1. Использование операций GROUPING SETS, ROLLUP и CUBE. Примеры запросов.

GROUPING SETS позволяют указать несколько группировок в одном запросе. Это позволяет создавать сложные отчеты с множеством различных агрегатов.

Пример: Подсчет общего количества продаж по разным категориям: по продуктам, по регионам, по комбинациям и по всем категориям.

```
SELECT
    product,
    region,
    SUM(sales) AS total_sales
FROM
    sales_data
GROUP BY
    GROUPING SETS ((product), (region), (product, region), ());
```

ROLLUP создает иерархическую группировку. Он группирует по перечисленным колонкам и их подмножествам.

Пример: Подсчет общего количества продаж по категориям и подкатегориям (например, год, месяц, день).

```
SELECT
    year,
    month,
    day,
    SUM(sales) AS total_sales
FROM
    sales_data
GROUP BY
    ROLLUP (year, month, day);
```

CUBE генерирует все возможные комбинации группировок для заданных столбцов.

Пример: Подсчет общего количества продаж по всем возможным комбинациям продукта и региона.

```
SELECT
    product,
    region,
    SUM(sales) AS total_sales
FROM
    sales_data
GROUP BY
    CUBE (product, region);
```

2. Использование комбинирующих запросов: оператор UNION. Пример запроса.

Оператор **UNION** объединяет результаты двух или более запросов в один набор данных, исключая дубликаты.

Пример: Объединение списков клиентов из двух таблиц.

```
SELECT customer_id, customer_name
FROM customers_2023
UNION
SELECT customer_id, customer_name
FROM customers_2024;
```

3. Использование комбинирующих запросов: оператор EXCEPT. Пример запроса.

Оператор **EXCEPT** возвращает строки, которые присутствуют в первом запросе, но отсутствуют во втором.

Пример: Найти клиентов, которые были в 2023 году, но не были в 2024.

```
SELECT customer_id, customer_name
FROM customers_2023
EXCEPT
SELECT customer_id, customer_name
FROM customers_2024;
```

4. Использование комбинирующих запросов: оператор INTERSECT. Пример запроса.

Оператор **INTERSECT** возвращает только те строки, которые присутствуют в обоих запросах.

Пример: Найти клиентов, которые были как в 2023, так и в 2024 году.

```
SELECT customer_id, customer_name
FROM customers_2023
INTERSECT
SELECT customer_id, customer_name
FROM customers_2024;
```

5. Основы подзапросов. Простые подзапросы.

2 SELECTA

Подзапрос (вложенный запрос) — это запрос внутри другого SQL -запроса.

Пример: Найти все продукты, которые были проданы в количестве больше среднего.

```
SELECT product_name
FROM sales_data
WHERE quantity > (
    SELECT AVG(quantity)
```

```
    FROM sales_data
);
```

6. Скалярные подзапросы.

Скалярный подзапрос возвращает одно значение и может использоваться там, где требуется одиночное значение.

Пример: Получить список продуктов с указанием средней цены по всем продуктам.

```
SELECT product_name,
       price,
       (SELECT AVG(price) FROM products) AS avg_price
FROM products;
```

7. Табличные подзапросы.

Табличный подзапрос возвращает набор строк и используется в качестве временной таблицы.

Пример: Получить список продуктов с наибольшими продажами по каждому региону.

```
SELECT product_name, region, total_sales
FROM (
    SELECT product_name, region, SUM(sales) AS total_sales
    FROM sales_data
    GROUP BY product_name, region
) AS sales_summary
WHERE total_sales = (
    SELECT MAX(total_sales)
    FROM (
        SELECT product_name, region, SUM(sales) AS total_sales
        FROM sales_data
        GROUP BY product_name, region
    ) AS region_sales
);
```

8. Сложные подзапросы.
3 SELECTa

Сложные подзапросы могут включать в себя множество вложенных запросов, которые могут быть связаны с внешним запросом.

Пример: Получить список сотрудников, которые зарабатывают больше среднего по их отделу.

```
SELECT employee_name, salary
FROM employees e1
WHERE salary > (
    SELECT AVG(salary)
    FROM employees e2
```

```
    WHERE e1.department_id = e2.department_id
);
```

9. Оконные функции.

Оконные функции позволяют выполнять вычисления по частям набора строк (окнам), не сводя результат к одной строке на группу.

