Оглавление

[БД, СУБД, РМД, SQL 5](#_Toc48513874)

[Операторы 6](#_Toc48513875)

[DDL (CREATE, ALTER, DROP) 6](#_Toc48513876)

[DML (SELECT, INSERT, UPDATE, DELETE) 7](#_Toc48513877)

[DCL (GRANT, REVOKE, DENY) 9](#_Toc48513878)

[TCL (COMMIT, ROLLBACK, SAVEPOINT) 9](#_Toc48513879)

[Типы данных 9](#_Toc48513880)

[TRUNCATE и DELETE 9](#_Toc48513881)

[NULL. Как проверить поле на NULL? 10](#_Toc48513882)

[JOIN 11](#_Toc48513883)

[Типы JOIN 11](#_Toc48513884)

[(INNER) JOIN 12](#_Toc48513885)

[LEFT (OUTER) JOIN 12](#_Toc48513886)

[RIGHT (OUTER) JOIN 13](#_Toc48513887)

[FULL (OUTER) JOIN 13](#_Toc48513888)

[CROSS JOIN (декартово произведение) 13](#_Toc48513889)

[JOIN и подзапросы 14](#_Toc48513890)

[Временная таблица 15](#_Toc48513891)

[Представление 15](#_Toc48513892)

[UNION 16](#_Toc48513893)

[HAVING 17](#_Toc48513894)

[HAVING и WHERE 17](#_Toc48513895)

[ORDER BY 18](#_Toc48513896)

[GROUP BY, DISTINCT 18](#_Toc48513897)

[GROUP BY 18](#_Toc48513898)

[GROUP BY и DISTINCT 19](#_Toc48513899)

[LIMIT 19](#_Toc48513900)

[EXISTS 19](#_Toc48513901)

[IN, BETWEEN, LIKE 19](#_Toc48513902)

[MERGE 20](#_Toc48513903)

[DATETIME и TIMESTAMP 21](#_Toc48513904)

[Агрегатные функции 21](#_Toc48513905)

[COUNT 21](#_Toc48513906)

[SUM 21](#_Toc48513907)

[AVG 21](#_Toc48513908)

[MAX 21](#_Toc48513909)

[MIN 21](#_Toc48513910)

[COUNT(\*) и COUNT({column}) 21](#_Toc48513911)

[Ограничения (constraints) на целостность данных 22](#_Toc48513912)

[PRIMARY KEY 22](#_Toc48513913)

[CHECK 22](#_Toc48513914)

[UNIQUE 22](#_Toc48513915)

[FOREIGN KEY 22](#_Toc48513916)

[PRIMARY и UNIQUE 23](#_Toc48513917)

[Ключи 23](#_Toc48513918)

[Суррогатный ключ 23](#_Toc48513919)

[Простой ключ 23](#_Toc48513920)

[Составной ключ (composite key) 23](#_Toc48513921)

[Потенциальный ключ (candidate key) 23](#_Toc48513922)

[Первичный ключ (primary key) 23](#_Toc48513923)

[Альтернативный ключ (alternate key) 23](#_Toc48513924)

[Внешний ключ (foreign key) 24](#_Toc48513925)

[Индекс (index) 24](#_Toc48513926)

[Типы индексов (*и*) 25](#_Toc48513927)

[Кластерные и некластерные индексы 30](#_Toc48513928)

[Индексировать данные, имеющие небольшое количество возможных значений? 31](#_Toc48513929)

[Полное сканирование выгоднее доступа по индексу? 31](#_Toc48513930)

[Создать индекс 32](#_Toc48513931)

[DELETE и TRUNCATE 32](#_Toc48513932)

[Хранимая процедура 32](#_Toc48513933)

[Триггер (trigger) 33](#_Toc48513934)

[Транзакция 34](#_Toc48513935)

[Основные свойства транзакции 34](#_Toc48513936)

[Уровни изолированности транзакций 35](#_Toc48513937)

[Проблемы при параллельном выполнении транзакции 36](#_Toc48513938)

[Нормализация и денормализация 36](#_Toc48513939)

[Нормализация 36](#_Toc48513940)

[Денормализация 36](#_Toc48513941)

[Нормальная форма 36](#_Toc48513942)

[PIVOT и UNPIVOT в Transact‑SQL 37](#_Toc48513943)

[INTERSECT, EXCEPT в Transact‑SQL 37](#_Toc48513944)

[Курсор 38](#_Toc48513945)

[Для каких числовых типов недопустимо использовать операции сложения/вычитания? 39](#_Toc48513946)

[Основные функции ранжирования в Transact‑SQL. 39](#_Toc48513947)

# БД, СУБД, РМД, SQL

База данных (БД) — организованный и адаптированный для обработки вычислительной системой набор информации.

Система управления базами данных (СУБД) — набор средств общего или специального назначения, обеспечивающий создание, доступ к материалам и управление БД.

Основные функции СУБД:  
∙ управление данными;  
∙ журнализация изменений данных;  
∙ резервное копирование и восстановление данных;  
∙ поддержка языка определения данных и манипулирования ими.

