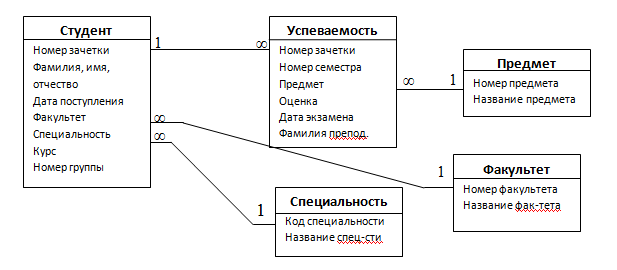
1. Основные понятия теории баз данных

2. Модели данных.

3. ER – диаграммы

ER-модель (от англ. entity-relationship model, модель «сущность — связь») — модель данных, позволяющая описывать концептуальные схемы предметной области.

ER-модель используется при высокоуровневом (концептуальном) проектировании баз данных. С её помощью можно выделить ключевые сущности и обозначить связи, которые могут устанавливаться между этими сущностями.

Во время проектирования баз данных происходит преобразование ER-модели в конкретную схему базы данных на основе выбранной модели данных (реляционной, объектной, сетевой или др.).

ER-модель представляет собой формальную конструкцию, которая сама по себе не предписывает никаких графических средств её визуализации. В качестве стандартной графической нотации, с помощью которой можно визуализировать ER-модель, была предложена диаграмма «сущность-связь»

4. Проектирование инфологической модели данных.

5. Основные понятия реляционной алгебры.

6. Операции реляционной алгебры.

7. Нормализация таблиц реляционной базы данных.

8. Общая структура оператора выборки и оператора создания таблицы в

языке SQL.

Синтаксис оператора SELECT для SQL имеет следующий вид:

SELECT [ALL | DISTINCT] (<Список полей>|\*)

FROM <Список таблиц>

[WHERE <Предикат-условие селекции или соединения>]

[GROUP BY <Список полей группировки>]

[HAVING <Предикат-условие селекции для группы>]

[ORDER BY <Список полей, по которым упорядочивается вывод записей>]

Например, в WHERE могут использоваться следующие предикаты:

* Сравнения;
* Between;
* IN;
* LIKE, NOT LIKE;
* IS NULL;
* EXIST.

Оператор CREATE TABLE создает таблицу с заданным именем в текущей базе данных. Если нет активной текущей базы данных или указанная таблица уже существует, то возникает ошибка выполнения команды. Ключевые слова IF NOT EXISTS указываются для того, чтобы не возникала ошибка, если указанная таблица уже существует.

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] Имя\_таблицы

[(Create-определение,...)]

[Опции\_таблицы]

[SELECT ... (любое корректное выражение SELECT)]

Create-определение может быть одним из вариантов:

Определение поля:

Имя\_столбца type [NOT NULL | NULL] [DEFAULT Значение\_по\_умолчанию] [AUTO\_INCREMENT]

Определение первичного ключа:

PRIMARY KEY (Имя\_столбца,...)

Определение индекса (ключа):

{INDEX | KEY} [index\_name] (Имя\_столбца,...)

9. Использование операторов: DELETE, INSERT, UPDATE.

10. Общая характеристика оператора SELECT и организация списка

ссылок на таблицы в разделе FROM.

11. Создание запросов SELECT. Виды запросов. Использование

DISTINCT.

12. Использование агрегатных функций.

Запросы могут производить обобщенное групповое значение полей точно также как и значение одного пол. Это делает с помощью агрегатных функций. Агрегатные функции производят одиночное значение для всей группы таблицы. Имеется список этих функций:

• COUNT - производит номера строк или не-NULL значения полей которые выбрал.

• SUM - производит арифметическую сумму всех выбранных значений.

• AVG - производит усреднение всех выбранных значений.

• MAX - производит наибольшее из всех выбранных значений.

• MIN - производит наименьшее из всех выбранных значений.

Групповые функции используются подобно именам полей в операторе SELECT, но с одним исключением: они берут имя поля как аргумент.

С функциями SUM и AVG могут использоваться только числовые поля.

С функциями COUNT, MAX и MIN могут использоваться как числовые, так и символьные поля. При использова¬нии с символьными полями МАХ и MIN будут транслировать их в эквивалент ASCII кода и обрабатывать в алфавитном порядке.

13. Использование предложения GROUP BY.

GROUP BY позволяет перейти от одной таблицы с одними объектами к другой таблице с другими объектами.

Пример: SELECT cnum FROM Orders GROUP BY cnum, cname

Выбор клиента, у которого больше 10 заказов:

Пример: SELECT cnum, COUNT(\*) FROM Orders GROUP BY cnum, cname HAVING COUNT(\*)>10

Группирует выбранный набор строк для получения набора сводных строк по значениям одного или нескольких столбцов или выражений в SQL Server 2014. Возвращается одна строка для каждой группы. Агрегатные функции в списке <select> предложения SELECT предоставляют информацию о каждой группе, а не об отдельных строках.

14. Фильтрация групп, используя предложение HAVING

В дополнение к способности группировать данные с помощью предложения GROUP BY, SQL также позволяет осуществлять фильтрацию — указывать, какие группы должны быть включены в результат, а какие исключены из него. Задать условия для групп можно с помощью предложения HAVING.