Пример: Получить накопительный итог по продажам для каждого месяца.

```
SELECT
    month,
    sales,
    SUM(sales) OVER (ORDER BY month) AS cumulative_sales
FROM
    sales_data;
```

10. Концепция транзакций: начать выполнение группы операций, зафиксировать, отменить, поставить точку сохранения.

Транзакция — это последовательность операций, выполняемых как единое целое.

Пример:

```
BEGIN TRANSACTION;

-- Выполнение нескольких операций
INSERT INTO accounts (account_id, balance) VALUES (1, 1000);
UPDATE accounts SET balance = balance - 500 WHERE account_id = 1;

-- Установка точки сохранения
SAVEPOINT my_savepoint;

-- Если все операции успешны, фиксируем транзакцию
COMMIT;

-- Если произошла ошибка, можно откатить до точки сохранения
-- ROLLBACK TO SAVEPOINT my_savepoint;

-- Или откатить всю транзакцию
-- ROLLBACK;
```

11. Транзакции и свойства ACID. Сериализация транзакций.

Свойства ACID — это набор из четырех свойств, гарантирующих надежность транзакций: Атомарность, Согласованность, Изолированность и Долговечность.

- **Атомарность (Atomicity)**: Все операции в транзакции либо выполняются полностью, либо не выполняются вовсе.
- **Согласованность (Consistency)**: Каждая транзакция переводит базу данных из одного согласованного состояния в другое.
- **Изолированность (Isolation)**: Одновременные транзакции не должны влиять на выполнение друг друга.

- **Долговечность (Durability)**: После завершения транзакции ее результаты должны быть постоянными, даже в случае сбоя.

Сериализация транзакций обеспечивает, чтобы транзакции выполнялись так, как будто они выполняются последовательно.

12. Уровни изоляции транзакций.

Уровни изоляции определяют степень видимости и блокировки данных между одновременными транзакциями:

1. **Read Uncommitted**: Транзакции могут видеть незафиксированные изменения других транзакций.
2. **Read Committed**: Транзакции видят только зафиксированные изменения.
3. **Repeatable Read**: Транзакции видят данные такими, какими они были в начале транзакции, даже если другие транзакции изменяют данные.
4. **Serializable**: Полная изоляция, транзакции исполняются так, как если бы они были выполнены последовательно.

13. Транзакции. Механизмы блокировки: на уровне таблиц, строк, рекомендательная блокировка.

Механизмы блокировки предотвращают одновременное выполнение операций, которые могут привести к некорректному состоянию данных.

- **Блокировка на уровне таблиц**: Заблокировать всю таблицу для изменения.
- **Блокировка на уровне строк**: Заблокировать только конкретные строки.
- **Рекомендательная блокировка (Advisory locks)**: Пользовательские блокировки, которые применяются по договоренности между приложениями.

Пример блокировки строки:

```
BEGIN TRANSACTION;

-- Блокировка строки с конкретным идентификатором
SELECT * FROM accounts WHERE account_id = 1 FOR UPDATE;

-- Выполнение операции
UPDATE accounts SET balance = balance - 500 WHERE account_id = 1;

COMMIT;
```

14. Семантическое описание предметной области, бизнес-правила.

Семантическое описание предметной области охватывает определения и правила, которые определяют, как данные связаны и какие операции возможны.

Пример: В банковской системе, счет клиента должен всегда иметь неотрицательный баланс. Это правило определяет ограничения на допустимые операции с данными.

15. Концептуальное проектирование. Модель «сущность-связь».

Концептуальное проектирование включает в себя создание модели данных на высоком уровне абстракции, обычно с использованием диаграмм "сущность-связь" (ER-моделей).

- **Сущности (Entities)**: Объекты, которые имеют значение в системе (например, Клиенты, Счета).
- **Связи (Relationships)**: Отношения между сущностями (например, Клиент владеет счетом).

16. Определение сущностей. Классификация сущностей.

Сущность — это объект или концепция, о которых собираются данные.

- **Сильная сущность**: Имеет собственный первичный ключ.
- **Слабая сущность**: Не имеет собственного ключа и зависит от другой сущности.
- **Абстрактная сущность**: Не существует как реальный объект, но имеет смысл в модели данных.

17. Определение атрибутов. Классификация атрибутов: простой, составной, однозначный, многозначный, производный, ключевой, неключевой, обязательный, необязательный.