Реляционная модель данных (РМД) — логическая модель данных и прикладная теория построения реляционных баз данных.

Реляционная модель данных включает в себя следующие компоненты:  
∙ структурный аспект (данные представляют собой набор отношений);  
∙ аспект целостности (отношения отвечают определенным условиям целостности: уровня домена (типа данных), уровня отношения и уровня БД);  
∙ аспект обработки (манипулирования) (поддержка операторов манипулирования отношениями (реляционная алгебра, реляционное исчисление));  
∙ нормальная форма (свойство отношения в РМД, характеризующее его с точки зрения избыточности и определенное как совокупность требований, которым должно удовлетворять отношение).

SQL (structured query language, язык структурированных запросов) — формальный непроцедурный язык программирования, применяемый для создания, модификации и управления данными в произвольной реляционной базе данных (РБД), управляемой соответствующей системой управления базами данных (СУБД).

# Операторы

## **DDL (CREATE, ALTER, DROP)**

Операторы DDL (операторы определения данных, data definition language) работают с объектами.

Объекты (О):

|  |  |
| --- | --- |
| БД | DATABASE |
| таблица | TABLE |
| представление | VIEW |
| пользователь |  |
| ограничения |  |

∙ CREATE — создает О:

|  |  |
| --- | --- |
| 1. | CREATE DATABASE <имя\_БД>; |
| 1.  2.  3.  4.  5. | CREATE TABLE <имя\_БД>.<имя\_таблицы> (  <имя\_1-ого\_столбца> <тип\_данных>,  <имя\_2-ого\_столбца> <тип\_данных>,  ...  ); |

∙ ALTER — изменяет О:

|  |  |
| --- | --- |
| 1.  2.  3.  4. | ALTER TABLE <имя\_БД>.<имя\_таблицы> ADD (  <имя\_нового\_столбца> <тип\_данных>,  ...  ); |

∙ DROP — удаляет О:

|  |  |
| --- | --- |
| 1. | DROP TABLE <имя\_БД>.<имя\_таблицы>; |

Дополнительно:

∙ TRUNCATE — удаляет содержимое таблицы.

|  |  |
| --- | --- |
| 1. | TRUNCATE TABLE <имя\_БД>.<имя\_таблицы; |

## DML (SELECT, INSERT, UPDATE, DELETE)

DML (операторы манипуляции данными , data manipulation language) работают с данными (Д).

∙ SELECT — возвращает набор данных (выборку) из базы данных, удовлетворяющих заданному условию.

Синтаксис:

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10. | SELECT  [DISTINCT | DISTINCTROW | ALL]  <выражение>, ...  FROM <имя\_БД>.<имя\_таблицы>  [WHERE where\_definition]  [GROUP BY {unsigned\_integer | column  | formula}]  [HAVING where\_definition]  [ORDER BY {unsigned\_integer | column  | formula} [ASC | DESC], ...] |

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10. | SELECT  [DISTINCT | DISTINCTROW | ALL]  select\_expression,...  FROM table\_references  [WHERE where\_definition]  [GROUP BY {unsigned\_integer | column  | formula}]  [HAVING where\_definition]  [ORDER BY {unsigned\_integer | column  | formula} [ASC | DESC], ...] |

∙ INSERT — добавляет новые Д:

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5. | INSERT INTO <имя\_БД>.<имя\_таблицы>  [<имя\_столбца>, ...]  {VALUES (<значение\_столбца>, ...)}  | <выражение\_запроса>  | {DEFAULT VALUES} |
| 1.  2. | INSERT INTO Product  VALUES ('B', 1157, 'PC'); |
| 1.  2. | INSERT INTO Product (type, model, maker)  VALUES ('PC', 1157, 'B'); |

∙ UPDATE — изменяет существующие Д:

|  |  |
| --- | --- |
| 1.  2.  3.  4. | UPDATE <имя\_БД>.<имя\_таблицы>  SET {<имя\_столбца> = {<новое\_значение> | NULL |  DEFAULT}, ...}  [{WHERE <условие>}]; |
| 1.  2. | UPDATE Laptop  SET price = price\*0.9; |
| 1.  2.  3. | UPDATE Laptop  SET hd = ram/2  WHERE hd < 10; |

∙ DELETE — удаляет Д:

|  |  |
| --- | --- |
| 1.  2. | DELETE FROM <имя\_БД>.<имя\_таблицы>  [WHERE <условие>]; |
| 1.  2. | UPDATE Laptop  SET price = price\*0.9; |
| 1.  2.  3. | UPDATE Laptop  SET hd = ram/2  WHERE hd < 10; |

## DCL (GRANT, REVOKE, DENY)

DCL (операторы определения доступа к Д, data control language).

∙ GRANT — предоставляет пользователю (группе) разрешения на операции с объектом;

∙ REVOKE — отзывает ранее выданные разрешения;

∙ DENY — задает запрет (приоритетнее GRANT).

## TCL (COMMIT, ROLLBACK, SAVEPOINT)

TCL (операторы управления транзакциями (T), transaction control language):

∙ COMMIT — применяет T;

∙ ROLLBACK — откатывает изменения текущей T;

∙ SAVEPOINT — разбивает T на более мелкие.

