ARENADATA

Корпоративная платформа хранения
и обработки больших данных

# Arenadata DB для Разработчиков

## Часть 2

# ADB – аналитическая СУБД для больших данных

**Простые таблицы в ADB:**
Семейства и типы хранения. Сжатие данных.
Распределение данных.

**Партиционированные таблицы в ADB:**
Устройство. Операции с партициями.

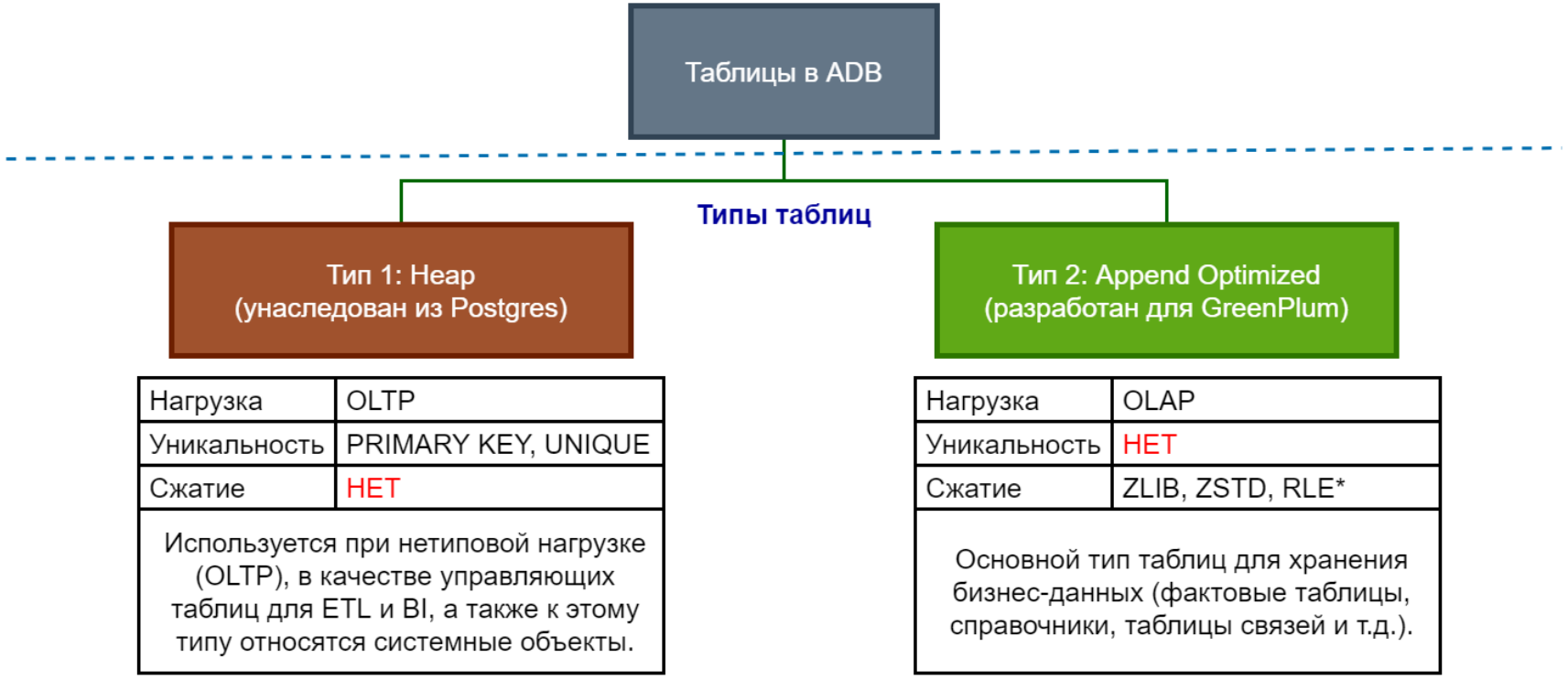
**Типы данных.**
Рекомендации. Сопоставление с Oracle. JSON и JSONB. XML.

ARENADATA

Корпоративная платформа хранения
и обработки больших данных

# Простые таблицы в ADB

# Типы таблиц в ADB: Heap и Append-Optimized



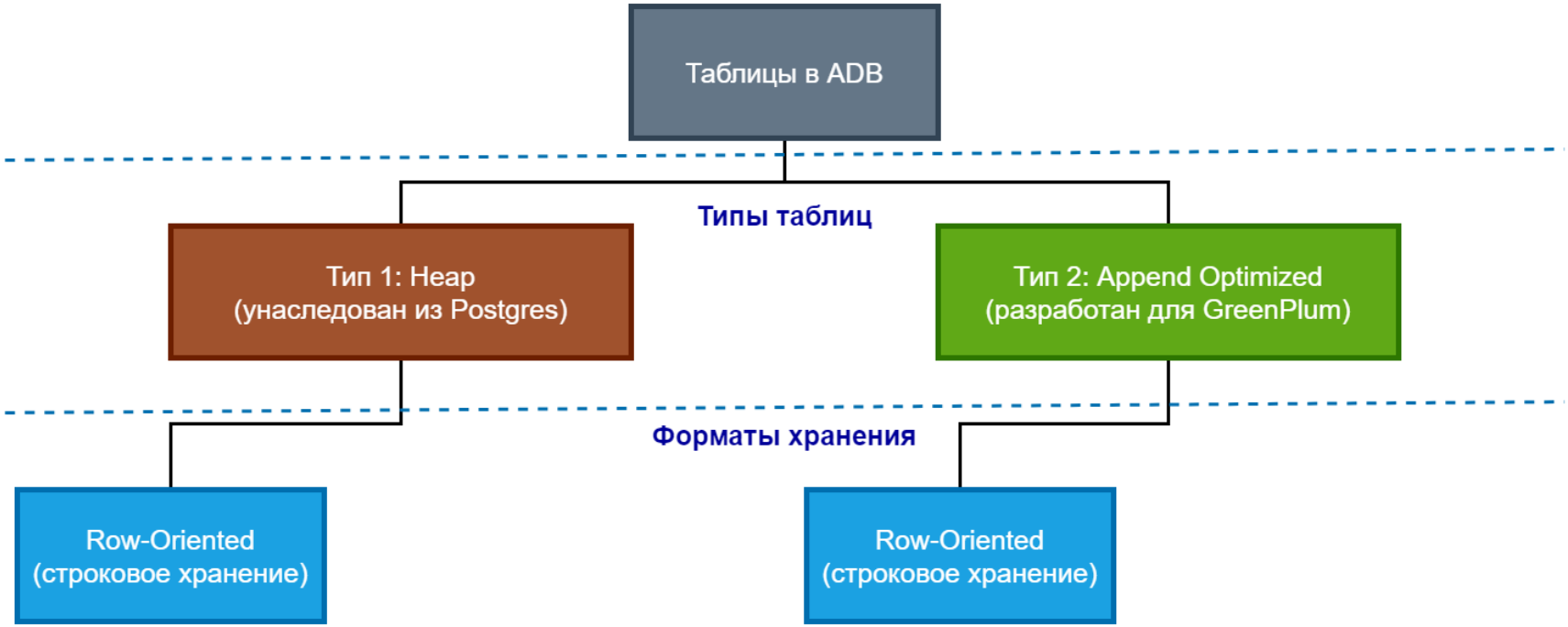

| | |
|---|---|
| Нагрузка | OLTP |
| Уникальность | PRIMARY KEY, UNIQUE |
| Сжатие | НЕТ |

Используется при нетиповой нагрузке (OLTP), в качестве управляющих таблиц для ETL и BI, а также к этому типу относятся системные объекты.

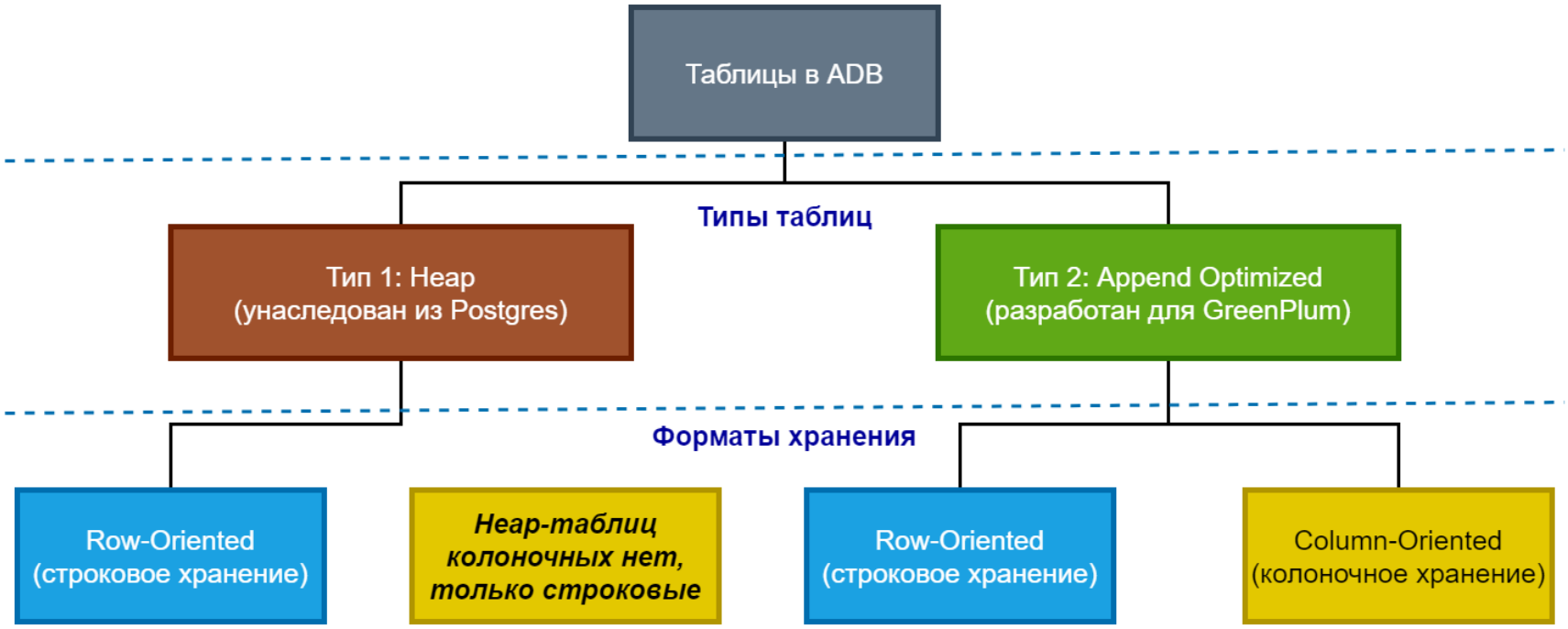
| | |
|---|---|
| Нагрузка | OLAP |
| Уникальность | НЕТ |
| Сжатие | ZLIB, ZSTD, RLE* |

Основной тип таблиц для хранения бизнес-данных (фактовые таблицы, справочники, таблицы связей и т.д.).