Атрибуты — характеристики или свойства сущностей.

- **Простой атрибут**: Не делится на более мелкие компоненты (например, возраст).
- **Составной атрибут**: Может быть разделен на более мелкие компоненты (например, адрес, состоящий из улицы, города и почтового кода).
- **Однозначный атрибут**: Имеет одно значение для сущности (например, идентификатор).
- **Многозначный атрибут**: Может иметь несколько значений для одной сущности (например, телефонные номера).
- **Производный атрибут**: Значение которого может быть вычислено (например, возраст, рассчитанный на основе даты рождения).
- **Ключевой атрибут**: Уникально идентифицирует сущность (например, ID).
- **Неключевой атрибут**: Не участвует в идентификации сущности.
- **Обязательный атрибут**: Должен иметь значение.
- **Необязательный атрибут**: Может не иметь значения.

18. Определение доменов. Определение ключей: суперключ, потенциальный ключ, первичный ключ, альтернативный ключ, внешний ключ.

Домен — это допустимый набор значений для атрибута.

- **Суперключ**: Любой набор атрибутов, который уникально идентифицирует строку в таблице.
- **Потенциальный ключ**: Минимальный суперключ (не содержит лишних атрибутов).
- **Первичный ключ**: Выбранный потенциальный ключ, который используется для уникальной идентификации строк.
- **Альтернативный ключ**: Потенциальный ключ, который не выбран в качестве первичного.

- **Внешний ключ**: Атрибут или набор атрибутов, который ссылается на первичный ключ другой таблицы.

19. Определение связей: обязательность, кратность (кардинальность).

Связи определяют, как сущности связаны друг с другом.

- **Обязательность**: Определяет, должна ли связь существовать (например, каждый счет должен быть связан с клиентом).
- **Кратность (кардинальность)**: Определяет количество возможных связей между сущностями (например, один ко многим, многие ко многим).

Пример: В системе "сотрудник - проект" кратность может быть "многие ко многим", поскольку один сотрудник может работать над несколькими проектами, а один проект может включать нескольких сотрудников.

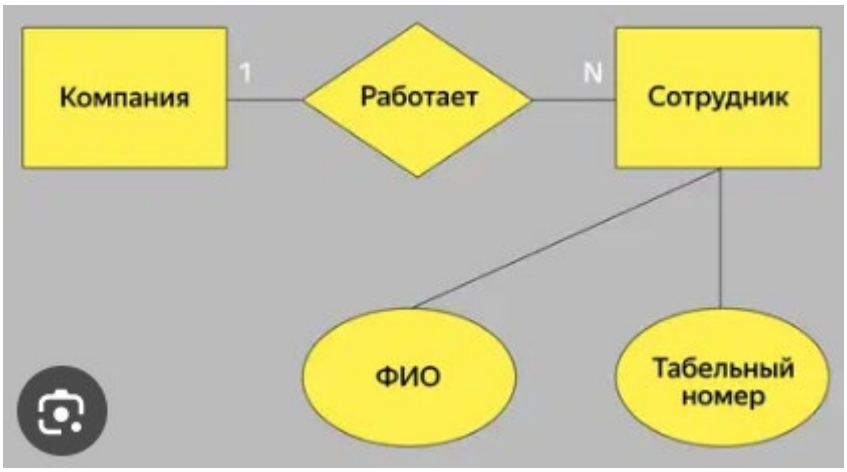
20. Моделирование связей между объектами: ER-диаграммы: нотация Чена

ER-диаграммы (Entity-Relationship) — это графический метод представления сущностей и связей между ними. **Нотация Чена** — одна из первых и широко используемых форм представления таких диаграмм.

Нотация Чена:

- **Сущности** изображаются прямоугольниками.
- **Атрибуты** изображаются овалами, соединенными линиями с соответствующими сущностями.
- **Связи** изображаются ромбами, соединяющими сущности. В ромбах указываются названия связей.
- **Кардинальность** указывается рядом с линиями, связывающими сущности, и показывает, сколько объектов одной сущности может быть связано с объектами другой сущности (например, "1", "N").