# Типы данных

сделать

# TRUNCATE и DELETE

|  |  |  |
| --- | --- | --- |
|  | DELETE | TRUNCATE |
| Оператор | DML | DDL |
| Удаляет | все данные или по условию | все данные |
| счетчик ключа | продолжается | обнуляется |

TRUNCATE неприменима, если есть какая-либо связь с таблицей, и что‑то с триггерами.

|  |  |  |
| --- | --- | --- |
| журнализируются | отдельные строки таблиц | освобожденные страницы таблицы |

# NULL. Как проверить поле на NULL?

NULL — специальное значение (псевдозначение), которое можно записать в поле таблицы БД. NULL соответствует понятию «пустое поле», т. е. «поле, не содержащее никакого значения».

NULL означает отсутствие, неизвестность информации. Значение NULL не является значением в полном смысле слова: по определению оно означает отсутствие значения и не принадлежит ни одному типу данных. Поэтому NULL не равно ни логическому значению FALSE, ни пустой строке, ни 0. При сравнении NULL с любым значением будет получен результат NULL, а не FALSE и не 0. Более того, NULL не равен NULL.

Условие проверки отсутствия (наличия) значения NULL в полях таблицы:

|  |  |
| --- | --- |
| 1. | IS [NOT] NULL |
| 1.  2.  3. | SELECT \*  FROM PC  WHERE price IS NULL; |

# JOIN

JOIN — оператор языка SQL, который является реализацией операции соединения реляционной алгебры. Предназначен для обеспечения выборки данных из двух таблиц и включения этих данных в один результирующий набор.

Особенности операции соединения:  
∙ в схему таблицы-результата входят столбцы обеих исходных таблиц (таблиц-операндов), т. е. схема результата является «сцеплением» схем операндов;  
∙ каждая строка таблицы-результата является «сцеплением» строки из одной таблицы‑операнда со строкой второй таблицы‑операнда;  
∙ при необходимости соединения более двух таблиц операция соединения применяется несколько раз (последовательно).

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8. | SELECT  field\_name [,... n]  FROM  Table1  {INNER | {LEFT | RIGHT | FULL} OUTER | CROSS }  JOIN  Table2  {ON <condition>  | USING (field\_name [,... n])} |

## Типы JOIN

City

|  |  |
| --- | --- |
| Id | Name |
| 1 | Москва |
| 2 | Санкт-Петербург |
| 3 | Казань |

Person

|  |  |
| --- | --- |
| Name | CityId |
| Андрей | 1 |
| Леонид | 2 |
| Сергей | 1 |
| Григорий | 4 |

### (INNER) JOIN

Общие записи обоих таблиц; порядок таблиц не важен. (симметричный).

Результатом объединения таблиц являются записи, общие для левой и правой таблиц. Порядок таблиц для оператора не важен, поскольку оператор является симметричным.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | CityId | Id | Name |
| Андрей | 1 | 1 | Москва |
| Леонид | 2 | 2 | Санкт-Петербург |
| Сергей | 1 | 1 | Москва |

### LEFT (OUTER) JOIN

Все записи левой и соответствующие ей правой (NULL); порядок таблиц важен (не симметричный).

Производит выбор всех записей первой таблицы и соответствующих им записей второй таблицы. Если записи во второй таблице не найдены, то вместо них подставляется пустой результат (NULL). Порядок таблиц для оператора важен, поскольку оператор не является симметричным.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | CityId | Id | Name |
| Андрей | 1 | 1 | Москва |
| Леонид | 2 | 2 | Санкт-Петербург |
| Сергей | 1 | 1 | Москва |
| Григорий | 4 | NULL | NULL |

### RIGHT (OUTER) JOIN

LEFT JOIN с операндами, расставленными в обратном порядке.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | CityId | Id | Name |
| Андрей | 1 | 1 | Москва |
| Сергей | 1 | 1 | Москва |
| Леонид | 2 | 2 | Санкт-Петербург |
| NULL | NULL | 3 | Казань |

### FULL (OUTER) JOIN

Все записи (NULL); порядок таблиц не важен. (симметричный).

Результатом объединения таблиц являются все записи, которые присутствуют в таблицах. Порядок таблиц для оператора не важен, поскольку оператор является симметричным.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | CityId | Id | Name |
| Андрей | 1 | 1 | Москва |
| Сергей | 1 | 1 | Москва |
| Леонид | 2 | 2 | Санкт-Петербург |
| NULL | NULL | 3 | Казань |
| Григорий | 4 | NULL | NULL |

### CROSS JOIN (декартово произведение)

Каждая строка одной таблицы объединяется с каждой строкой второй; порядок таблиц не важен. (симметричный).