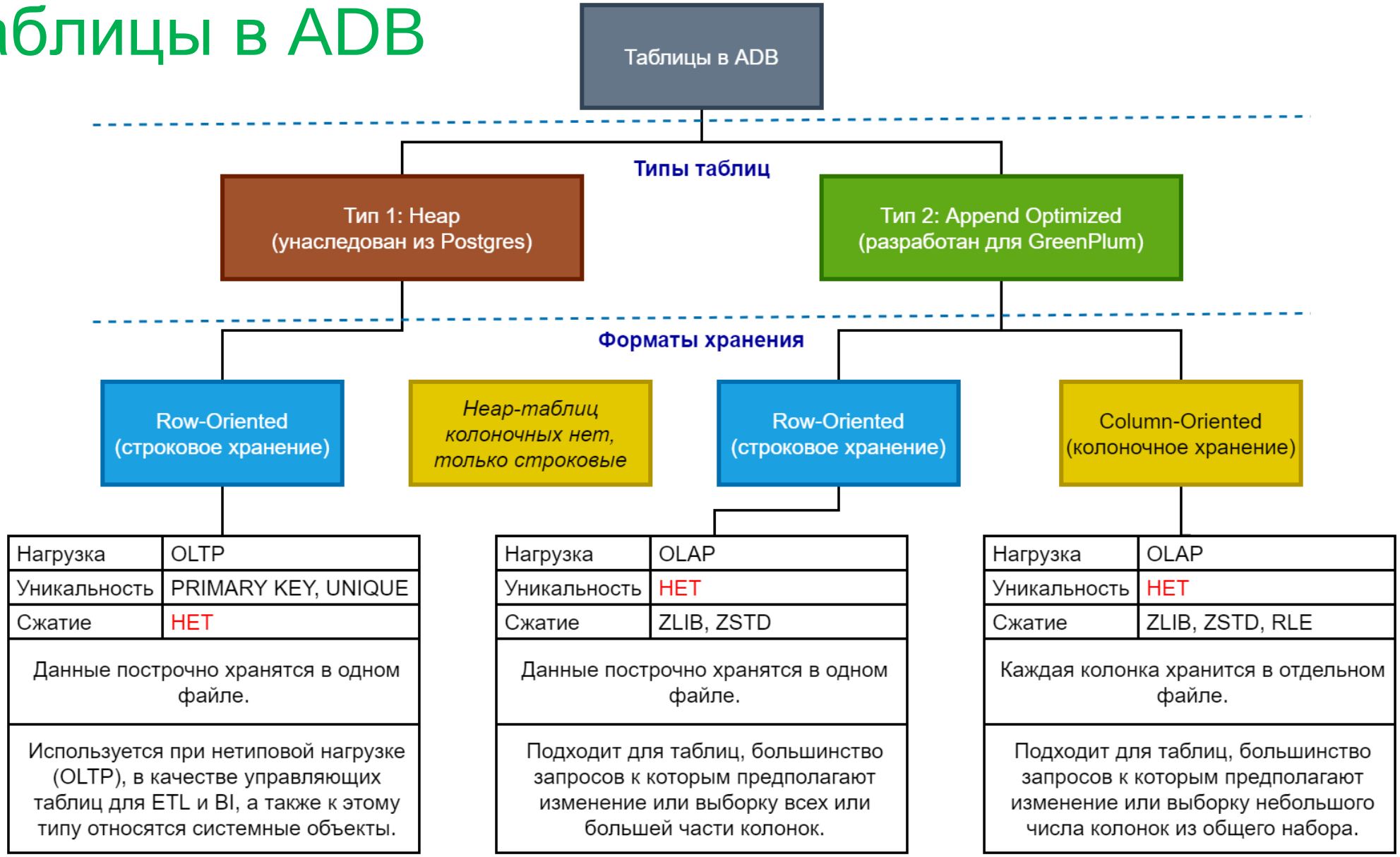
# Форматы хранения таблиц в ADB: строковый



# Форматы хранения таблиц в ADB: колоночный



# Таблицы в ADB




ARENADATA

# Сжатие данных таблиц в ADB

- Сжимать данные можно только в AO-таблицах.
- Тип и степень сжатия изменить после создания таблицы нельзя.
- Для строковых (ROW-oriented) доступно два кодека: ZLIB и ZSTD.
- Для колоночных (COLUMN-oriented) доступно три: ZLIB, ZSTD, RLE.
- Используя сжатие, вы получаете экономию по памяти, тратя ресурс CPU.
- Уровень сжатия выше 5 на практике не эффективен, накладные расходы по CPU превышают пользу от экономии дискового пространства.
- Для подавляющего большинства задач лучшей будет сжатие ZSTD с уровнем 1 (соизмеримый ZLIB с уровнем 5).
- Данные на практике сжимаются хорошо, самый негативный сценарий – сжатие вдвое, обычно получается сжать в 5-7 раз (и даже выше для банковских данных).
- RLE иногда даёт огромный прирост, если кардинальность данных в столбце низкая.

# Таблицы в ADB: синтаксис создания

```
CREATE {TEMPORARY | TEMP} | UNLOGGED] TABLE table_name (
[ { column_name data_type [ DEFAULT default_expr ]
[column_constraint [ ... ] [ ENCODING (COMPRESSTYPE={ZLIB | ZSTD | RLE_TYPE | NONE} [COMPRESSLEVEL={0-9} ]
[BLOCKSIZE={8192-2097152} ])]]
| table_constraint
| LIKE other_table [{INCLUDING|EXCLUDING} {DEFAULTS|CONSTRAINTS|INDEXES|STORAGE|COMMENTS|ALL}] ...}[, ... ] ] )
[ INHERITS ( parent_table [, ... ] ) ]
[ WITH (
[{APPENDOPTIMIZED | APPENDONLY}={TRUE|FALSE}, ]
[ BLOCKSIZE={8192-2097152}, ]
[ ORIENTATION={COLUMN|ROW}, ]
[ CHECKSUM={TRUE|FALSE},]
[ COMPRESSTYPE={ZLIB|ZSTD|RLE_TYPE|NONE}, ]
[ COMPRESSLEVEL={0-9}, ]
[ FILLFACTOR={10-100}, ]
[OIDS[=TRUE|FALSE] ]) ]
[ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]
[ TABLESPACE tablespace ]
[ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED ]
[ PARTITION BY partition_type (column)...]
```

# Таблицы в ADB: синтаксис создания

```
CREATE [{TEMPORARY | TEMP} | UNLOGGED] TABLE table_name (
     [ { column_name data_type [ DEFAULT default_expr ]    [column_constraint [ ... ] [ ENCODING (
          COMPRESSTYPE={ZLIB | ZSTD | RLE_TYPE | NONE} [COMPRESSLEVEL={0-9} ] [BLOCKSIZE={8192-2097152} ]
     )]]
        | table_constraint
        | LIKE other_table [{INCLUDING|EXCLUDING} {DEFAULTS|CONSTRAINTS|INDEXES|STORAGE|COMMENTS|ALL}] ...}[, ... ] ] )
   [ INHERITS ( parent_table [, ... ] ) ]
   [ WITH (
     {APPENDOPTIMIZED | APPENDONLY}={TRUE|FALSE}
     BLOCKSIZE={8192-2097152}
     ORIENTATION={COLUMN|ROW}
     CHECKSUM={TRUE|FALSE}
     COMPRESSTYPE={ZLIB|ZSTD|RLE_TYPE|NONE}
     COMPRESSLEVEL={0-9}
     FILLFACTOR={10-100}
     OIDS[=TRUE|FALSE])
   [ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]
   [ TABLESPACE tablespace ]
   [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED ]
   [ PARTITION BY partition_type (column)…]
```

# Таблицы в ADB: синтаксис создания

```
CREATE [{TEMPORARY | TEMP} | UNLOGGED] TABLE table_name (
     [ { column_name data_type [ DEFAULT default_expr ]    [column_constraint [ ... ] [ ENCODING (
          COMPRESSTYPE={ZLIB | ZSTD | RLE_TYPE | NONE} [COMPRESSLEVEL={0-9} ] [BLOCKSIZE={8192-2097152} ]
     )]]
        | table_constraint
        | LIKE other_table [{INCLUDING|EXCLUDING} {DEFAULTS|CONSTRAINTS|INDEXES|STORAGE|COMMENTS|ALL}] ...}[, ... ] ] )
   [ INHERITS ( parent_table [, ... ] ) ]
   [ WITH (
     {APPENDOPTIMIZED | APPENDONLY}={TRUE|FALSE}
     BLOCKSIZE={8192-2097152}
     ORIENTATION={COLUMN|ROW}
     CHECKSUM={TRUE|FALSE}
     COMPRESSTYPE={ZLIB|ZSTD|RLE_TYPE|NONE}
     COMPRESSLEVEL={0-9}
     FILLFACTOR={10-100}
     OIDS[=TRUE|FALSE])
   [ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]
   [ TABLESPACE tablespace ]
   [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED ]
   [ PARTITION BY partition_type (column)…]
```

# Таблицы в ADB: синтаксис создания

```
CREATE [{TEMPORARY | TEMP} | UNLOGGED] TABLE table_name (
     [ { column_name data_type [ DEFAULT default_expr ]    [column_constraint [ ... ] [ ENCODING (
          COMPRESSTYPE={ZLIB | ZSTD | RLE_TYPE | NONE} [COMPRESSLEVEL={0-9} ] [BLOCKSIZE={8192-2097152} ]
     )]]
        | table_constraint
        | LIKE other_table [{INCLUDING|EXCLUDING} {DEFAULTS|CONSTRAINTS|INDEXES|STORAGE|COMMENTS|ALL}] ...}[, ... ] ] )
   [ INHERITS ( parent_table [, ... ] ) ]
   [ WITH (
     {APPENDOPTIMIZED | APPENDONLY}={TRUE|FALSE}
     BLOCKSIZE={8192-2097152}
     ORIENTATION={COLUMN|ROW}
     CHECKSUM={TRUE|FALSE}
     COMPRESSTYPE={ZLIB|ZSTD|RLE_TYPE|NONE}
     COMPRESSLEVEL={0-9}
     FILLFACTOR={10-100}
     OIDS[=TRUE|FALSE])
   [ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]
   [ TABLESPACE tablespace ]
   [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED ]
   [ PARTITION BY partition_type (column)…]
```

*Сжатие по колонкам - только для колоночных AO-таблиц*

# Таблицы в ADB: синтаксис создания

```
CREATE [{TEMPORARY | TEMP} | UNLOGGED] TABLE table_name (
      [ { column_name data_type [ DEFAULT default_expr ]    [column_constraint [ ... ] [ ENCODING (
           COMPRESSTYPE={ZLIB | ZSTD | RLE_TYPE | NONE} [COMPRESSLEVEL={0-9} ] [BLOCKSIZE={8192-2097152} ]
      )]]
         | table_constraint
         | LIKE other_table [{INCLUDING|EXCLUDING} {DEFAULTS|CONSTRAINTS|INDEXES|STORAGE|COMMENTS|ALL}] ...}[, ... ] ] )
   [ INHERITS ( parent_table [, ... ] ) ]
   [ WITH (
     {APPENDOPTIMIZED | APPENDONLY}={TRUE|FALSE}
     BLOCKSIZE={8192-2097152}
     ORIENTATION={COLUMN|ROW}
     CHECKSUM={TRUE|FALSE}
     COMPRESSTYPE={ZLIB|ZSTD|RLE_TYPE|NONE}
     COMPRESSLEVEL={0-9}
     FILLFACTOR={10-100}
     OIDS[=TRUE|FALSE])
   [ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]
   [ TABLESPACE tablespace ]
   [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED ]
   [ PARTITION BY partition_type (column)…]
```

*Вместо описания колонок можно взять их перечень и свойства из другой таблицы*

ARENADATA

# Таблицы в ADB: синтаксис создания

```
CREATE [{TEMPORARY | TEMP} | UNLOGGED] TABLE table_name (
     [ { column_name data_type [ DEFAULT default_expr ]    [column_constraint [ ... ] [ ENCODING (
          COMPRESSTYPE={ZLIB | ZSTD | RLE_TYPE | NONE} [COMPRESSLEVEL={0-9} ] [BLOCKSIZE={8192-2097152} ]
     )]]
        | table_constraint
        | LIKE other_table [{INCLUDING|EXCLUDING} {DEFAULTS|CONSTRAINTS|INDEXES|STORAGE|COMMENTS|ALL}] ...}[, ... ] ] )
   [ INHERITS ( parent_table [, ... ] ) ]
   [ WITH (
     {APPENDOPTIMIZED | APPENDONLY}={TRUE|FALSE}
     BLOCKSIZE={8192-2097152}
     ORIENTATION={COLUMN|ROW}
     CHECKSUM={TRUE|FALSE}
     COMPRESSTYPE={ZLIB|ZSTD|RLE_TYPE|NONE}
     COMPRESSLEVEL={0-9}
     FILLFACTOR={10-100}
     OIDS[=TRUE|FALSE])
   [ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]
   [ TABLESPACE tablespace ]
   [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED ]
   [ PARTITION BY partition_type (column)…]
```

# Таблицы в ADB: синтаксис создания

```
CREATE [{TEMPORARY | TEMP} | UNLOGGED] TABLE table_name (
     [ { column_name data_type [ DEFAULT default_expr ]    [column_constraint [ ... ] [ ENCODING (
          COMPRESSTYPE={ZLIB | ZSTD | RLE_TYPE | NONE} [COMPRESSLEVEL={0-9} ] [BLOCKSIZE={8192-2097152} ]
     )]]
        | table_constraint
        | LIKE other_table [{INCLUDING|EXCLUDING} {DEFAULTS|CONSTRAINTS|INDEXES|STORAGE|COMMENTS|ALL}] ...}[, ... ] ] )
   [ INHERITS ( parent_table [, ... ] ) ]
   [ WITH (
     {APPENDOPTIMIZED | APPENDONLY}={TRUE|FALSE}
     BLOCKSIZE={8192-2097152}
     ORIENTATION={COLUMN|ROW}
     CHECKSUM={TRUE|FALSE}
     COMPRESSTYPE={ZLIB|ZSTD|RLE_TYPE|NONE}
     COMPRESSLEVEL={0-9}
     FILLFACTOR={10-100}
     OIDS[=TRUE|FALSE])
   [ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]
   [ TABLESPACE tablespace ]
   [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED ]
   [ PARTITION BY partition_type (column)…]
```