Пример ER-диаграммы:



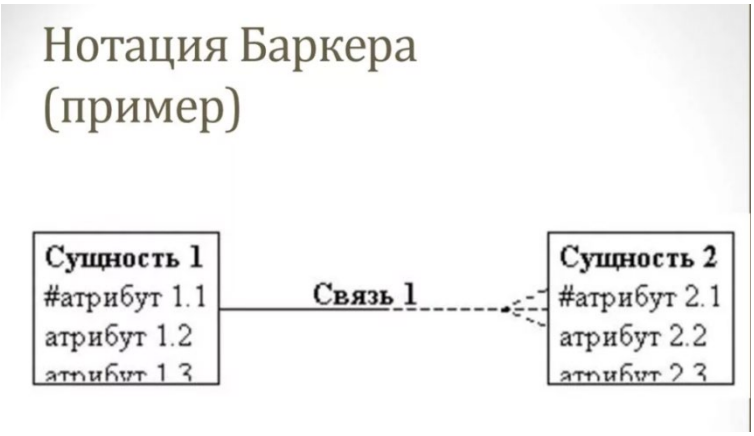
21. Моделирование связей между объектами: ER-диаграммы: нотация Баркера

Нотация Баркера (Barker's Notation) популярна благодаря своей простой визуализации и часто используется в анализе и проектировании информационных систем.

Нотация Баркера:

- **Сущности** представлены прямоугольниками.
- **Атрибуты** в данной нотации обычно не отображаются на основной диаграмме (они могут быть указаны в дополнительных документах).
- **Связи** обозначаются линиями, соединяющими сущности. Концы линий обозначают тип связи и кардинальность.
- **Кардинальность** обозначается прямо на концах линий:
  - Кольцо (O) обозначает, что участие необязательно (ноль или больше).
  - Черта (|) обозначает, что участие обязательно (один или больше).

Пример ER-диаграммы:



:

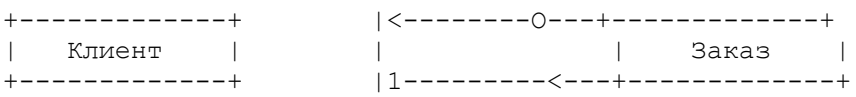
22. Моделирование связей между объектами: ER-диаграммы: нотация «Вороньи лапки»

Нотация «Вороньи лапки» (Crow's Foot Notation) широко используется для ER-диаграмм благодаря своей простоте и ясности в представлении кардинальности.

Нотация «Вороньи лапки»:

- **Сущности** изображаются прямоугольниками.
- **Атрибуты** могут отображаться в прямоугольниках сущностей или в виде отдельных овалов.
- **Связи** изображаются линиями с «вороньими лапками» для обозначения «многие» (N) и одиночными линиями для «один» (1).
- **Кардинальность** обозначается следующими символами:
  - Прямая линия (|) обозначает один.
  - Кружок (O) обозначает ноль.
  - «Вороньи лапки» (иногда представляют как >-) обозначают много.

Пример ER-диаграммы:



В данном примере:

- Один «Клиент» может иметь много «Заказов».
- Каждый «Заказ» относится к одному и только одному «Клиенту».

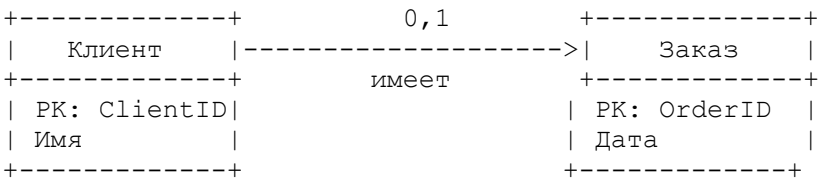
23. Моделирование связей между объектами: ER-диаграммы: нотация IDEF1X

Нотация IDEF1X (Integration Definition for Information Modeling) — стандарт, разработанный для моделирования данных и структурирования информации, широко применяемый в промышленности и правительственных организациях.

Нотация IDEF1X:

- **Сущности** изображаются прямоугольниками. Прямоугольник разделен на два или три блока: имя сущности, ключевые атрибуты и неключевые атрибуты.
- **Атрибуты** отображаются внутри сущностей.
- **Связи** изображаются линиями с круглыми и квадратными концами для обозначения кардинальности.
- **Кардинальность** обозначается как 0, 1 или N на концах линий.

Пример ER-диаграммы:



В данном примере:

- Один «Клиент» может иметь ноль или один «Заказ».
- «Заказ» всегда связан с одним «Клиентом».

24. Расширения модели «сущность-связь»: уточнение/обобщение, агрегирование, композиция. Определение суперклассов и подклассов.