При выборе каждая строка одной таблицы объединяется с каждой строкой второй таблицы, давая тем самым все возможные сочетания строк двух таблиц. Порядок таблиц для оператора не важен, поскольку оператор является симметричным.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | CityId | Id | Name |
| Андрей | 1 | 1 | Москва |
| Андрей | 1 | 2 | Санкт-Петербург |
| Андрей | 1 | 3 | Казань |
| Леонид | 2 | 1 | Москва |
| Леонид | 2 | 2 | Санкт-Петербург |
| Леонид | 2 | 3 | Казань |
| Сергей | 1 | 1 | Москва |
| Сергей | 1 | 2 | Санкт-Петербург |
| Сергей | 1 | 3 | Казань |
| Григорий | 4 | 1 | Москва |
| Григорий | 4 | 2 | Санкт-Петербург |
| Григорий | 4 | 3 | Казань |

## JOIN и подзапросы

Обычно лучше использовать JOIN, поскольку в большинстве случаев он более понятен и лучше оптимизируется СУБД (но 100% этого гарантировать нельзя). Так же JOIN имеет заметное преимущество над подзапросами в случае, когда список выбора SELECT содержит столбцы более чем из одной таблицы.

Подзапросы лучше использовать в случаях, когда нужно вычислять агрегатные значения и использовать их для сравнений во внешних запросах.

# HAVING

HAVING используется для фильтрации результата GROUP BY по заданным логическим условиям.

Artists

|  |  |  |
| --- | --- | --- |
| Singer | Album | Sale |
| The Prodigy | Invaders Must Die | 1 200 000 |
| Drowning Pool | Sinner | 400 000 |
| Massive Attack | Mezzanine | 2 300 000 |
| The Prodigy | Fat of the Land | 600 000 |
| The Prodigy | Music For The Jilted Generation | 1 500 000 |
| Massive Attack | 100th Window | 1 200 000 |
| Drowning Pool | Full Circle | 800 000 |
| Massive Attack | Danny The Dog | 1 900 000 |
| Drowning Pool | Resilience | 500 000 |

|  |  |
| --- | --- |
| 1.  2.  3.  4. | SELECT Singer, SUM(Sale)  FROM Artists  GROUP BY Singer  HAVING SUM(Sale) > 2 000 000 |

|  |  |
| --- | --- |
| Singer | Sale |
| Massive Attack | 5 400 000 |
| The Prodigy | 3 300 000 |

## HAVING и WHERE

HAVING используется как WHERE, но в другой части SQL-выражения и, соответственно, на другой стадии формирования ответа.

|  |  |  |  |
| --- | --- | --- | --- |
| Операция | назначение | применяется | принадлежит |
| WHERE | ограничивает выражение | до получения результата | SELECT  UPDATE  DELETE |
| HAVING | фильтрует выражение | к результату | GROUP BY  (SELECT) |

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8. | SELECT  ...  FROM table\_references  [WHERE where\_definition]  [GROUP BY {unsigned\_integer | column  | formula}]  [HAVING where\_definition]  ... |

# GROUP BY

GROUP BY используется для агрегации записей результата по заданным признакам-атрибутам.

При использовании GROUP BY все значения NULL считаются равными.

Artists

|  |  |  |
| --- | --- | --- |
| Singer | Album | Sale |
| The Prodigy | Invaders Must Die | 1 200 000 |
| Drowning Pool | Sinner | 400 000 |
| Massive Attack | Mezzanine | 2 300 000 |
| The Prodigy | Fat of the Land | 600 000 |
| The Prodigy | Music For The Jilted Generation | 1 500 000 |
| Massive Attack | 100th Window | 1 200 000 |
| Drowning Pool | Full Circle | 800 000 |
| Massive Attack | Danny The Dog | 1 900 000 |
| Drowning Pool | Resilience | 500 000 |

|  |  |
| --- | --- |
| 1.  2.  3. | SELECT Singer, SUM(Sale) AS AllSales  FROM Artists  GROUP BY Singer |

|  |  |
| --- | --- |
| Singer | Sale |
| Drowning Pool | 1 700 000 |
| Massive Attack | 5 400 000 |
| The Prodigy | 3 300 000 |

## GROUP BY и DISTINCT

DISTINCT указывает, что для вычислений используются только уникальные значения столбца. NULL считается как отдельное значение. GROUP BY создает отдельную группу для всех возможных значений (включая значение NULL).

Если нужно удалить только дубликаты лучше использовать DISTINCT, GROUP BY лучше использовать для определения групп записей, к которым могут применяться агрегатные функции.

# ORDER BY

ORDER BY упорядочивает вывод запроса согласно значениям в том или ином количестве выбранных столбцов. Многочисленные столбцы упорядочиваются один внутри другого. Возможно определять возрастание ASC или убывание DESC для каждого столбца. По умолчанию установлено — возрастание.

|  |  |
| --- | --- |
| 1.  2.  3. | SELECT \*  FROM Artists  ORDER BY Singer |

# Временная таблица

Временная таблица — объект БД, который хранится и управляется системой БД на временной основе. Они могут быть локальными и глобальными. Используется для сохранения результатов вызова хранимой процедуры, уменьшение числа строк при соединениях, агрегирование данных из различных источников или как замена курсоров и параметризованных представлений.