*Внутри блока параметры указываются через запятую. Параметры опциональны*

# Таблицы в ADB: синтаксис создания

```
CREATE [{TEMPORARY | TEMP} | UNLOGGED] TABLE table_name (
      [ { column_name data_type [ DEFAULT default_expr ]    [column_constraint [ ... ] [ ENCODING (
            COMPRESSTYPE={ZLIB | ZSTD | RLE_TYPE | NONE} [COMPRESSLEVEL={0-9} ] [BLOCKSIZE={8192-2097152} ]
      )]]
         | table_constraint
         | LIKE other_table [{INCLUDING|EXCLUDING} {DEFAULTS|CONSTRAINTS|INDEXES|STORAGE|COMMENTS|ALL}] ...}[, ... ] ] )
   [ INHERITS ( parent_table [, ... ] ) ]
   [ WITH (
     {APPENDOPTIMIZED | APPENDONLY}={TRUE|FALSE} – В конфигурации по умолчанию создаются Heap-таблицы
     BLOCKSIZE={8192-2097152}
     ORIENTATION={COLUMN|ROW}
     CHECKSUM={TRUE|FALSE}
     COMPRESSTYPE={ZLIB|ZSTD|RLE_TYPE|NONE}
     COMPRESSLEVEL={0-9}
     FILLFACTOR={10-100}
     OIDS[=TRUE|FALSE])
   [ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]
   [ TABLESPACE tablespace ]
   [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED ]
   [ PARTITION BY partition_type (column)…]
```

# Таблицы в ADB: синтаксис создания

```
CREATE [{TEMPORARY | TEMP} | UNLOGGED] TABLE table_name (
     [ { column_name data_type [ DEFAULT default_expr ]    [column_constraint [ ... ] [ ENCODING (
          COMPRESSTYPE={ZLIB | ZSTD | RLE_TYPE | NONE} [COMPRESSLEVEL={0-9} ] [BLOCKSIZE={8192-2097152} ]
     )]]
        | table_constraint
        | LIKE other_table [{INCLUDING|EXCLUDING} {DEFAULTS|CONSTRAINTS|INDEXES|STORAGE|COMMENTS|ALL}] ...}[, ... ] ] )
   [ INHERITS ( parent_table [, ... ] ) ]
   [ WITH (
     {APPENDOPTIMIZED | APPENDONLY}={TRUE|FALSE} – В конфигурации по умолчанию создаются Heap-таблицы
     BLOCKSIZE={8192-2097152} – размер блока сжатия AO-таблицы. К Heap-таблицам не применимо
     ORIENTATION={COLUMN|ROW}
     CHECKSUM={TRUE|FALSE}
     COMPRESSTYPE={ZLIB|ZSTD|RLE_TYPE|NONE}
     COMPRESSLEVEL={0-9}
     FILLFACTOR={10-100}
     OIDS[=TRUE|FALSE])
   [ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]
   [ TABLESPACE tablespace ]
   [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED ]
   [ PARTITION BY partition_type (column)…]
```

# Таблицы в ADB: синтаксис создания

```
CREATE [{TEMPORARY | TEMP} | UNLOGGED] TABLE table_name (
     [ { column_name data_type [ DEFAULT default_expr ]    [column_constraint [ ... ] [ ENCODING (
          COMPRESSTYPE={ZLIB | ZSTD | RLE_TYPE | NONE} [COMPRESSLEVEL={0-9} ] [BLOCKSIZE={8192-2097152} ]
     )]]
        | table_constraint
        | LIKE other_table [{INCLUDING|EXCLUDING} {DEFAULTS|CONSTRAINTS|INDEXES|STORAGE|COMMENTS|ALL}] ...}[, ... ] ] )
   [ INHERITS ( parent_table [, ... ] ) ]
   [ WITH (
     {APPENDOPTIMIZED | APPENDONLY}={TRUE|FALSE} – В конфигурации по умолчанию создаются Heap-таблицы
     BLOCKSIZE={8192-2097152} – размер блока сжатия AO-таблицы. К Heap-таблицам не применимо
     ORIENTATION={COLUMN|ROW} – формат хранения AO-таблицы. По умолчанию ROW. К Heap-таблицам не применимо
     CHECKSUM={TRUE|FALSE}
     COMPRESSTYPE={ZLIB|ZSTD|RLE_TYPE|NONE}
     COMPRESSLEVEL={0-9}
     FILLFACTOR={10-100}
     OIDS[=TRUE|FALSE])
   [ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]
   [ TABLESPACE tablespace ]
   [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED ]
   [ PARTITION BY partition_type (column)…]
```

# Таблицы в ADB: синтаксис создания

```
CREATE [{TEMPORARY | TEMP} | UNLOGGED] TABLE table_name (
      [ { column_name data_type [ DEFAULT default_expr ]    [column_constraint [ ... ] [ ENCODING (
            COMPRESSTYPE={ZLIB | ZSTD | RLE_TYPE | NONE} [COMPRESSLEVEL={0-9} ] [BLOCKSIZE={8192-2097152} ]
      )]]
         | table_constraint
         | LIKE other_table [{INCLUDING|EXCLUDING} {DEFAULTS|CONSTRAINTS|INDEXES|STORAGE|COMMENTS|ALL}] ...}[, ... ] ] )
   [ INHERITS ( parent_table [, ... ] ) ]
   [ WITH (
     {APPENDOPTIMIZED | APPENDONLY}={TRUE|FALSE} – В конфигурации по умолчанию создаются Heap-таблицы
     BLOCKSIZE={8192-2097152} – размер блока сжатия AO-таблицы. К Heap-таблицам не применимо
     ORIENTATION={COLUMN|ROW} – формат хранения AO-таблицы. По умолчанию ROW. К Heap-таблицам не применимо
     CHECKSUM={TRUE|FALSE} – по умолчанию TRUE
     COMPRESSTYPE={ZLIB|ZSTD|RLE_TYPE|NONE}
     COMPRESSLEVEL={0-9}
     FILLFACTOR={10-100}
     OIDS[=TRUE|FALSE])
   [ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]
   [ TABLESPACE tablespace ]
   [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED ]
   [ PARTITION BY partition_type (column)…]
```

# Таблицы в ADB: синтаксис создания

```
CREATE [{TEMPORARY | TEMP} | UNLOGGED] TABLE table_name (
      [ { column_name data_type [ DEFAULT default_expr ]    [column_constraint [ ... ] [ ENCODING (
            COMPRESSTYPE={ZLIB | ZSTD | RLE_TYPE | NONE} [COMPRESSLEVEL={0-9} ] [BLOCKSIZE={8192-2097152} ]
      )]]
         | table_constraint
         | LIKE other_table [{INCLUDING|EXCLUDING} {DEFAULTS|CONSTRAINTS|INDEXES|STORAGE|COMMENTS|ALL}] ...}[, ... ] ] )
   [ INHERITS ( parent_table [, ... ] ) ]
   [ WITH (
     {APPENDOPTIMIZED | APPENDONLY}={TRUE|FALSE} – В конфигурации по умолчанию создаются Heap-таблицы
     BLOCKSIZE={8192-2097152} – размер блока сжатия AO-таблицы. К Heap-таблицам не применимо
     ORIENTATION={COLUMN|ROW} – формат хранения AO-таблицы. По умолчанию ROW. К Heap-таблицам не применимо
     CHECKSUM={TRUE|FALSE} – по умолчанию TRUE
     COMPRESSTYPE={ZLIB|ZSTD|RLE_TYPE|NONE} – тип сжатия AO-таблицы. По умолчанию NONE. К Heap-таблицам не применимо
     COMPRESSLEVEL={0-9} – степень сжатия AO-таблицы. Не используется без указания COMPRESSTYPE. К Heap-таблицам не применимо
     FILLFACTOR={10-100}
     OIDS[=TRUE|FALSE])
   [ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]
   [ TABLESPACE tablespace ]
   [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED ]
   [ PARTITION BY partition_type (column)…]
```

*У колоночных таблиц по данным параметрам будут сжаты колонки, для которых не прописано индивидуальное сжатие*

ARENADATA

# Таблицы в ADB: синтаксис создания

```
CREATE [{TEMPORARY | TEMP} | UNLOGGED] TABLE table_name (
      [ { column_name data_type [ DEFAULT default_expr ]    [column_constraint [ ... ] [ ENCODING (
            COMPRESSTYPE={ZLIB | ZSTD | RLE_TYPE | NONE} [COMPRESSLEVEL={0-9} ] [BLOCKSIZE={8192-2097152} ]
      )]]
         | table_constraint
         | LIKE other_table [{INCLUDING|EXCLUDING} {DEFAULTS|CONSTRAINTS|INDEXES|STORAGE|COMMENTS|ALL}] ...}[, ... ] ] )
   [ INHERITS ( parent_table [, ... ] ) ]
   [ WITH (
      {APPENDOPTIMIZED | APPENDONLY}={TRUE|FALSE} – В конфигурации по умолчанию создаются Heap-таблицы
      BLOCKSIZE={8192-2097152} – размер блока сжатия AO-таблицы. К Heap-таблицам не применимо
      ORIENTATION={COLUMN|ROW} – формат хранения AO-таблицы. По умолчанию ROW. К Heap-таблицам не применимо
      CHECKSUM={TRUE|FALSE} – по умолчанию TRUE
      COMPRESSTYPE={ZLIB|ZSTD|RLE_TYPE|NONE} – тип сжатия AO-таблицы. По умолчанию NONE. К Heap-таблицам не применимо
      COMPRESSLEVEL={0-9} – степень сжатия AO-таблицы. Не используется без указания COMPRESSTYPE. К Heap-таблицам не применимо
      FILLFACTOR={10-100} – степень заполнения стандартного восьмикилобайтного блока Heap-таблицы. К AO-таблицам не применимо
      OIDS[=TRUE|FALSE])
   [ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]
   [ TABLESPACE tablespace ]
   [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED ]
   [ PARTITION BY partition_type (column)…]
```

# Таблицы в ADB: синтаксис создания

```
CREATE [{TEMPORARY | TEMP} | UNLOGGED] TABLE table_name (
      [ { column_name data_type [ DEFAULT default_expr ]    [column_constraint [ ... ] [ ENCODING (
            COMPRESSTYPE={ZLIB | ZSTD | RLE_TYPE | NONE} [COMPRESSLEVEL={0-9} ] [BLOCKSIZE={8192-2097152} ]
      )]]
         | table_constraint
         | LIKE other_table [{INCLUDING|EXCLUDING} {DEFAULTS|CONSTRAINTS|INDEXES|STORAGE|COMMENTS|ALL}] ...}[, ... ] ] )
   [ INHERITS ( parent_table [, ... ] ) ]
   [ WITH (
      {APPENDOPTIMIZED | APPENDONLY}={TRUE|FALSE} – В конфигурации по умолчанию создаются Heap-таблицы
      BLOCKSIZE={8192-2097152} – размер блока сжатия AO-таблицы. К Heap-таблицам не применимо
      ORIENTATION={COLUMN|ROW} – формат хранения AO-таблицы. По умолчанию ROW. К Heap-таблицам не применимо
      CHECKSUM={TRUE|FALSE} – по умолчанию TRUE
      COMPRESSTYPE={ZLIB|ZSTD|RLE_TYPE|NONE} – тип сжатия AO-таблицы. По умолчанию NONE. К Heap-таблицам не применимо
      COMPRESSLEVEL={0-9} – степень сжатия AO-таблицы. Не используется без указания COMPRESSTYPE. К Heap-таблицам не применимо
      FILLFACTOR={10-100} – степень заполнения стандартного восьмикилобайтного блока Heap-таблицы. К AO-таблицам не применимо
      OIDS[=TRUE|FALSE]) – добавлять ли колонку с OID-ами. Для больших таблиц не следует использовать из-за лимита счетчика OID-ов
   [ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]
   [ TABLESPACE tablespace ]
   [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED ]
   [ PARTITION BY partition_type (column)…]
```