Расширения модели «сущность-связь» включают концепции, которые позволяют более детально моделировать сложные структуры данных.

Уточнение и обобщение:

- **Обобщение** (Generalization): объединение нескольких похожих сущностей в одну общую (например, сущности «Сотрудник» и «Контрактный Работник» можно объединить в общую сущность «Работник»).

- **Уточнение** (Specialization): разделение одной сущности на более специфичные подтипы (например, «Транспортное средство» можно разделить на «Автомобиль», «Мотоцикл» и «Велосипед»).

Агрегирование:

- Агрегирование — это процесс создания связи между сущностью и другим набором сущностей или связей. Это полезно для моделирования ситуаций, где один объект состоит из других объектов.

Пример: Курс обучения, который состоит из нескольких модулей.

Композиция:

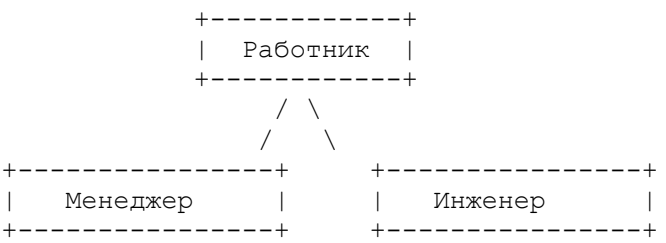
- Композиция — это сильная форма агрегирования, указывающая на жесткую зависимость части от целого. Удаление целого влечет за собой удаление всех частей.

Пример: Составляющие компоненты компьютера (жесткий диск, оперативная память) зависят от существования самого компьютера.

Определение суперклассов и подклассов:

- **Суперкласс** — это общая сущность, которая содержит общие атрибуты и связи (например, «Работник»).
- **Подкласс** — это сущность, которая наследует атрибуты и связи суперкласса, но также имеет свои собственные специфичные атрибуты (например, «Менеджер» и «Инженер» как подклассы «Работника»).

Пример:



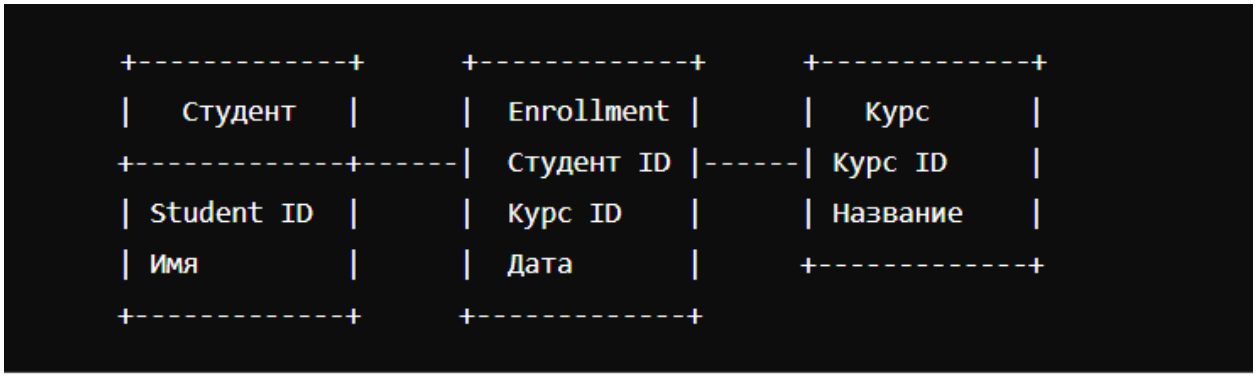
25. Дополнительные действия со связями. Определение ассоциативных связей.

Ассоциативные связи (или сущности-связи) используются для моделирования связей «многие ко многим» и хранения дополнительной информации о связях.

Ассоциативные связи:

- Ассоциативные связи представляются как сущности с отношениями.
- Эти сущности могут содержать атрибуты, которые описывают связь между двумя другими сущностями.

Пример: Таблица «Enrollment» (Запись на курс) для моделирования связи «многие ко многим» между студентами и курсами.



26. Определение неперемещаемых, иерархических, рекурсивных и дуговых связей.

Неперемещаемые связи (Non-Transferable Relationships):

- Указывают, что объект не может быть связан с другим объектом после создания первоначальной связи.