# Представление

Представление (view) — виртуальная таблица, представляющая данные одной или более таблиц альтернативным образом.

В действительности представление — всего лишь результат выполнения оператора SELECT, который хранится в структуре памяти, напоминающей SQL таблицу. Они работают в запросах и операторах DML точно также как и основные таблицы, но не содержат никаких собственных данных. Представления значительно расширяют возможности управления данными. Это способ дать публичный доступ к некоторой (но не всей) информации в таблице.

# UNION

Объединяет результаты 2‑х SQL‑запросов в единую таблицу, состоящую из схожих записей. Оба запроса должны возвращать одинаковое число столбцов и совместимые типы данных в соответствующих столбцах. Необходимо отметить, что UNION сам по себе не гарантирует порядок записей. Записи из второго запроса могут оказаться в начале, в конце или вообще перемешаться с записями из первого запроса. В случаях, когда требуется определенный порядок, необходимо использовать ORDER BY.

# LIMIT

Выводит указанное число строк из таблицы.

# EXISTS

Берет подзапрос, как аргумент, и оценивает его как TRUE, если подзапрос возвращает какие‑либо записи и FALSE, если нет.

# IN, BETWEEN, LIKE

IN — определяет набор значений.

|  |  |
| --- | --- |
| 1.  2. | SELECT \* FROM Persons WHERE name  IN ('Ivan','Petr','Pavel'); |

BETWEEN определяет диапазон значений (чувствителен к порядку, и первое значение в предложении должно быть первым по алфавитному или числовому порядку).

|  |  |
| --- | --- |
| 1. | SELECT \* FROM Persons WHERE age BETWEEN 20 AND 25; |

LIKE применим только к полям типа CHAR или VARCHAR (любые строки?), с которыми он используется чтобы находить подстроки. В качестве условия используются символы шаблонизации (wildkards) (специальные символы, которые могут соответствовать чему-нибудь):  
∙ \_ замещает любой одиночный символ. Например, 'b\_t' будет соответствовать словам 'bat' или 'bit', но не будет соответствовать 'brat'.  
∙ % замещает последовательность любого числа символов. Например '%p%t' будет соответствовать словам 'put', 'posit', или 'opt', но не 'spite'.

|  |  |
| --- | --- |
| 1. | SELECT \* FROM UNIVERSITY WHERE NAME LIKE '%o'; |

# MERGE

MERGE позволяет осуществить слияние данных одной таблицы с данными другой таблицы. При слиянии таблиц проверяется условие, и если оно истинно, то выполняется UPDATE, а если нет — INSERT. При этом изменять поля таблицы в секции UPDATE, по которым идет связывание двух таблиц, нельзя.

# DATETIME и TIMESTAMP

|  |  |  |  |
| --- | --- | --- | --- |
| тип данных | формат хранения | учитывается  часовой пояс | размер |
| TIME | HH:MM:SS | нет | 3 байта |
| DATE | YYYY-MM-DD | нет | 3 байта |
| DATETIME | YYYY-MM-DD HH:MM:SS | нет | 8 байт |
| TIMESTAMP | секунды, миллисекунды или наносекунды с полуночи 1 января 1970 года | да | 4 байта |

# Агрегатные функции

Сводят группы значений к одиночному.

COUNT (количество записей, удовлетворяющих условию запроса);

SUM (арифметическая сумму);

AVG (среднее арифметическое);

MAX (наибольшее);

MIN (наименьшее).

## COUNT(\*) и COUNT({column})

|  |  |  |
| --- | --- | --- |
| оператор | подчитывает количество | NULL |
| COUNT(\*) | записей | учитывает |
| COUNT({column}) | значений | не учитывает |

# Ограничения (constraints) на целостность данных

CONSTRAINTS — позволяет задать собственное имя ограничению.

PRIMARY KEY  
∙ набор полей из уникальных комбинаций для однозначной идентификации записи в таблице;  
∙ только один на таблицу;  
∙ для обеспечения целостности сущности, которая описана таблицей.

CHECK — ограничение множества значений столбца (задать формат, значения по умолчанию, условия значения по отношению к другим столбцам). Обычно используется с IN, BETWEEN и LIKE.

UNIQUE — отсутствие дубликатов в столбце или наборе столбцов.

FOREIGN KEY  
∙ защищает от действий, которые могут нарушить связи между таблицами;  
∙ указывает на PRIMARY KEY в другой таблицы. Поэтому данное ограничение нацелено на то, чтобы не было записей FOREIGN KEY, которым не отвечают записи PRIMARY KEY.

Значение столбца, на который наложено ограничение FOREIGN KEY, может равняться NULL, если на данный столбец не наложено ограничение NOT NULL.

## PRIMARY и UNIQUE

|  |  |  |
| --- | --- | --- |
| оператор | Индекс по умолчанию | NULL |
| PRIMARY | кластерный | не разрешает |
| UNIQUE | некластерный | один или несколько (в зависимости от БД |

# Ключи

Суррогатный ключ — дополнительное служебное поле в качестве первичного ключа.

Простой ключ (1): одно поле.