# Таблицы в ADB: синтаксис создания

```
CREATE [{TEMPORARY | TEMP} | UNLOGGED] TABLE table_name (
     [ { column_name data_type [ DEFAULT default_expr ]    [column_constraint [ ... ] [ ENCODING (
          COMPRESSTYPE={ZLIB | ZSTD | RLE_TYPE | NONE} [COMPRESSLEVEL={0-9} ] [BLOCKSIZE={8192-2097152} ]
     )]]
        | table_constraint
        | LIKE other_table [{INCLUDING|EXCLUDING} {DEFAULTS|CONSTRAINTS|INDEXES|STORAGE|COMMENTS|ALL}] ...}[, ... ] ] )
   [ INHERITS ( parent_table [, ... ] ) ]
   [ WITH (
     {APPENDOPTIMIZED | APPENDONLY}={TRUE|FALSE}
     BLOCKSIZE={8192-2097152}
     ORIENTATION={COLUMN|ROW}
     CHECKSUM={TRUE|FALSE}
     COMPRESSTYPE={ZLIB|ZSTD|RLE_TYPE|NONE}
     COMPRESSLEVEL={0-9}
     FILLFACTOR={10-100}
     OIDS[=TRUE|FALSE])
   [ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ] – поведение временной таблицы после использования её в транзакции
   [ TABLESPACE tablespace ]
   [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED ]
   [ PARTITION BY partition_type (column)…]
```

# Таблицы в ADB: синтаксис создания

```
CREATE [{TEMPORARY | TEMP} | UNLOGGED] TABLE table_name (
      [ { column_name data_type [ DEFAULT default_expr ]    [column_constraint [ ... ] [ ENCODING (
            COMPRESSTYPE={ZLIB | ZSTD | RLE_TYPE | NONE} [COMPRESSLEVEL={0-9} ] [BLOCKSIZE={8192-2097152} ]
      )]]
         | table_constraint
         | LIKE other_table [{INCLUDING|EXCLUDING} {DEFAULTS|CONSTRAINTS|INDEXES|STORAGE|COMMENTS|ALL}] ...}[, ... ] ] )
   [ INHERITS ( parent_table [, ... ] ) ]
   [ WITH (
     {APPENDOPTIMIZED | APPENDONLY}={TRUE|FALSE}
     BLOCKSIZE={8192-2097152}
     ORIENTATION={COLUMN|ROW}
     CHECKSUM={TRUE|FALSE}
     COMPRESSTYPE={ZLIB|ZSTD|RLE_TYPE|NONE}
     COMPRESSLEVEL={0-9}
     FILLFACTOR={10-100}
     OIDS[=TRUE|FALSE])
   [ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]
   [ TABLESPACE tablespace ] – поместить таблицу в определенный тейблспейс
   [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED ]
   [ PARTITION BY partition_type (column)…]
```

# Таблицы в ADB: синтаксис создания

```
CREATE [{TEMPORARY | TEMP} | UNLOGGED] TABLE table_name (
     [ { column_name data_type [ DEFAULT default_expr ]    [column_constraint [ ... ] [ ENCODING (
          COMPRESSTYPE={ZLIB | ZSTD | RLE_TYPE | NONE} [COMPRESSLEVEL={0-9} ] [BLOCKSIZE={8192-2097152} ]
     )]]
        | table_constraint
        | LIKE other_table [{INCLUDING|EXCLUDING} {DEFAULTS|CONSTRAINTS|INDEXES|STORAGE|COMMENTS|ALL}] ...}[, ... ] ] )
   [ INHERITS ( parent_table [, ... ] ) ]
   [ WITH (
     {APPENDOPTIMIZED | APPENDONLY}={TRUE|FALSE}
     BLOCKSIZE={8192-2097152}
     ORIENTATION={COLUMN|ROW}
     CHECKSUM={TRUE|FALSE}
     COMPRESSTYPE={ZLIB|ZSTD|RLE_TYPE|NONE}
     COMPRESSLEVEL={0-9}
     FILLFACTOR={10-100}
     OIDS[=TRUE|FALSE])
   [ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]
   [ TABLESPACE tablespace ]
   [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED ] – политика распределения таблицы
   [ PARTITION BY partition_type (column)…]
```

*- Если не указать, то таблица будет распределена по ключу в виде первой колонки. Это плохая практика, так делать не стоит*

ARENADATA

# Таблицы в ADB: синтаксис создания

```
CREATE [{TEMPORARY | TEMP} | UNLOGGED] TABLE table_name (
     [ { column_name data_type [ DEFAULT default_expr ]    [column_constraint [ ... ] [ ENCODING (
          COMPRESSTYPE={ZLIB | ZSTD | RLE_TYPE | NONE} [COMPRESSLEVEL={0-9} ] [BLOCKSIZE={8192-2097152} ]
     )]]
        | table_constraint
        | LIKE other_table [{INCLUDING|EXCLUDING} {DEFAULTS|CONSTRAINTS|INDEXES|STORAGE|COMMENTS|ALL}] ...}[, ... ] ] )
   [ INHERITS ( parent_table [, ... ] ) ]
   [ WITH (
     {APPENDOPTIMIZED | APPENDONLY}={TRUE|FALSE}
     BLOCKSIZE={8192-2097152}
     ORIENTATION={COLUMN|ROW}
     CHECKSUM={TRUE|FALSE}
     COMPRESSTYPE={ZLIB|ZSTD|RLE_TYPE|NONE}
     COMPRESSLEVEL={0-9}
     FILLFACTOR={10-100}
     OIDS[=TRUE|FALSE])
   [ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]
   [ TABLESPACE tablespace ]
   [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY | DISTRIBUTED REPLICATED ]
   [ PARTITION BY partition_type (column)…] – партиционирование таблицы. Создается объект более сложный, чем простая таблица
```

# Таблицы в ADB: материализованные представления

- Хранят данные наподобие обычных таблиц.
- Данные не обновляются автоматически. Для обновления данных необходимо выполнить команду:

```
REFRESH MATERIALIZED VIEW <view name>
```

- Только read only.
- Синтаксис создания:

```
CREATE MATERIALIZED VIEW table_name
[ (column_name [, ...] ) ]
[ WITH ( storage_parameter [= value] [, ... ] ) ]
[ TABLESPACE tablespace_name ]
AS query
[ WITH [ NO ] DATA ] – выполнять запрос и заполнять данными при создании или нет
[DISTRIBUTED {| BY column [opclass], [ ... ] | RANDOMLY | REPLICATED }]
```

# Таблицы в ADB: констрейнты

- CHECK – есть для AO и Heap.
- NOT NULL – есть для AO и Heap.
- UNIQUE – только для Heap.
- PRIMARY KEY – только для Heap
- FOREIGN KEY – указать можно для Heap,
  но они не имеют функциональности.

```
CREATE TABLE films (
code        char(5) PRIMARY KEY,
did         integer,
title       varchar(40)
);
```

```
CREATE TABLE films (
code        char(5),
did         integer,
title       varchar(40),
PRIMARY KEY (code,did)
);
```

# Таблицы в ADB: Тейблспейсы

- Позволяют размещать объекты СУБД (таблицы, временные файлы) в разных директориях или на разных дисках;
- Tablespace – глобальный объект, который доступен для всех баз данных внутри СУБД;
- Директория, которая указывается в `LOCATION`, должна быть на всех серверах кластера;
- По умолчанию, существует два tablespace: `pg_global` (часть каталога) и `pg_default` (для объектов).
- Посмотреть все Tablespace в СУБД:

```
SELECT oid, * FROM pg_tablespace;
```

- Посмотреть расположение директорий:

```
SELECT * FROM gp_tablespace_location(<tablespace_oid>);
```

# Таблицы в ADB: Тейблспейсы

Пример создания Tablespace:

```
CREATE TABLESPACE fastdisk LOCATION '/fastdisk/gpdb';
```

Пример создания таблицы с указанием Tablespace:

```
CREATE TABLE foo(i int) TABLESPACE fastdisk;
```

Назначение Tablespace по умолчанию на время сессии:

```
SET default_tablespace = fastdisk;
```

Перенос директории временных файлов (спилов) и временных таблиц:

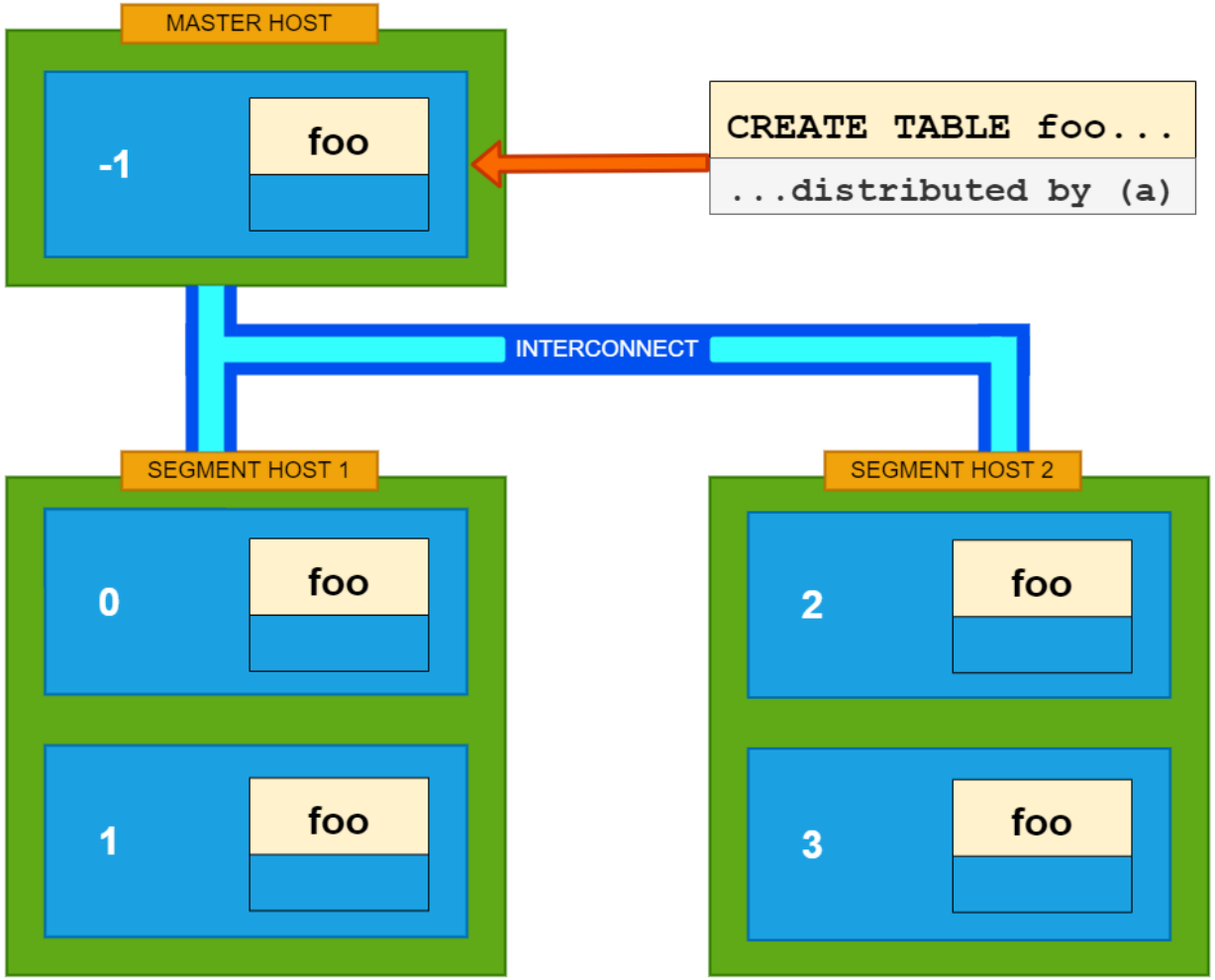
```
gpconfig -c temp_tablespaces –v fastdisk --masteronly
```

# Распределение данных таблиц

Каждая таблица при создании размещается на всех сегментах.

Данные таблиц хранят сегменты на сегментных серверах. Данные построчно распределяются между праймари-сегментами. Одна строка может храниться только на одном праймари-сегменте (кроме случая создания replicated таблиц).

В норме каждому сегменту должна достаться равная доля строк таблицы.



# Типы распределения

Данные таблицы распределяются согласно выбранной для неё политики распределения, которых существует три вида:

1. `DISTRIBUTED RANDOMLY` – планировщик сам раскидывает строки, используя алгоритм round-robin.
2. `DISTRIBUTED BY (column(s))` – строка направляется на конкретный сегмент по хешу ключа.
3. `DISTRIBUTED REPLICATED` – полная копия таблицы хранится на каждом сегменте.