Пример: Связь между пациентом и его медицинской картой — карта пациента не может быть передана другому пациенту.

Иерархические связи (Hierarchical Relationships):

- Отражают структуры типа «один ко многим» или «многие к одному», где одна сущность является родительской для других сущностей.

Пример: Организационная структура компании с отделами и подотделами.

Рекурсивные связи (Recursive Relationships):

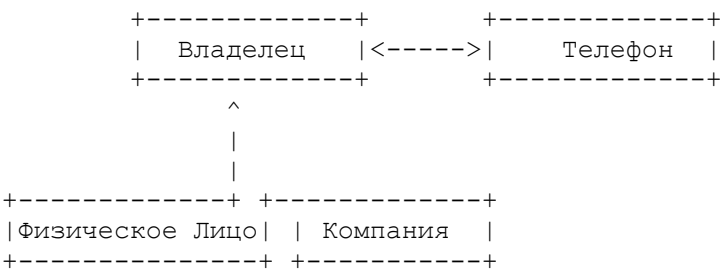
- Отображают связи сущности самой с собой.

Пример: Структура управления, где один сотрудник подчиняется другому сотруднику.

Дуговые связи (Arc Relationships):

- Используются для представления альтернативных связей, когда сущность может быть связана с одним из нескольких объектов, но не одновременно с двумя или более.

Пример: Мобильный телефон может быть зарегистрирован только на одного владельца — физическое лицо или компанию, но не одновременно на обоих.



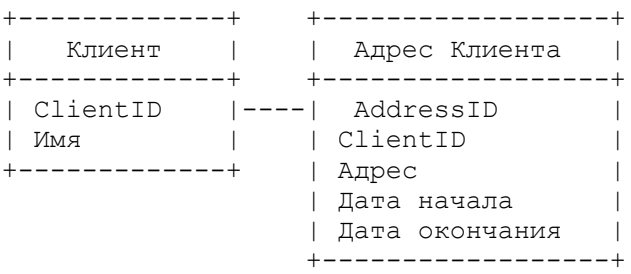
27. Моделирование данных на протяжении времени.

Моделирование данных на протяжении времени (temporal data modeling) включает учет временных аспектов в данных, что позволяет отслеживать изменения состояния объектов или связей с течением времени.

Временные аспекты данных:

- **Временные отметки** (timestamps): указывают время начала и/или окончания действия.
- **Исторические данные**: хранят прошлые состояния данных.
- **Временные интервалы**: отображают диапазоны времени для атрибутов или связей.

Пример: Отслеживание изменения адреса клиента:



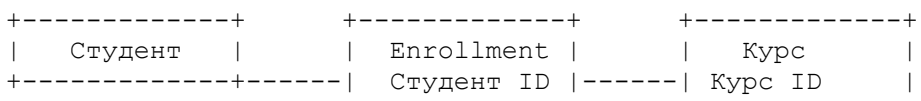
28. Решение связей типа N

. Решение рекурсивных связей.

Решение связей типа N:M:

Для моделирования связей типа «многие ко многим» используется промежуточная таблица (или сущность), называемая ассоциативной сущностью.

Пример: Связь между студентами и курсами:



```
| Student ID  |      |  Курс ID    |      | Название    |
| Имя         |      |  Дата       |      +-------------+
+-------------+      +-------------+
```

Решение рекурсивных связей:

Для моделирования рекурсивных связей добавляется внешняя ссылка на первичный ключ той же таблицы.

Пример: Связь между сотрудниками и их руководителями:

```
+-------------+
|  Сотрудник  |
+-------------+
| EmployeeID  |
| Имя         |
| ManagerID   |  -> Внешний ключ, ссылающийся на EmployeeID
+-------------+
```

29. Логическое проектирование. Удаление связей с атрибутами. Удаление сложных, избыточных связей. Удаление многозначных атрибутов.

Логическое проектирование:

- **Логическое проектирование** фокусируется на построении абстрактной модели данных, отражающей бизнес-требования.

Удаление связей с атрибутами:

- Если связь имеет собственные атрибуты, она превращается в ассоциативную сущность.

Пример: Связь между «Студентом» и «Курсом», содержащая дату записи, превращается в сущность «Enrollment».