Составной ключ (composite key) (2): два и более полей.

Потенциальный ключ (candidate key) (3): (1) или (2), который является уникальным.

Первичный ключ (primary key) (4): один из (3).

Альтернативный ключ (alternate key) остальные из (3).

(3) должен обладать критерием неизбыточности (при удалении поля ключ утрачивает уникальность).

Критерии выбора (4):  
∙ размер и/или количество полей;  
∙ с наибольшей вероятностью не утратит уникальность.

Внешний ключ (foreign key) — обеспечивает однозначную логическую связь между таблицами одной БД (в одной из таблиц — потенциальный ключ).

# Индекс (index)

Индекс — объект БД для повышения производительности выборки данных.

Наборы данных могут иметь большое количество записей, которые хранятся в произвольном порядке, и их поиск по заданному критерию путем последовательного просмотра набора данных запись за записью может занимать много времени. Индекс формируется из значений одного или нескольких полей и указателей на соответствующие записи набора данных, — т. о., достигается значительный прирост скорости выборки из этих данных.

Преимущества:  
∙ ускорение поиска и сортировки по определенному полю (набору полей);  
∙ обеспечение уникальности данных.

Недостатки:  
∙ дополнительное место (на диске и в оперативной памяти) в зависимости от длины ключа;  
∙ замедление операций (вставки, обновления и удаления записей — приходится обновлять сами индексы).

Стоит использовать для:  
∙ поля-счетчика (избежать повторения значений в этом поле);  
∙ поля с сортировкой;  
∙ полей, по которым часто проводится соединение наборов данных (данные располагаются в порядке возрастания индекса и соединение происходит значительно быстрее);  
∙ поля с первичным ключом;  
∙ поля из диапазона (как только будет найдена первая запись с нужным значением, все последующие значения будут расположены рядом).

Не стоит использовать для:  
∙ редко используемых полей (в запросах);  
∙ полей с двумя или тремя значениями (мужской/женский, да/нет).

## Типы индексов (и)

По порядку сортировки:  
∙ упорядоченные;  
∙ возрастающие;  
∙ убывающие;  
∙ неупорядоченные.

По источнику данных:  
∙ и по представлению;  
∙ и по выражениям.

[По воздействию на источник данных:  
∙ кластерный и;  
∙ некластерный и.](#_Кластерные_и_некластерные)

По структуре:  
∙ B\*‑деревья;  
∙ B+‑деревья;  
∙ B‑деревья;  
∙ хэши.

По количественному составу:  
∙ простой и (по одному полю);  
∙ составной и (по нескольким полям);  
∙ и с включенными столбцами (некластеризованный индекс c ключевыми столбцами + неключевыми);  
∙ главный и (индексный ключ, под управлением которого в данный момент находится набор данных).

По характеристике содержимого:  
∙ уникальный и (из множества уникальных значений поля);  
∙ плотный и (и, при котором, каждом документе в индексируемой коллекции соответствует запись в индексе, даже если в документе нет индексируемого поля);  
∙ разреженный и (и, в котором представлены только те документы, для которых индексируемый ключ имеет какое‑то определенное значение (существует));  
∙ пространственный и (оптимизирован для описания географического местоположения (многоключевой индекс из широты и долготы));  
∙ составной пространственный и (и, включающий в себя кроме широты и долготы еще какие‑либо мета‑данные (например теги); но географические координаты должны стоять на первом месте);  
∙ полнотекстовый и (словарь, в котором перечислены все слова и указано, в каких местах они встречаются; при наличии такого индекса достаточно осуществить поиск нужных слов в нем и тогда сразу же будет получен список документов, в которых они встречаются);  
∙ хэш-и (вместо значений хранит хэши — уменьшается размер (увеличивается скорость их обработки) и из больших полей; при запросах сравниваются не искомое со значения поля, а хэш от искомого значения с хэшами полей; нельзя сортировать по значению, невозможно использовать в сравнениях больше/меньше и «is null»; хэши не уникальны — совпадающих хэшей применяются методы разрешения коллизий);  
∙ битовый и (метод битовых индексов заключается в создании отдельных битовых карт (последовательностей 0 и 1) для каждого возможного значения столбца, где каждому биту соответствует запись с индексируемым значением, а его значение равное 1 означает, что запись, соответствующая позиции бита содержит индексируемое значение для данного столбца или свойства);  
∙ обратный и (B‑tree и, но с реверсированным ключом, используемый в основном для монотонно возрастающих значений (например, автоинкрементный идентификатор) в OLTP системах с целью снятия конкуренции за последний листовой блок и, т. к. благодаря переворачиванию значения две соседние записи индекса попадают в разные блоки и. Он не может использоваться для диапазонного поиска);  
∙ функциональный и (и, ключи которого хранят результат пользовательских функций; часто строятся для полей, значения которых проходят предварительную обработку перед сравнением в команде SQL; например, при сравнении строковых данных без учета регистра символов часто используется функция UPPER; кроме того, функциональный индекс может помочь реализовать любой другой отсутствующий тип индексов данной СУБД);  
∙ первичный и (уникальный индекс по полю первичного ключа);  
∙ вторичный и (и по другим полям кроме поля первичного ключа);  
∙ XML-и (вырезанное материализованное представление больших двоичных XML‑объектов (BLOB) в столбце с типом данных xml).