В каждой таблице (кроме replicated-таблиц) есть скрытое поле `gp_segment_id`, которое содержит content того сегмента, на котором находится запись.
Не используйте это поле для больших таблиц в запросах на регулярной основе.

Если тип распределения не указать, таблица распределится по ключу в виде первой колонки.
Это плохая практика, всегда указывайте политику при создании таблицы.

# Распределение данных таблиц

Таблица распределена по ключу в виде колонки a:

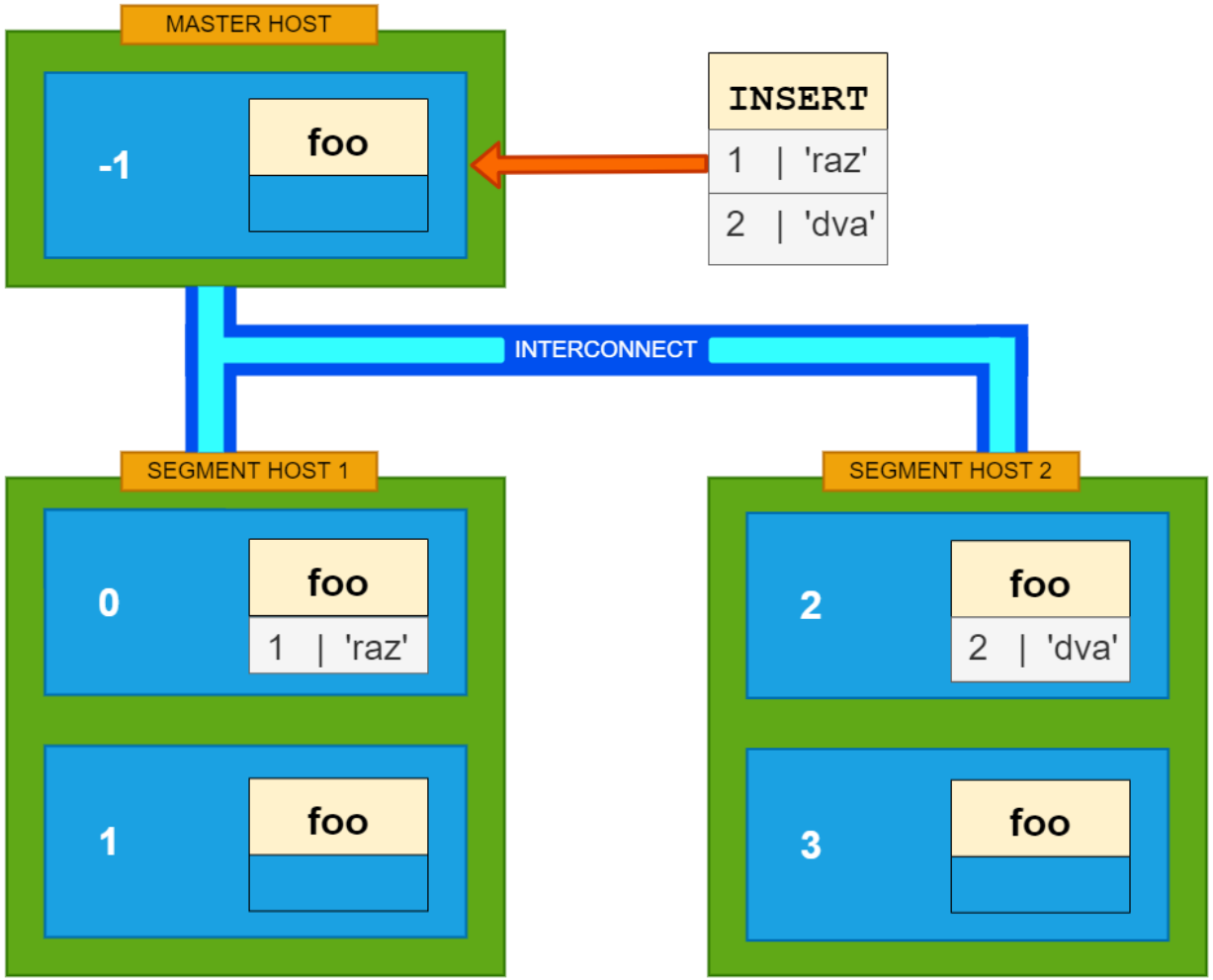
```
adb=# CREATE TABLE foo (a int, b text) DISTRIBUTED BY (a);


adb=# insert into foo values (1,'raz'),(2,'dva');
INSERT 0 2


adb=# select gp_segment_id,* from foo;
 gp_segment_id | a |  b
             0 | 1 | raz
             2 | 2 | dva
```

# Распределенные таблицы

```
adb=# CREATE TABLE foo (a int, b text)
DISTRIBUTED BY (a);
```

```
adb=# insert into foo values
(1,'raz'),(2,'dva');

INSERT 0 2
```

```
adb=# select gp_segment_id,* from foo;

 gp_segment_id | a |  b
             0 | 1 | raz
             2 | 2 | dva
```

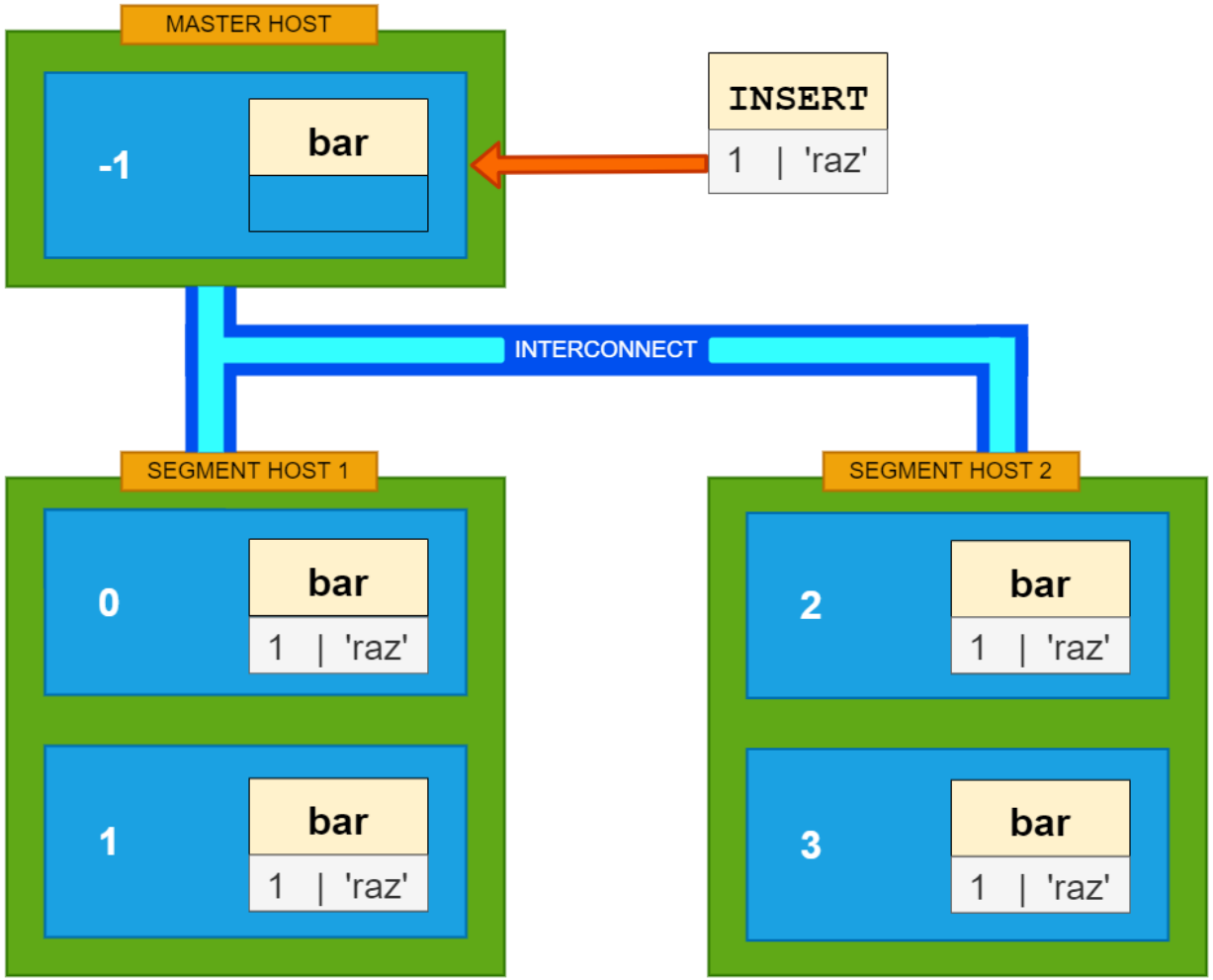


# Реплицированные таблицы

```
adb=# CREATE TABLE bar (a int, b text)
DISTRIBUTED REPLICATED;
```

```
adb=# insert into bar values (1,'raz');
```

Данные копируются на каждый праймари-сегмент целиком.



# Нюансы распределения

Путь к успеху – равномерное распределение данных по сегментам, отсутствие перекоса (skew).

Не выбирайте для ключей дистрибуции поля, записи в которых распределены сильно неравномерно:

- Не выбирайте даты.
- Не выбирайте поля, где может быть большое число значений NULL.
- Не выбирайте поля, в последующем распределении которых вы не уверены.

Таблицу можно перераспределить:

```
ALTER TABLE "foo" SET DISTRIBUTED BY (a);
```

Перераспределить таблицу, не меняя ключ:

```
ALTER TABLE "foo" SET WITH (REORGANIZE=TRUE) DISTRIBUTED BY (a);
```

Значение ключа дистрибуции у строки можно менять.

# Перекос в хранении данных

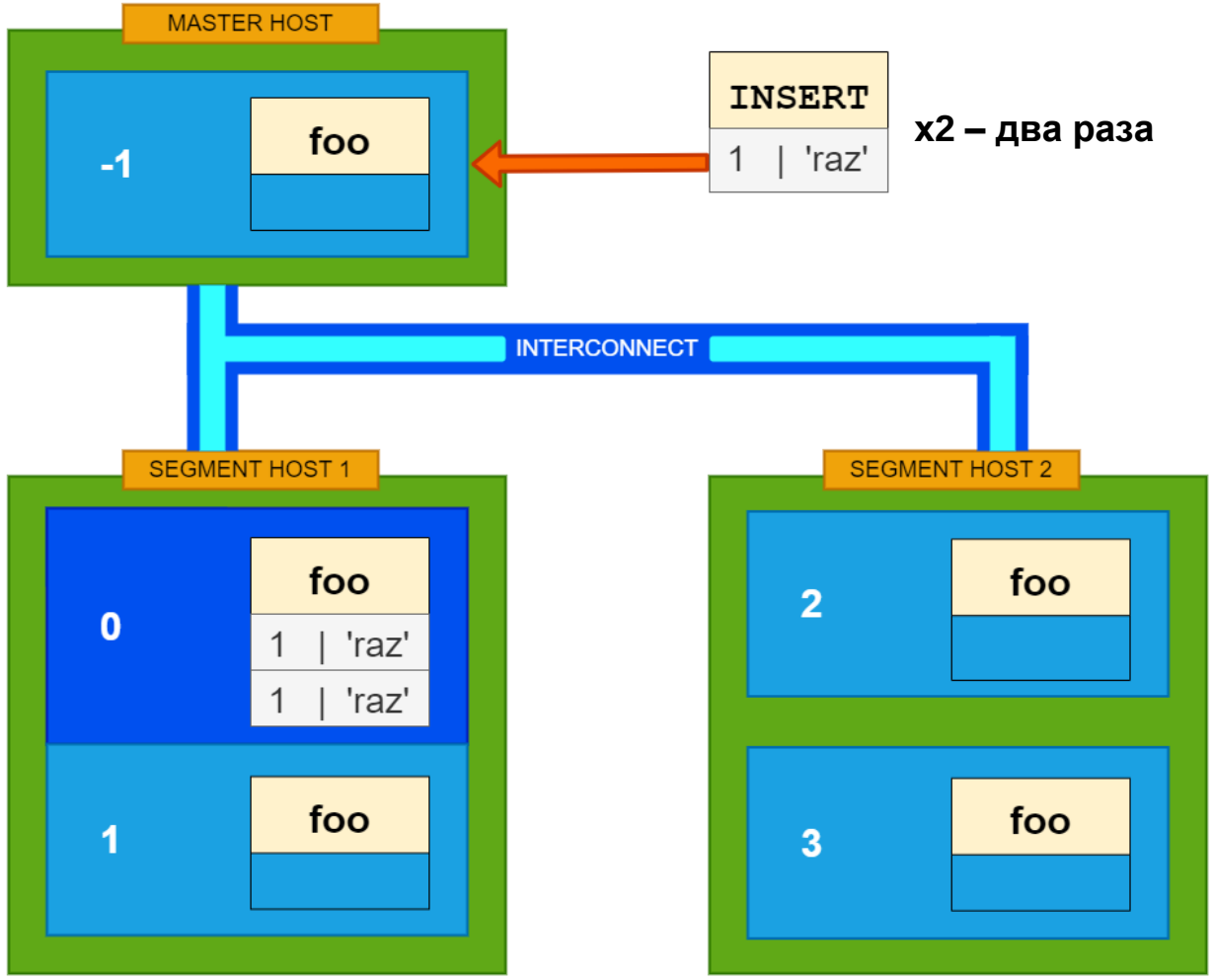
```
adb=# CREATE TABLE foo (a int, b text) DISTRIBUTED BY (a);
CREATE TABLE

adb=# insert into foo values (1,'raz'),(1,'raz');
INSERT 0 2

adb=# select gp_segment_id,* from foo;
 gp_segment_id | a |  b
---------------+---+-----
             0 | 1 | raz
             0 | 1 | raz
```

# Перекос в хранении данных

Избегайте перекоса!