Удаление сложных, избыточных связей:

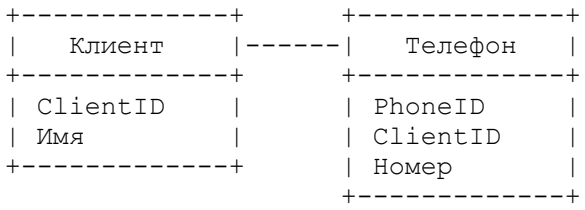
- Убираются связи, которые дублируют информацию или создают сложности без необходимости.

Пример: Если у нас есть прямые связи между «Клиентом» и «Продуктом», а также между «Клиентом» и «Заказом», и «Заказом» и «Продуктом», то прямую связь между «Клиентом» и «Продуктом» можно удалить как избыточную.

Удаление многозначных атрибутов:

- Многозначные атрибуты (когда один атрибут может содержать несколько значений) преобразуются в отдельные сущности.

Пример: Если у «Клиента» есть несколько телефонных номеров, атрибут «Телефон» преобразуется в отдельную таблицу «Телефон».



30. Переход к логической модели: правила формирования отношений.

Правила формирования отношений:

1. **Идентификация сущностей**: Каждая сущность должна иметь уникальный идентификатор (первичный ключ).
2. **Определение атрибутов**: Все атрибуты сущностей должны быть определены и соответствовать реальным бизнес-требованиям.
3. **Определение связей**: Все связи между сущностями должны быть четко определены, включая кардинальность и обязательность.
4. **Использование внешних ключей**: Внешние ключи устанавливают связи между сущностями и обеспечивают целостность данных.
5. **Нормализация данных**: Удаление избыточности и предотвращение аномалий данных через нормализацию.
6. **Определение бизнес-правил**: Учитываются все бизнес-правила, которые должны соблюдаться в данных.

31. Переход к физической модели. Преобразование логической модели в реляционную.

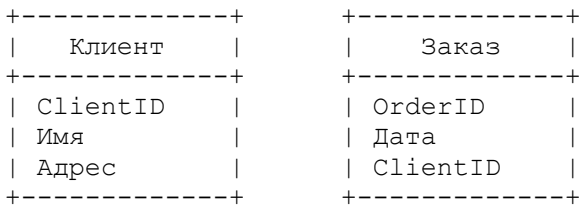
Переход к физической модели:

- **Физическая модель** включает создание реальной структуры базы данных с учетом выбранной СУБД.

Преобразование логической модели в реляционную:

1. **Создание таблиц** для каждой сущности.
2. **Определение полей** (столбцов) таблиц, включая типы данных и ограничения.
3. **Установка первичных ключей** для уникальной идентификации записей.
4. **Установка внешних ключей** для обеспечения целостности данных между таблицами.
5. **Создание индексов** для оптимизации запросов.
6. **Определение связей** и кардинальности между таблицами.

Пример преобразования логической модели:



32. Соглашение имен базы данных: применение правил именования объектов, используемых в физических моделях.

Правила именования объектов:

1. **Использование понятных и описательных имен**: Имена должны четко отражать содержание и назначение объекта.
2. **Единообразие**: Применение единого стиля именования для всех объектов (таблиц, колонок, индексов и т.д.).
3. **Избегание зарезервированных слов**: Нельзя использовать зарезервированные слова СУБД.
4. **Использование префиксов или суффиксов**: Может помочь в идентификации типов объектов (например, tbl_, idx_).
5. **Поддержка длины имен**: Учитывание ограничений длины имен в конкретной СУБД.
6. **Избегание спецсимволов и пробелов**: Использование только букв, цифр и подчеркиваний.

Пример соглашения имен:

- Таблицы: `tbl_Customers`, `tbl_Orders`
- Столбцы: `CustomerID`, `OrderDate`
- Индексы: `idx_Customers_LastName`

33. Хранимые процедуры и функции. Операторы создания и использования процедур и функций.

Хранимые процедуры:

- **Хранимая процедура** — это набор SQL-запросов, сохраненных на сервере и выполняемых как единое целое.
- Используются для выполнения повторяющихся операций, уменьшения трафика между клиентом и сервером и повышения безопасности.

Создание хранимой процедуры:

```
sql

CREATE PROCEDURE GetCustomerOrders
    @CustomerID INT
AS
BEGIN
    SELECT * FROM Orders WHERE CustomerID = @CustomerID;
END;
```

Использование хранимой процедуры:

```
sql

EXEC GetCustomerOrders @CustomerID = 1;
```

Функции:

- **Функция** — это программный блок, который возвращает значение и может использоваться в выражениях SQL.
- Отличаются от процедур тем, что всегда возвращают результат.