По механизму обновления:  
∙ полностью перестраиваемый (при добавлении элемента заново перестраивается весь и);  
∙ пополняемый (при добавлении элементов и перестраивается частично (например одна из ветви) и периодически балансируется).

По покрытию индексируемого содержимого:  
∙ полностью покрывающий и (покрывает все содержимое индексируемого объекта);  
∙ частичный и (и, построенный на части набора данных, удовлетворяющей определенному условию самого и; создан для уменьшения размера и);  
∙ инкрементный и (делта—и) (индексируется малая часть данных (дельта), как правило, по истечении определенного времени; используется при интенсивной записи; например, полный индекс перестраивается раз в сутки, а дельта‑и строится каждый час; по сути это частичный и по временной метке);  
∙ и реального времени (особый вид инкрементного и, характеризующийся высокой скоростью построения; предназначен для часто меняющихся данных).

Индексы в кластерных системах:  
∙ глобальный и (и по всему содержимому всех сегментов БД);  
∙ сегментный и (глобальный и по полю-сегментируемому ключу; используется для быстрого определения сегмента, на котором хранятся данные в процессе маршрутизации запроса в кластере БД);  
∙ локальный и (и по содержимому только одного сегмента БД).

## Кластерные и некластерные индексы

Кластерный индекс:  
∙ данные физически упорядочены;  
∙ повышает скорость выборок данных в случае последовательного доступа к данным;  
∙ для одного набора данных — один кластерный индекс.

Некластерный индекс:  
∙ данные физически расположены в произвольном порядке, но логически упорядочены согласно индексу;  
∙ для часто изменяемого набора данных.

## Индексировать данные, имеющие небольшое количество возможных значений?

Если объем информации (в байтах) не удовлетворяющей условию выборки меньше, чем размер индекса (в байтах) по данному условию выборки, то в общем случае оптимизация приведет к замедлению выборки.

## Полное сканирование выгоднее доступа по индексу?

Полное сканирование — многоблочным чтением.

Сканирование по индексу — одноблочным. Также, при доступе по индексу сначала идет сканирование самого индекса, а затем чтение блоков из набора данных. Число блоков, которые надо при этом прочитать из набора зависит от фактора кластеризации. Если суммарная стоимость всех необходимых одноблочных чтений больше стоимости полного сканирования многоблочным чтением, то полное сканирование выгоднее, и оно выбирается оптимизатором.

Полное сканирование:  
∙ при слабой селективности предикатов запроса и/или слабой кластеризации данных;  
∙ в случае очень маленьких наборов данных.

# Создать индекс

Индекс можно создать либо с помощью выражения CREATE INDEX:

|  |  |
| --- | --- |
| 1.  2. | CREATE INDEX index\_name  ON table\_name (column\_name) |

Либо указав ограничение целостности в виде уникального UNIQUE или первичного PRIMARY ключа в операторе создания таблицы CREATE TABLE.

# DELETE и TRUNCATE

|  |  |  |  |
| --- | --- | --- | --- |
| DELETE | DML | удаляет записи из таблицы (WHERE) | задействуются триггеры, ограничения и т. д. |
| TRUNCATE | DDL | удаляет таблицу и заново создает | если используется в репликации на таблицу или есть ссылки FOREIGN KEY, то пересоздать не получится |

# Хранимая процедура

Хранимая процедура — некоторая аналогия методов в БД. Могут  
∙ иметь входные и выходные параметры, локальные переменные;  
∙ производить числовые вычисления и операции над символьными данными, присваивать результаты переменным и параметрам;  
∙ выполняться стандартные операции с базами данных;  
∙ иметь циклы и ветвления (могут использоваться инструкции управления процессом исполнения).

Позволяют  
∙ повысить производительность;  
∙ расширяют возможности программирования;  
∙ поддерживают функции безопасности данных.

В большинстве СУБД при первом запуске компилируются (в дальнейшем ее обработка осуществляется быстрее).

## Триггер (trigger)

Триггер — хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: добавлением, удалением или изменением данных в заданной таблице реляционной базы данных. Триггеры применяются для обеспечения целостности данных и реализации сложной бизнес-логики. Триггер запускается сервером автоматически и все производимые им модификации данных рассматриваются как выполняемые в транзакции, в которой выполнено действие, вызвавшее срабатывание триггера. Соответственно, в случае обнаружения ошибки или нарушения целостности данных может произойти откат этой транзакции.

Момент запуска триггера определяется с помощью ключевых слов BEFORE (триггер запускается до выполнения связанного с ним события) или AFTER (после события). В случае, если триггер вызывается до события, он может внести изменения в модифицируемую событием запись. Кроме того, триггеры могут быть привязаны не к таблице, а к представлению. В этом случае с их помощью реализуется механизм «обновляемого представления». В этом случае ключевые слова BEFORE и AFTER влияют лишь на последовательность вызова триггеров, т. к. собственно событие (удаление, вставка или обновление) не происходит.