При неверно выбранном ключе распределения на некоторые сегменты может попадать больше данных, чем на остальные.

Работа кластера в таком случае замедляется и могут быть проблемы из-за нехватки памяти у загруженных сегментов.



# Почему не использовать RANDOMLY всегда?

У распределения по ключу есть преимущества: джойны по ключам распределения – самые быстрые, так как каждый сегмент имеет у себя все строки, которые должны соединиться.

Джойны не по ключам распределения приводят к необходимости на время запроса перераспределить в оперативной памяти данные таблицы. В плане запроса возникает операция MOTION:

- BROADCAST MOTION – репликация малой таблицы по всем сегментам кластера. Эта операция не должна встречаться на проде.
- REDISTRIBUTE MOTION – перераспределение таблицы в памяти или в спилл-файлах. Избежать полностью во всех случаях невозможно, но старайтесь минимизировать. Кроме повышенной нагрузки есть опасность перегрузить некоторые сегменты, направив на них больше данных, чем на остальные. Это происходит, если ключ джойна плохо подходит для равномерного распределения.

Для оптимального использования ресурсов кластера:

- Используйте одинаковые типы данных в полях джойна.
- Следите за ключами джойнов, чтобы не вызвать динамический перекос.
- REPLICATED таблицы могут быть полезны для небольших таблиц-справочников, так как устраняют любой MOTION при джойнах.

# Таблицы распределены по ключу джойна

```
adb=# CREATE TABLE bar (a int, b text)
adb-# DISTRIBUTED BY (a);
CREATE TABLE
adb=#
adb=# insert into bar values (1,'raz'),(2,'dva');
INSERT 0 2
adb=# explain select * from foo f join bar b on f.a=b.a;
                                    QUERY PLAN
-------------------------------------------------------------------------------
 Gather Motion 8:1  (slice1; segments: 8)  (cost=0.00..862.00 rows=2 width=16)
   ->  Hash Join  (cost=0.00..862.00 rows=1 width=16)
         Hash Cond: foo.a = bar.a
         ->  Table Scan on foo  (cost=0.00..431.00 rows=1 width=8)
         ->  Hash  (cost=431.00..431.00 rows=1 width=8)
               ->  Table Scan on bar  (cost=0.00..431.00 rows=1 width=8)
```

# Одна таблица распределена не по ключу джойна

```
adb=# CREATE TABLE bar (a int, b text)
adb-# DISTRIBUTED BY (b);
CREATE TABLE
adb=# insert into bar values (1,'raz'),(2,'dva');
INSERT 0 2
adb=# explain select * from foo f join bar b on f.a=b.a;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Gather Motion 8:1  (slice2; segments: 8)  (cost=0.00..862.00 rows=2 width=16)
   ->  Hash Join  (cost=0.00..862.00 rows=1 width=16)
         Hash Cond: foo.a = bar.a
         ->  Table Scan on foo  (cost=0.00..431.00 rows=1 width=8)
         ->  Hash  (cost=431.00..431.00 rows=1 width=8)
               ->  Redistribute Motion 8:8  (slice1; segments: 8)  (cost=0.00..431.00 rows=1 width=8)
                     Hash Key: bar.a
                     ->  Table Scan on bar  (cost=0.00..431.00 rows=1 width=8)
```

# Резюме

- Все таблицы в ADB распределены или реплицированы.
- Есть три политики распределения: по хешу выбранного ключа, рандомное распределение и полная репликация таблицы.
- Две основные задачи при распределении:

1. Избежать перекосов в объёме хранимых данных по сегментам.
2. Минимизировать передачу данных между сегментами при выполнении запросов.

- Ключ распределения выбирается также таким образом, чтобы он мог использоваться в JOIN-ах.
- Если подходящий ключ распределения выбрать невозможно, подойдет рандомное распределение, которое эффективно решает первую задачу.
- Реплицированные таблицы убирают перемещение данных между сегментами, но занимают больше места и медленнее заполняются и обновляются.

# Лабораторная работа. Создание таблиц (1 часть)

Создайте таблицу `table1` со следующими параметрами:

- Поля: `id1 int, id2 int, gen1 text, gen2 text`.
- Первичным ключом сделайте поля `id1, id2, gen1`.
- Ключом распределения сделайте поле `id1`.

Ответьте на вопросы:

- Какого типа может быть таблица?
- Какая компрессия может использоваться в таблице?

Создайте таблицу `table2` со следующими параметрами:

- Возьмите набор полей `table1` с помощью директивы `LIKE`.
- Храните таблицу колоночно, сожмите таблицу с помощью `ZSTD` уровня `1`;
- Распределите таблицу по полю `id2`.

# Лабораторная работа. создание таблиц (2 часть)

Сгенерируйте данные и вставьте их в обе таблицы:

```
insert into table1 select gen,gen, gen::text || 'text1', gen::text || 'text2' from
generate_series(1,200000) gen;
insert into table2 select gen,gen, gen::text || 'text1', gen::text || 'text2' from
generate_series(1,400000) gen;
```

С помощью директивы `EXPLAIN` просмотрите план соединения таблиц `table1` и `table2` по ключу `id1`.

Оптимизируйте ситуацию, убрав `REDISTRIBUTE MOTION`, не изменяя сам запрос.

ARENADATA

Корпоративная платформа хранения
и обработки больших данных

# Партиционированные таблицы в ADB

# Партиционирование: общие сведения

- Партиционирование – способ увеличения производительности запросов за счёт разделения таблицы на подтаблицы и выборочного сканирования некоторых из них на основе условий в блоке `WHERE`.
- `Partition elimination` – механизм, уменьшает количество партиций, которые сканируются при выполнении запроса.
- Ключ партиционирования – одно поле в таблице, по которому определяется принадлежность записи к той или иной партиции (в случае многоуровнего партиционирования – набор полей).
- Варианты партиционирования: `RANGE` или `LIST`.
- В случае `RANGE` можно указывать партиции вручную или же указать интервал.
- Партиционирование может быть многоуровневым.
- Партиционированная таблица – сложный объект из набора таблиц. Создаётся отдельно, не может быть создана путем модификации обычной таблицы. Используются наследование (`INHERITS`) и констренты (`CHECK`).

# Партиционирование: нюансы использования

- На каждый уровень доступно максимально в районе 32 тысяч партиций.
- Не рекомендуется создавать более 1000 партиций на таблицу.
- Ключи для `PK` или `UNIQUE` должны содержать ключ партиционирования.
- `DEFAULT PARTITION` – для строк-потеряшек. Сканируется всегда. Не является обязательной.
- Таблицы, у которых ключ распределения `DISTRIBUTED REPLICATED` не могут быть партиционированны.
- Загрузка данных в партиционированные таблицы неэффективна. Поэтому следует подготавливать новый слепок данных для партиции во обычной таблице и затем делать замену партиции на неё при помощи `EXCHANGE PARTITION`.
- `Partition Elimination` работает с: `=, < , <= , >, >= , <>.`
- `Partition Elimination` работает с `STABLE` и `IMMUTABLE` функциями, но не работает с `VOLATILE`.
- Апдейтить поля, по которым партиционирована таблица, в ADB можно.
- Только одно поле в качестве ключа на один уровень. Составные ключи на уровне недопустимы.

# Партиционирование: варианты создания

**1. RANGE INTERVAL**

```
CREATE TABLE sales (id int, date date, amt
decimal(10,2)) DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
(START (date '2016-01-01') INCLUSIVE
END (date '2017-01-01') EXCLUSIVE
EVERY (INTERVAL '1 month'));
```

**2. RANGE INTERVAL by INT**

```
CREATE TABLE rank (id int, rank int, year int,
gender char(1), count int)
DISTRIBUTED BY (id)
PARTITION BY RANGE (year)
(START (2006) END (2016) EVERY (1),
DEFAULT PARTITION extra);
```

**3. RANGE INDIVIDUALLY**

```
CREATE TABLE sales (id int, date date, amt
decimal(10,2))
DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
(PARTITION Jan16 START (date '2016-01-01') INCLUSIVE,
PARTITION Feb16 START (date '2016-02-01') INCLUSIVE,
PARTITION Mar16 START (date '2016-03-01') INCLUSIVE,
PARTITION Apr16 START (date '2016-04-01') INCLUSIVE,
PARTITION May16 START (date '2016-05-01') INCLUSIVE,
PARTITION Jun16 START (date '2016-06-01') INCLUSIVE
END (date '2016-07-01') EXCLUSIVE);
```

**4. LIST**

```
CREATE TABLE rank (id int, rank int, year int,
gender
char(1), count int )
DISTRIBUTED BY (id)
PARTITION BY LIST (gender)
( PARTITION girls VALUES ('F'),
PARTITION boys VALUES ('M'),
DEFAULT PARTITION other);
```

# Партиционирование: варианты создания

**1. RANGE INTERVAL**

```
CREATE TABLE sales (id int, date date, amt
decimal(10,2)) DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
(START (date '2016-01-01') INCLUSIVE
END (date '2017-01-01') EXCLUSIVE
EVERY (INTERVAL '1 month'));
```

**2. RANGE INTERVAL by INT**

```
CREATE TABLE rank (id int, rank int, year int,
gender char(1), count int)
DISTRIBUTED BY (id)
PARTITION BY RANGE (year)
(START (2006) END (2016) EVERY (1),
DEFAULT PARTITION extra);
```

**3. RANGE INDIVIDUALLY**

```
CREATE TABLE sales (id int, date date, amt
decimal(10,2))
DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
(PARTITION Jan16 START (date '2016-01-01') INCLUSIVE,
PARTITION Feb16 START (date '2016-02-01') INCLUSIVE,
PARTITION Mar16 START (date '2016-03-01') INCLUSIVE,
PARTITION Apr16 START (date '2016-04-01') INCLUSIVE,
PARTITION May16 START (date '2016-05-01') INCLUSIVE,
PARTITION Jun16 START (date '2016-06-01') INCLUSIVE
END (date '2016-07-01') EXCLUSIVE);
```

**4. LIST**

```
CREATE TABLE rank (id int, rank int, year int,
gender
char(1), count int )
DISTRIBUTED BY (id)
PARTITION BY LIST (gender)
( PARTITION girls VALUES ('F'),
PARTITION boys VALUES ('M'),
DEFAULT PARTITION other);
```

# Партиционирование: варианты создания

### 1. RANGE INTERVAL

```
CREATE TABLE sales (id int, date date, amt
decimal(10,2)) DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
(START (date '2016-01-01') INCLUSIVE
END (date '2017-01-01') EXCLUSIVE
EVERY (INTERVAL '1 month'));
```

### 2. RANGE INTERVAL by INT

```
CREATE TABLE rank (id int, rank int, year int,
gender char(1), count int)
DISTRIBUTED BY (id)
PARTITION BY RANGE (year)
(START (2006) END (2016) EVERY (1),
DEFAULT PARTITION extra);
```

### 3. RANGE INDIVIDUALLY

```
CREATE TABLE sales (id int, date date, amt
decimal(10,2))
DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
(PARTITION Jan16 START (date '2016-01-01') INCLUSIVE,
PARTITION Feb16 START (date '2016-02-01') INCLUSIVE,
PARTITION Mar16 START (date '2016-03-01') INCLUSIVE,
PARTITION Apr16 START (date '2016-04-01') INCLUSIVE,
PARTITION May16 START (date '2016-05-01') INCLUSIVE,
PARTITION Jun16 START (date '2016-06-01') INCLUSIVE
END (date '2016-07-01') EXCLUSIVE);
```

