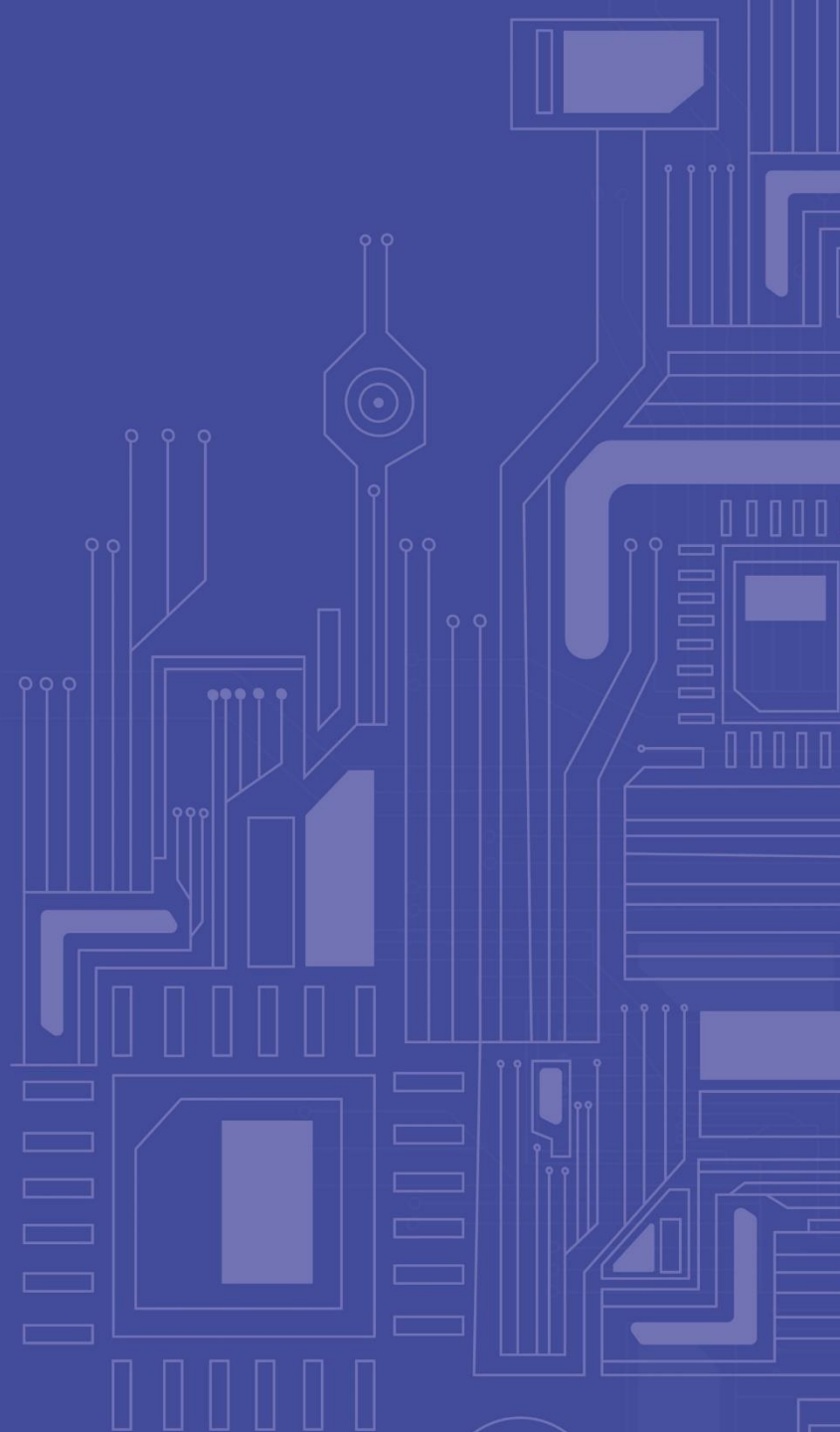
```
create_view ::= CREATE [ RECURSIVE ] VIEW table_name
     [ column_name_comma_list ]
   AS query_expression
   [ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