# Транзакция

Транзакция — воздействие на БД согласно основным свойствам.

## Основные свойства транзакции

∙ Атомарность (выполнена/не выполнено, без промежуточного состояния);

∙ Согласованность (завершенная транзакция сохраняет согласованность БД);

∙ Изолированность (работают независимо друг от друга);

∙ Долговечность (гарантия, что изменения завершенной транзакцией будет применены).

Уровни изолированности транзакций (в порядке увеличения изолированности транзакций и надежности работы с данными):

∙ Чтение неподтвержденных данных (грязное чтение)  
∙ чтение незафиксированных изменений транзакций;  
∙ (2), (3), (4).

∙ Чтение подтвержденных данных  
∙ чтение зафиксированных изменений транзакций;  
∙ (3), (4).

∙ Повторяемость чтения  
∙ чтение всех изменений своей транзакции, любые изменения, внесенные параллельными транзакциями после начала своей работы, недоступно;  
∙ (4).

∙ Упорядочиваемость  
∙ результат параллельного выполнения сериализуемой транзакции с другими транзакциями должен быть логически эквивалентен результату их какого‑либо последовательного выполнения;  
∙ проблемы синхронизации не возникают.

## Проблемы при параллельном выполнении транзакции

1. Потерянное обновление (одновременное изменение одного блока);

2. «Грязное» чтение (откат);

3. Неповторяющееся чтение (при повторном чтении данные изменились);

4. Фантомы (одни и те же выборки дают разные множества записей).

# Нормализация и денормализация

Нормализация:  
∙ приведение к виду, отвечающему нормальным формам;  
∙ минимальная избыточность.

Денормализация:  
∙ приведение к виду, не соответствующему правилам нормализации;  
∙ для повышения производительности и скорости извлечения данных (увеличение избыточности).

Нормальная форма:

1NF:  
∙ значения атрибутов атомарны (ячейка не содержит перечисления).

2NF:  
∙ 1NF;  
∙ неключевые атрибуты зависят только от ключа целиком (если он составной).

3NF:  
∙ 2NF;  
∙ неключевые атрибуты не зависят друг от друга.

# PIVOT и UNPIVOT в Transact‑SQL

PIVOT и UNPIVOT являются нестандартными реляционными операторами, которые поддерживаются Transact‑SQL.

Оператор PIVOT разворачивает возвращающее табличное значение выражение, преобразуя уникальные значения одного столбца выражения в несколько выходных столбцов, а также, в случае необходимости, объединяет оставшиеся повторяющиеся значения столбца и отображает их в выходных данных. Оператор UNPIVOT производит действия, обратные PIVOT, преобразуя столбцы возвращающего табличное значение выражения в значения столбца.

# INTERSECT, EXCEPT в Transact‑SQL

Оператор EXCEPT возвращает уникальные записи из левого входного запроса, которые не выводятся правым входным запросом.

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9. | SELECT ID, NAME, AMOUNT, DATE  FROM CUSTOMERS  LEFT JOIN ORDERS  ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID  EXCEPT  SELECT ID, NAME, AMOUNT, DATE  FROM CUSTOMERS  RIGHT JOIN ORDERS  ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID; |

Оператор INTERSECT возвращает уникальные записи, выводимые левым и правым входными запросами.

|  |  |
| --- | --- |
| 1.  2.  3. | SELECT ship FROM Outcomes  INTERSECT  SELECT name FROM Ships; |

# Курсор

Курсор — объект БД, который позволяет приложениям работать с записями «по‑одной», а не сразу с множеством, как это делается в обычных SQL командах.

Порядок работы с курсором такой:  
∙ определить курсор (DECLARE);  
∙ открыть курсор (OPEN);  
∙ получить запись из курсора (FETCH);  
∙ обработать запись;  
∙ закрыть курсор (CLOSE);  
∙ удалить ссылку курсора (DEALLOCATE) (когда удаляется последняя ссылка курсора, SQL освобождает структуры данных, составляющие курсор).

# Для каких числовых типов недопустимо использовать операции сложения/вычитания?

В качестве операндов операций сложения и вычитания нельзя использовать числовой тип BIT.

# Основные функции ранжирования в Transact‑SQL.

Ранжирующие функции — функции, которые возвращают значение для каждой записи группы в результирующем наборе данных. На практике они могут быть использованы, например, для простой нумерации списка, составления рейтинга или постраничной навигации.

ROW\_NUMBER — функция нумерации в Transact-SQL, которая возвращает просто номер записи.

RANK возвращает ранг каждой записи. В данном случае, в отличие от ROW\_NUMBER, идет уже анализ значений и в случае нахождения одинаковых возвращает одинаковый ранг с пропуском следующего.

DENSE\_RANK так же возвращает ранг каждой записи, но в отличие от RANK в случае нахождения одинаковых значений возвращает ранг без пропуска следующего.

NTILE — функция Transact-SQL, которая делит результирующий набор на группы по определенному столбцу.