### 4. LIST

```
CREATE TABLE rank (id int, rank int, year int,
gender
char(1), count int )
DISTRIBUTED BY (id)
PARTITION BY LIST (gender)
( PARTITION girls VALUES ('F'),
PARTITION boys VALUES ('M'),
DEFAULT PARTITION other);
```

# Партиционирование: варианты создания

1. RANGE INTERVAL

```
CREATE TABLE sales (id int, date date, amt
decimal(10,2)) DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
(START (date '2016-01-01') INCLUSIVE
END (date '2017-01-01') EXCLUSIVE
EVERY (INTERVAL '1 month'));
```

2. RANGE INTERVAL by INT

```
CREATE TABLE rank (id int, rank int, year int,
gender char(1), count int)
DISTRIBUTED BY (id)
PARTITION BY RANGE (year)
(START (2006) END (2016) EVERY (1),
DEFAULT PARTITION extra);
```

**3. RANGE INDIVIDUALLY**

```
CREATE TABLE sales (id int, date date, amt
decimal(10,2))
DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
(PARTITION Jan16 START (date '2016-01-01') INCLUSIVE,
PARTITION Feb16 START (date '2016-02-01') INCLUSIVE,
PARTITION Mar16 START (date '2016-03-01') INCLUSIVE,
PARTITION Apr16 START (date '2016-04-01') INCLUSIVE,
PARTITION May16 START (date '2016-05-01') INCLUSIVE,
PARTITION Jun16 START (date '2016-06-01') INCLUSIVE
END (date '2016-07-01') EXCLUSIVE);
```

4. LIST

```
CREATE TABLE rank (id int, rank int, year int,
gender
char(1), count int )
DISTRIBUTED BY (id)
PARTITION BY LIST (gender)
( PARTITION girls VALUES ('F'),
PARTITION boys VALUES ('M'),
DEFAULT PARTITION other);
```

# Партиционирование: варианты создания

### 1. RANGE INTERVAL

```
CREATE TABLE sales (id int, date date, amt
decimal(10,2)) DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
(START (date '2016-01-01') INCLUSIVE
END (date '2017-01-01') EXCLUSIVE
EVERY (INTERVAL '1 month'));
```

### 2. RANGE INTERVAL by INT

```
CREATE TABLE rank (id int, rank int, year int,
gender char(1), count int)
DISTRIBUTED BY (id)
PARTITION BY RANGE (year)
(START (2006) END (2016) EVERY (1),
DEFAULT PARTITION extra);
```

### 3. RANGE INDIVIDUALLY

```
CREATE TABLE sales (id int, date date, amt
decimal(10,2))
DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
(PARTITION Jan16 START (date '2016-01-01') INCLUSIVE,
PARTITION Feb16 START (date '2016-02-01') INCLUSIVE,
PARTITION Mar16 START (date '2016-03-01') INCLUSIVE,
PARTITION Apr16 START (date '2016-04-01') INCLUSIVE,
PARTITION May16 START (date '2016-05-01') INCLUSIVE
WITH (appendonly=true),
PARTITION Jun16 START (date '2016-06-01') INCLUSIVE
END (date '2016-07-01') EXCLUSIVE);
```

### 4. LIST

```
CREATE TABLE rank (id int, rank int, year int,
gender
char(1), count int )
DISTRIBUTED BY (id)
PARTITION BY LIST (gender)
( PARTITION girls VALUES ('F'),
PARTITION boys VALUES ('M'),
DEFAULT PARTITION other);
```

# Партиционирование: варианты создания

### 1. RANGE INTERVAL

```
CREATE TABLE sales (id int, date date, amt
decimal(10,2)) DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
(START (date '2016-01-01') INCLUSIVE
END (date '2017-01-01') EXCLUSIVE
EVERY (INTERVAL '1 month'));
```

### 2. RANGE INTERVAL by INT

```
CREATE TABLE rank (id int, rank int, year int,
gender char(1), count int)
DISTRIBUTED BY (id)
PARTITION BY RANGE (year)
(START (2006) END (2016) EVERY (1),
DEFAULT PARTITION extra);
```

### 3. RANGE INDIVIDUALLY

```
CREATE TABLE sales (id int, date date, amt
decimal(10,2))
DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
(PARTITION Jan16 START (date '2016-01-01') INCLUSIVE,
PARTITION Feb16 START (date '2016-02-01') INCLUSIVE,
PARTITION Mar16 START (date '2016-03-01') INCLUSIVE,
PARTITION Apr16 START (date '2016-04-01') INCLUSIVE,
PARTITION May16 START (date '2016-05-01') INCLUSIVE,
PARTITION Jun16 START (date '2016-06-01') INCLUSIVE
END (date '2016-07-01') EXCLUSIVE);
```

### 4. LIST

```
CREATE TABLE rank (id int, rank int, year int,
gender
char(1), count int )
DISTRIBUTED BY (id)
PARTITION BY LIST (gender)
( PARTITION girls VALUES ('F'),
PARTITION boys VALUES ('M'),
DEFAULT PARTITION other);
```

# Партиционирование: вариант с двумя уровнями

```
CREATE TABLE sales (trans_id int, date date, amount decimal(9,2), region text)
DISTRIBUTED BY (trans_id)
PARTITION BY RANGE (date)
SUBPARTITION BY LIST (region)
  SUBPARTITION TEMPLATE (
    SUBPARTITION usa VALUES ('usa'),
    SUBPARTITION asia VALUES ('asia'),
    SUBPARTITION europe VALUES ('europe'),
    DEFAULT SUBPARTITION other_regions)
(START (date '2011-01-01') INCLUSIVE END (date '2012-01-01') EXCLUSIVE EVERY (INTERVAL '1
month'), DEFAULT PARTITION outlying_dates);
```

**Количество партиций перемножается, следите за их общим числом!**

ARENADATA

# Партиционирование: модификация таблиц

Представление `pg_catalog.pg_partitions` подскажет текущую схему партиционирования по таблице.

Имена партиций наследуются, в т.ч. при переименовании: `<parentname>_<level>_prt_<partition_name>`.

Операции по изменению партиций выполняются при помощи запроса ALTER к главной партиции. Запрос:

```
ALTER TABLE sales RENAME PARTITION FOR ('2016-01-01') TO jan16;
```

задаст имя дочерней партиции следующего вида: `sales_1_prt_jan16` (т.е. это переименование партиции).

Можно распределять дочерние партиции `RANDOMLY` или по ключу основной таблицы. По иному ключу нельзя.

При добавлении новой партиции без указания типа хранения, создаётся HEAP-партиция.

Обращаться к дочерней партиции при модификации можно:

По значению ключа партиционирования:

```
ALTER TABLE sales RENAME PARTITION FOR ('2016-01-01') TO jan16;
```

По имени партиции (не полному имени таблици-партиции, а по её постфиксу):

```
ALTER TABLE sales RENAME PARTITION january16 TO jan16;
```

По значению `RANK:`

```
ALTER TABLE sales RENAME PARTITION FOR (RANK(1)) TO jan16;
```

# Партиционирование: разделение партиции

Разделение партиции надвое:

```
ALTER TABLE sales SPLIT PARTITION FOR ('2017-01-01')
AT ('2017-01-16') INTO (PARTITION jan171to15, PARTITION jan1716to31);
```

Отделение новой партиции от дефолтной:

```
ALTER TABLE sales SPLIT DEFAULT PARTITION
START ('2017-01-01') INCLUSIVE
END ('2017-02-01') EXCLUSIVE
INTO (PARTITION jan17, default partition);
```

Разделить партицию за раз можно только на две. Объединить партиции нельзя, `MERGE` в ADB нет вообще!

# Добавление партиции

- Партицию можно добавлять только «в пустое место», пересечения недопустимы.
- Если есть DEFAULT PARTITION, добавить партицию ADB не позволит.

```
ALTER TABLE sales ADD PARTITION START (date '2017-02-01') INCLUSIVE  END (date
'2017-03-01') EXCLUSIVE;

ALTER TABLE sales ADD DEFAULT PARTITION other;

ALTER TABLE sales DROP PARTITION FOR (RANK(1));

ALTER TABLE sales TRUNCATE PARTITION FOR (RANK(1));
```

# Замена партиции

- Позволяет заменить партицию на таблицу с такой же структурой.
- Позволяет создавать полиморфные объекты за счет замены на таблицы с иными настройками хранения.
- Позволяет менять партиции на внешние таблицы (ограничение: только таблицы на чтение).
- При замене происходит обмен ссылками на таблицы, они переименовываются.

Создаём таблицу и меняем на неё партицию:

```
CREATE TABLE jan12table (LIKE sales) WITH (appendonly=true);
ALTER TABLE sales EXCHANGE PARTITION jan12 WITH TABLE jan12table WITH VALIDATION;
```

Опции операции:

- `WITH VALIDATION` – проверит, что все записи в таблице соответствуют отрезку ключа партиционирования. При несовпадении операция завершится с ошибкой.
- `WITHOUT VALIDATION` – ничего не проверяется. Используется при замене партиции на внешнюю таблицу.

# Лабораторная работа. Переименование

1. Создайте две одинаковые таблицы foo1 и foo2 с атрибутами id int, state text.
2. Добавьте по одной различающейся строке в каждую.
3. Создайте VIEW на первую таблицу (`select * from foo1`), выполните запрос SELECT к представлению.
4. Выполните в одной транзакции переименование таблиц друг в друга;

```
alter table foo1 rename to foo_tmp;
alter table foo2 rename to foo1;
alter table foo_tmp rename to foo2;
```

5. Выполните запрос SELECT к представлению, изменились ли результаты?
6. Посмотрите DDL представления.

# Замена партиции – альтернатива переименованию

`RENAME TABLE`: после переименования таблицы зависимые объекты, например `VIEW`, продолжат ссылаться на исходную таблицу.

- В случае использования подхода с подготовкой данных в стейджинговой таблице, а затем ее переименованием в целевую, необходимо пересоздавать и представление.
- Вместо этого можно использовать подход с `exchange partition`. Справочник, например, можно сделать в виде одной дефолтной партиции и подменять её на актуальный слепок данных заменой на таблицу.

# Лабораторная работа. Партиционирование

Создайте партиционированную таблицу `table3 (dttm timestamp, id int)`:

- С 1-го января 2016 включительно по 1-ое января 2017 храните данные колоночно со сжатием `ZSTD` уровня 5;
- С 1-го января 2017 включительно по 1-ое января 2018 храните данные колоночно со сжатием `ZSTD` уровня 1;
- С 1-го января 2018 включительно по 1-го января 2019 храните данные в `HEAP`-таблице.
- Предусмотрите дефолтную партицию.

Добавьте партицию для периода с 1-го января 2015 включительно по 1-ое января 2016. Используйте сжатие `ZSTD` уровня 19.

Переименуйте созданную партицию в `old_one`.

Создайте партиционированную таблицу `table4 (dttm timestamp, id int)`:

- С 1-го января 2015 включительно по 1-ое января 2016.
- Переместите old_one из table3 в table4. В table3 не должно остаться партиции за 2015 год.

ARENADATA

Корпоративная платформа хранения
и обработки больших данных

# Типы данных

# Рекомендации по типам данных

- Используйте наименьшие возможные типы данных для полей (INT вместо BIGINT, TEXT вместо CHAR(n)). Не храните целые числа в NUMERIC, если они умещаются в стандартные целочисленные.
- По возможности используйте специализированные типы данных (INET, CIDR, JSON, JSONB, MACADDR…).
- Используйте одинаковые типы данных в полях, по которым выполняются JOIN-операции.
- Создавайте свои типы данных (типы-справочники – ENUM – могут дать огромный прирост производительности).