```
create_view ::= CREATE VIEW table_name
     [ column_name_comma_list ]
   AS query_expression
```

ПРЕДСТАВЛЕНИЯ (VIEW) (2/2)

Практическое использование в бизнес-архитектурах

ПРИМЕРЫ



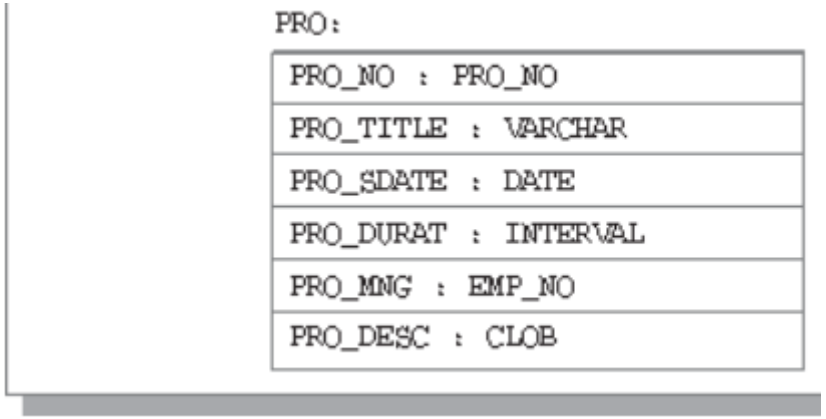
ПРИМЕРЫ ОПРЕДЕЛЕНИЯ БАЗОВЫХ ТАБЛИЦ (1/2)

DEPT:

DEPT_NO : DEPT_NO
DEPT_NAME : VARCHAR
DEPT_EMP_NO : INTEGER
DEPT_TOTAL_SAL : SALARY
DEPT_MNG : EMP_NO

```
(1)     CREATE TABLE DEPT (
(2)     DEPT_NO DEPT_NO PRIMARY KEY,
(3)     DEPT_EMP_NO INTEGER NO NULL CHECK (
        VALUE BETWEEN 1 AND 100),
(4)     DEPT_NAME VARCHAR(200) DEFAULT 'Nameless' NOT NULL,
(5)     DEPT_TOTAL_SAL SALARY DEFAULT 1000000.00
   NO NULL CHECK (VALUE > = 100000.00),
(6)     DEPT_MNG EMP_NO DEFAULT NULL
      REFERENCES EMP ON DELETE SET NULL
      CHECK (IF (VALUE IS NOT NULL) THEN
         ((SELECT COUNT(*) FROM DEPT
            WHERE DEPT.DEPT_MNG = VALUE) = 1),
(7)     CHECK (DEPT_EMP_NO =
      (SELECT COUNT(*) FROM EMP
         WHERE DEPT_NO = EMP.DEPT_NO)),
(8)     CHECK (DEPT_TOTAL_SAL >=
      (SELECT SUM(EMP_SAL) FROM EMP
         WHERE DEPT_NO = EMP.DEPT_NO)));
```

ПРИМЕРЫ ОПРЕДЕЛЕНИЯ БАЗОВЫХ ТАБЛИЦ (2/2)




```
(1)     CREATE TABLE PRO (
(2)     PRO_NO PRO_NO PRIMARY KEY,
(3)     PRO_TITLE VARCHAR(200)DEFAULT 'No title' NOT NULL,
(4)     PRO_SDATE DATE DEFAULT CURRENT_DATE NOT NULL,
(5)     PRO_DURAT INTERVAL YEAR DEFAUL INTERVAL '1'
        YEAR NOT NULL,
(6)     PRO_MNG EMP_NO UNIQUE NOT NULL
        REFERENCES EMP ON DELETE NO ACTION,
(7)     PRO_DESC CLOB(10M));
```

РАЗНЫЕ БАЗЫ ДАННЫХ И СУБД (1/2)

1. MS SQL
2. Postgres
3. Mongo
4. SQlite

РАЗНЫЕ БАЗЫ ДАННЫХ И СУБД (2/2)

Как получить список всех таблиц из разных БД:

```
-- H2, HSQLDB, MySQL, PostgreSQL, SQL Server
        SELECT table_schema, table_name
        FROM information_schema.tables
-- DB2
        SELECT tabschema, tabname
        FROM syscat.tables
-- Oracle
        SELECT owner, table_name
        FROM all_tables
-- SQLite
        SELECT name
        FROM sqlite_master
-- Teradata
        SELECT databasename, tablename
        FROM dbc.tables
```

ПРИМЕР СОЗДАНИЯ ТАБЛИЦ В MS SQL

Создать *таблицу* для хранения данных о товарах, поступающих в продажу в некоторой торговой фирме. Необходимо учесть такие сведения, как название и тип товара, его цена, сорт и город, где товар производится.

```
CREATE TABLE Товар
(Название       VARCHAR(50) NOT NULL,
    Цена        MONEY NOT     NULL,
    Тип         VARCHAR(50) NOT NULL,
    Сорт        VARCHAR(50),
    ГородТовара VARCHAR(50))
```

Создать таблицу для сохранения сведений о постоянных клиентах с указанием названий города и фирмы, фамилии, имени и отчества клиента, номера его телефона.

```
CREATE TABLE  Клиент
    (Фирма       VARCHAR(50) NOT NULL,
    Фамилия      VARCHAR(50) NOT NULL,
    Имя          VARCHAR(50) NOT NULL,
    Отчество     VARCHAR(50),
    ГородКлиента VARCHAR(50),
    Телефон      CHAR(10) NOT NULL)
```

ПРИМЕРЫ SELECT В MS SQL

Составить *список* сведений о всех клиентах.

```
SELECT * FROM Клиент
```

Составить список всех фирм.

```
SELECT ALL Клиент.Фирма
FROM Клиент
```

Или (что эквивалентно)

```
SELECT Клиент.Фирма
FROM Клиент
```

Составить список всех уникальных фирм.

```
SELECT DISTINCT Клиент.Фирма
FROM Клиент
```

ПРИМЕР БД С ТЕСТАМИ ПРИБОРОВ

```
ivasonik@DESKTOP-8O2FCMA:~$ sqlite3 dev-tests.s3db
```

```
ivasonik@DESKTOP-8O2FCMA:~$ nano schema.sql
```

schema.sql

```
CREATE table devices (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        sn TEXT NOT NULL UNIQUE
);

CREATE TABLE tests (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        ts INTEGER NOT NULL,
        device_id INTEGER NOT NULL REFERENCES devices(id),
        result INTEGER --0 - fail, 1 - passed
);
```

```
ivasonik@DESKTOP-8O2FCMA:~$ sqlite3 dev-tests.s3db <schema.sql
```

CONNECTION

Краткое упоминание:

При работе с БД на клиенте, внутри БД, занимаясь исключительно запросами, работаем с файловым дескриптором – работаем внутри файловой системы

При работе приложения, клиент-серверная архитектура – работаем с дескриптором соединения, курсором, запросы передаются, результаты возвращаются через соединение, а не крутятся внутри файловой системы на клиенте.

Для автоматизации соединения создаем connection, передаем cursor, который соответствует одной транзакции.

ORM

Краткое упоминание:

Проектировать БД можно вручную на берегу, продумывая архитектуру, а можно на берегу писать ко приложения, а на основе объектов автоматически создать БД посредством Object Relation Manager.

Плюсы и минусы.

Вернемся подробнее на 13й лекции. Здесь нужно, чтоб «с вертолета» оглядеть практику создания БД.


ВСЁ!