В самом общем смысле, предложение HAVING работает аналогично предложению WHERE, но применяется к группам. WHERE накладывает ограничения на строки, a HAVING — на группы. Т.е. сначала GROUP BY разделяет строки на наборы (по типу), затем на полученные группы накладываются условия предложения HAVING.

Если в списке выбора имеются агрегирующие функции, предложение WHERE выполняется перед ними, тогда как предложение HAVING применяется ко всему запросу в целом, после вычисления значений функций и разбиения на группы.

С точки зрения синтаксиса условного выражения, предложения HAVING и WHERE идентичны, отличие состоит лишь в том, что в условии предложения WHERE не могут находиться агрегирующие функции. Кроме того, элементы предложения HAVING быть перечислены в списке выбора.

Если в предложении HAVING есть несколько условий, они объединяются с помощью операторов AND, OR и NOT.

15.Соединения.

# Внутренние соединения

Соединения с помощью предиката JOIN бывают внутренние и внешние.

* Внутренние – INNER JOIN, JOIN, CROSS JOIN
* Внешние – OUTER FULL JOIN, RIGHT JOIN, LEFT JOIN

Начнем с внутренних операций JOIN

Пример:

|  |  |  |  |
| --- | --- | --- | --- |
| Auto | | Selling | |
| id | Name | Id | sum |
| 1 | BMW | 1 | 250 |
| 2 | Opel | 5 | 450 |
| 3 | KIA | 3 | 800 |
| 4 | Audi | 6 | 400 |

1. INNER JOIN  
   SELECT \*  
   FROM Auto  
   [INNER] Join Selling  
   ON Auto.id = Selling.id

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Name | Id | sum |
| 1 | BMW | 1 | 250 |
| 3 | KIA | 3 | 800 |

1. CROSS JOIN – выдает все виды комбинаций всех строк первой таблицы со строками второй таблицы.

SELECT Autoid AS id  
FROM Auto  
CROSS Join Selling  
ON Auto.id = Selling.id

Внешние соединения

1. FULL JOIN – соединяет все (полное соединение всего (все строчки и первой и второй таблицы))

SELECT Auto.id, Auto.name, Auto.sum

FROM Auto

Full Join Selling

ON Auto.id= Selling.id

|  |  |  |
| --- | --- | --- |
| Auto.id | Name | Id |
| 1 | BMW | 250 |
| 2 | Opel | NULL |
| 3 | KIA | 300 |
| 4 | Audi | NULL |
| 5 | NULL | 450 |
| 6 | NULL | 400 |

1. LEFT JOIN – соединяет левую таблицу полностью данными из правой таблицы, если их нет, то заполнит поля NULL  
   SELECT Auto.id, Auto.name, Auto.sum

FROM Auto

LEFT Join Selling

ON Auto.id= Selling.id

|  |  |  |
| --- | --- | --- |
| Auto.id | Name | Id |
| 1 | BMW | 250 |
| 2 | Opel | NULL |
| 3 | KIA | 300 |
| 4 | Audi | NULL |

16-17. Создание автономных подзапросов и создание коррелированных подзапросов.

Подзапросы делятся на два типа:

* Автономные
* Коррелированные

Автономный запрос – возможно выполнение определенных строк в SQL Server.

Коррелированный – невозможно выполнение определенных строк т.к. есть корреляция с внешними запросами, запрос не выполнится.

Пример автономного запроса:  
SELECT \* FORM Orders WHERE snum (SELECT snum FROM salespeople WHERE city=”Barcelona”)

Пример коррелированного запроса:

SELECT snum, sname FROM Salespeople main WHERE 1 < (SELECT COUNT (\*) FROM Customers WHERE snum=main.snum)

18. Нормативное регулирование бухгалтерского учёта в России.

19. Виды учёта. Ведущая роль бухгалтерского учёта.

20. Задачи бухгалтерского учёта.

21. Метод бухгалтерского учёта. Калькуляция. Двойная запись. 22.Метод бухгалтерского учёта. Отчётность. Её роль в экономикепредприятия.

23. Предмет бухгалтерского учёта. Хозяйственные средства.

24. Учётные измерители.

25. План счетов бухгалтерского учёта. Виды счетов. 26.Формирование уставного капитала. Счет 80.

27. Расчеты с учредителями. Счет 75

28. Активный счёт. Пассивный счёт. Дать характеристику.

29. Как определить сальдо конечное на активных счетах.

30. Как определить сальдо конечное на пассивных счетах 31.Активно – пассивные счета. Примеры.

32. Материалы. Счёт No10.

33. Основное производство. Счёт No20.

34. Общехозяйственные расходы. Счет 26

35. Начисление заработной платы. Счет 70

36. Наличные расчеты. Касса. Счёт No50.

37. НДС. Особенности его учёта. Счёт No19. 68

38. Безналчные расчёты. Счёт 51. Как открыть расчётный счёт. 39.Товары. Учёт товаров. Счёт 41

40. НДФЛ заработной платы. Счёт 68. 01.

41. Учет Основных средств. Счет 08.04, счет 01.

42. Расчет амортизации. Счет 02

43. Готовая продукция. Учёт готовой продукции. Счёт 43.

44. Учёт расчётов с поставщиками и подрядчиками. Счёт 60.

45. Учёт расчётов с покупателями и заказчиками. Счёт 62.

46. Налоги. Виды налогов. Расчёты по налогам и сборам. Счёт 68.

47. Государственные внебюджетные фонды. Счёт 69.

48. Регламентированная отчетность, баланс