# Численные типы

| | | | | |
|---|---|---|---|---|
| decimal [ (p, s) ]1 | numeric [ (p, s) ] | variable | no limit | user-specified precision, exact |
| double precision | float8<br>float | 8 bytes | 15 decimal digits precision | variable-precision, inexact |
| integer | int, int4 | 4 bytes | -2147483648 to +2147483647 | usual choice for integer |
| money | | 8 bytes | -92233720368547758.08 to +92233720368547758.07 | currency amount |
| real | float4 | 4 bytes | 6 decimal digits precision | variable-precision, inexact |
| serial | serial4 | 4 bytes | 1 to 2147483647 | autoincrementing integer |
| smallint | int2 | 2 bytes | -32768 to +32767 | small range integer |

# Сопоставление с Oracle

| Oracle | Greenplum | Комментарий |
|---|---|---|
| NUMBER | NUMERIC<br><br>DECIMAL | Не во всех случаях эффективно. Следует подобрать подходящий тип, соответствующий фактическому содержимому поля (см. ниже) |
| NUMBER (p, s) | NUMERIC (p, s)<br><br>DECIMAL (p, s)<br><br>REAL<br><br>FLOAT<br><br>MONEY | Типы REAL и FLOAT имеют ограничения:<br><br>1. REAL: с точностью до 6 десятичных знаков.<br><br>2. FLOAT: с точностью до 15 десятичных знаков.<br><br>Numeric в Greenplum в целом идентичен Number с переменными диапазоном и точностью, поэтому может использоваться для любых числовых полей, но иногда более предпочтительны целочисленные поля (см. ниже) и числа с плавающей запятой |
| NUMBER (p), где p ≤ 4 | SMALLINT (2 байта) | |
| NUMBER (p), где 4< p ≤ 9 | INTEGER – (4 байта) | |

# Сопоставление с Oracle

| Oracle | Greenplum | Комментарий |
|---|---|---|
| NUMBER (p), где 9< p ≤ 18 | BIGINT (8 байтов) | |
| NUMBER (p), где 18 < p | NUMERIC (p) | |
| FLOAT | FLOAT | |
| CHAR (n) | CHAR (n) | Стоит помнить, что объём записей символьного типа в Oracle может задаваться как в знаках, так и в байтах, а в Greenplum только в знаках |
| NCHAR (n) | CHAR (n) | |
| VARCHAR2(n) | VARCHAR (n) | |

# Сопоставление с Oracle

| Oracle | Greenplum | Комментарий |
|---|---|---|
| NVARCHAR2(n) | VARCHAR (n) | |
| CLOB<br>NCLOB<br>LONG<br>LONG RAW | TEXT | |
| XMLTYPE | XML | |
| BFILE ( > 1 ГБ) | Large Objects (до 2 ГБ) | |
| RAW BFILE( < 1 ГБ) | BYTEA | |

# Сопоставление с Oracle

| Oracle | Greenplum | Комментарий |
|---|---|---|
| DATE (включает время до секунд) | DATE<br>или TIMESTAMP | Выбор между типами DATE и TIMESTAMP зависит от фактического содержимого поля.<br>- Если поле содержит время -> TIMESTAMP<br>- Если поле не содержит время -> DATE<br>Чтобы избавиться от фракций секунд, рекомендуется использовать тип TIMESTAMP (0) |
| TIMESTAMP with TIME ZONE | TIMESTAMPTZ | |
| TIMESTAMP<br>TIMESTAMP (n) | TIMESTAMP | |
| INTERVAL | INTERVAL/TIME – | |
| MDSYS.SDO_GEOMETRY | geometry (PostGIS) | |

# DDL в Oracle

*Oracle:*

```
CREATE TABLE employees
    ( employee_id    NUMBER(6)
    , first_name     VARCHAR2(20)
    , last_name      VARCHAR2(25)
    , email          VARCHAR2(25)
    , hire_date      DATE
    , job_id         NUMBER(10)
    , salary         NUMBER(8,2)
    ) ;
```

# DDL в Greenplum

*Greenplum*

```
CREATE TABLE employees
    ( employee_id    INTEGER
    , first_name     VARCHAR(20)
    , last_name      VARCHAR(25)
    , email          VARCHAR(25)
    , hire_date      DATE
    , job_id         BIGINT
    , salary         MONEY
    ) ;
```



ARENADATA

Корпоративная платформа хранения
и обработки больших данных

# Работа с JSON

# Тип JSON

Поддерживается два типа данных: json и jsonb

JSON:

- Хранится в таблице как обычный текст с сохранением структуры и пробелов;
- Может содержать дублирующие ключи на одном уровне. Если таковые есть, то последнее значение будет более актуальнее.

JSONB:

- Пробелы не сохраняются;
- При загрузке значений дублирующих ключей на одном уровне, сохраняется только последнее значение;
- Поддерживает индексы.

# Индексы JSON

Поддерживается два типа индексов: GIN и BTREE индексы

GIN:

- Применяются с операторами  @>, ?, ?& и ?|
- Можно применять два различных оператора класса:  jsonb_ops (по умолчанию) и jsonb_path_ops
- Оператор класса jsonb_path_ops поддерживает только оператор @>

BTREE:

- Полезен только тогда, когда важно проверить равенство документов JSON.

# Операторы JSON и JSONB

Список поддерживаемых операторов для JSON и JSONB:

- **->** Получить значение по номеру элемента (int) или по ключу (text) в виде типа json или jsonb
- **->>** Получить значение по номеру элемента (int) или по ключу (text) в виде типа text
- **#>** Получить список JSON объектов по заданному пути в виде типа данных json или jsonb
- **#>>** Получить список JSON объектов по заданному пути в виде типа данных text

# Операторы JSONB

Список поддерживаемых операторов только типом JSONB:

- **@>** Левое значение JSON содержит путь/значение JSON справа
- **@<** Путь/значение JSON слева содержится в правом значении JSON
- **?** Присутствует ли строка в качестве ключа в значении JSON
- **?|** Присутствуют какие-либо *строки* массива в качестве ключей
- **?&** Все строки массива присутствуют в качестве ключей

Операторы сравнения:

<, >, <=, >=, =, <> или !=

# Функции создания обработки JSON

| | |
|---|---|
| to_json(anyelement) | Возвращает значение в виде json. |
| json_build_array(VARIADIC "any") | Формирует массив JSON (возможно, разнородный) из переменного списка аргументов. |
| json_build_object(VARIADIC "any") | Формирует объект JSON из переменного списка аргументов. По соглашению в этом списке перечисляются по очереди ключи и значения. |
| json_object(text[]) | Формирует объект JSON из текстового массива. Этот массив должен иметь либо одну размерность с чётным числом элементов (в этом случае они воспринимаются как чередующиеся ключи/значения), либо две размерности и при этом каждый внутренний массив содержит ровно два элемента, которые воспринимаются как пара ключ/значение. |
| json_object(keys text[], values text[]) | Формирует объект JSON, принимая ключи и значения по парам из двух отдельных массивов. |

# Функции для обработки JSON

| | | |
|---|---|---|
| json_array_length(json)<br>jsonb_array_length(jsonb) | int | Возвращает число элементов во внешнем массиве JSON. |
| json_each(json)<br><br>jsonb_each(jsonb) | setof key text, value json или jsonb | Разворачивает внешний объект JSON в набор пар ключ/значение (key/value). |
| json_extract_path(from_json json, VARIADIC path_elems text[])<br><br>jsonb_extract_path(from_json jsonb, VARIADIC path_elems text[]) | json<br><br>jsonb | Возвращает значение JSON по пути, заданному элементами пути (path_elems) (равнозначно оператору #> operator). |
| json_object_keys(json)<br><br>jsonb_object_keys(jsonb) | setof text | Возвращает набор ключей во внешнем объекте JSON. |
| json_array_elements(json)<br><br>jsonb_array_elements(jsonb) | setof json<br><br>setof jsonb | Разворачивает массив JSON в набор значений JSON. |

# Агрегатные функции JSON

| Функция | Тип входных параметров | Тип возвращаемых данных | Описание |
|---|---|---|---|
| json_agg(record) | any | json | Агрегирует значения, включая NULL, в виде массива JSON |
| json_object_agg(name, value) | ("any", "any") | json | Агрегирует name/value значения в набор значений JSON |

ARENADATA

Корпоративная платформа хранения
и обработки больших данных

# Работа с XML

# Работа с XML

- Для хранения XML-данных используется тип данных xml.
- При использовании типа данных XML значение проверяется на допустимость по правилам XML.
- Поддерживаются как правильно оформленные «документы», в соответствии со стандартом XML, так и фрагменты документов.
- Для работы с XML используются специальные функции.

# Работа с XML. Преобразование в тип

- Для получения типа XML из текстовой строки, используйте функцию **xmlparse**:

  *XMLPARSE ( { DOCUMENT | CONTENT } value)*

  Пример*:*

*XMLPARSE (DOCUMENT '<?xml version="1.0"?><book><title>Manual</title><chapter>...</chapter></book>')*

*XMLPARSE (CONTENT 'abc<foo>bar</foo><bar>foo</bar>')*

  Другие варианты:

  *xml  ' <foo>bar</foo>'*

  *'<foo>bar</foo>'::xml*

- Для преобразования типа XML в другой тип данных используется функция **xmlserialize** :

  *xmlserialize ( { DOCUMENT | CONTENT } value AS type )*

# Работа с XML. Обработка

Для обработки значений типа xml существуют функции **xpath** и **xpath_exists**.

- Функция xpath вычисляет выражение XPath (аргумент *xpath* типа text) для заданного *xml*. Она возвращает массив XML-значений с набором узлов, полученных при вычислении выражения XPath:

  xpath(*xpath*, *xml* [, *nsarray*])

- Функция xpath_exists возвращает только одно логическое значение, показывающее, есть ли такие узлы:

  xpath_exists(xpath, xml [, nsarray])

# Работа с XML. Примеры обработки

Функция **xpath**:

```
SELECT xpath('/my:a/text()', '<my:a xmlns:my="http://example.com">test</my:a>',
        ARRAY[ARRAY['my', 'http://example.com']]);
 xpath
--------
 {test}
(1 row)
```

Функция **xpath_exists:**

```
SELECT xpath_exists('/my:a/text()', '<my:a xmlns:my="http://example.com">test</my:a>',
            ARRAY[ARRAY['my', 'http://example.com']]);
 xpath_exists
  --------------
 t
(1 row)
```

# Работа с XML. Преобразование объектов в XML

Функции, которые отображают содержимое реляционных таблиц в формате XML:

- table_to_xml(tbl regclass, nulls boolean, tableforest boolean, targetns text)
- query_to_xml(query *text*, nulls boolean, tableforest boolean, targetns text)
- cursor_to_xml(cursor refcursor, count int, nulls boolean, tableforest boolean, targetns text)

tbl – имя таблицы;

query – запрос, результат которого необходимо выгрузить в xml;

cursor – имя курсора; count – кол-во строк (fetch);

nulls – выгружать ли значения с NULL (<columnname xsi:nil="true"/>);

tableforest – вид выгружаемого XML. Если значение true, то каждый фрагмент будет содержать имя таблицы;

targetns – XML namespace;

Аналогичные функции для отображение схем и баз данных в формате XML:

- schema_to_xml(schema name, nulls boolean, tableforest boolean, targetns text)
- database_to_xml(nulls boolean, tableforest boolean, targetns text)

# Работа с XML. Преобразование вXML Schema

Функции, которые выдают документы XML Schema:

- table_to_xmlschema(tbl regclass, nulls boolean, tableforest boolean, targetns text)
- query_to_xmlschema(query text, nulls boolean, tableforest boolean, targetns text)
- cursor_to_xmlschema(cursor refcursor, nulls boolean, tableforest boolean, targetns text)

Функции,  которые отображают данные в формате XML и соответствующую XML-схему в одном документе:

- table_to_xml_and_xmlschema(tbl regclass, nulls boolean, tableforest boolean,  targetns text)
- query_to_xml_and_xmlschema(query text, nulls boolean, tableforest boolean,  targetns text)

Аналогичные функции существуют для отображение схем и баз данных в формате XML.

# Работа с XML. Функции для работы с XML

Для получения XML-контента из данных SQL существует целый набор функций и функциональных выражений:

- `xmlcomment(text)` - создаёт XML-значение, содержащее XML-комментарий с заданным текстом.
- `xmlconcat(xml[, ...])` - объединяет несколько XML-значений и выдаёт в результате один фрагмент XML-контента;=.
- `xmlelement(name name [, xmlattributes(value [AS attname] [, ... ])] [, content, ...])` - создаёт XML-элемент с заданным именем, атрибутами и содержимым.
- `xmlforest(content [AS name] [, ...])` - создаёт последовательность XML-элементов с заданными именами и содержимым.
- `xmlpi(name` *`target`* `[,` *`content`*`])` - создаёт инструкцию обработки XML.
- `xmlroot(`*`xml`*`, version` *`text`* `| no value [, standalone yes|no|no value])` - изменяет свойства корневого узла XML-значения.
- *`xml`* `IS DOCUMENT` - возвращает true, если аргумент представляет собой правильный XML-документ, false в противном случае.

# Конец второй части