Создание функции:

```
sql

CREATE FUNCTION GetCustomerBalance(@CustomerID INT)
RETURNS DECIMAL(10, 2)
AS
BEGIN
    DECLARE @Balance DECIMAL(10, 2);
    SELECT @Balance = SUM(Amount) FROM Payments WHERE CustomerID =
@CustomerID;
    RETURN @Balance;
END;
```

Использование функции:

```
sql

SELECT dbo.GetCustomerBalance(1);
```

34. Триггеры

Триггеры:

- **Триггеры** — это специальные виды хранимых процедур, которые автоматически выполняются в ответ на определенные события в таблице или представлении (INSERT, UPDATE, DELETE).
- Используются для обеспечения целостности данных, автоматизации аудита и других задач.

Создание триггера:

```
sql

CREATE TRIGGER trgAfterInsert
ON Customers
AFTER INSERT
AS
BEGIN
    PRINT 'Record inserted into Customers table';
END;
```

Использование триггера:

Триггер активируется автоматически при добавлении записи в таблицу `Customers`:

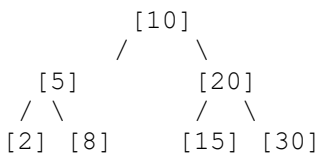
```
sql

INSERT INTO Customers (CustomerID, Name) VALUES (1, 'John Doe');
```

35. Индексные структуры: B-деревья, битовые карты, другие виды индексов

Индексные структуры:

1. **B-деревья (B-trees)**:
   - Широко используются для индексации данных.
   - Обеспечивают сбалансированную и быструю навигацию.
   - Поддерживают упорядоченное хранение и быстрый поиск.
2. **Битовые карты (Bitmap Indexes)**:
   - Эффективны для столбцов с небольшим числом уникальных значений.
   - Используют битовые маски для представления наличия значений.
   - Хороши для запросов с множеством условий (например, в хранилищах данных).
3. **Другие виды индексов**:
   - **Хэш-индексы**: Быстрый доступ к данным по ключу, но не поддерживают упорядоченные запросы.
   - **Функциональные индексы**: Индексация результатов выражений или функций.
   - **Геопространственные индексы**: Поддержка поиска по географическим данным.

Пример B-дерева:



36. Создание индекса, удаление индекса. Примеры запросов на SQL.

Создание индекса:

```
sql

CREATE INDEX idx_CustomerName
ON Customers (Name);
```

Удаление индекса:

```
sql

DROP INDEX idx_CustomerName ON Customers;
```

37. Уникальные индексы

Уникальные индексы:

- Гарантируют, что значения в столбце или комбинации столбцов будут уникальными.
- Используются для обеспечения уникальности данных (например, уникальный email пользователя).

Создание уникального индекса:

```
sql

CREATE UNIQUE INDEX idx_UniqueEmail
ON Users (Email);
```

38. Составные индексы

Составные индексы:

- Индексы, создаваемые на нескольких столбцах.
- Улучшают производительность запросов, фильтрующих или сортирующих по комбинации столбцов.

Создание составного индекса:

```
sql

CREATE INDEX idx_OrderCustomer
ON Orders (CustomerID, OrderDate);
```

39. Индексы по выражениям. Частичные индексы.

Индексы по выражениям:

- Индексация вычисляемых значений выражений или функций.

Пример создания индекса по выражению:

```
sql

CREATE INDEX idx_LowerCaseName
ON Employees (LOWER(LastName));
```

Частичные индексы:

- Индексы, создаваемые только для подмножества строк таблицы.

Пример создания частичного индекса:

```
sql

CREATE INDEX idx_ActiveOrders
ON Orders (OrderDate)
WHERE Status = 'Active';
```

40. Индексы и порядок соединений

Индексы и порядок соединений:

- Индексы влияют на порядок выполнения соединений и могут значительно улучшить производительность запросов с JOIN.

Пример:

```
sql

SELECT e.Name, d.Name
FROM Employees e
JOIN Departments d ON e.DepartmentID = d.DepartmentID
WHERE d.Location = 'New York';
```

Создание индексов на `Employees.DepartmentID` и `Departments.DepartmentID` может ускорить выполнение данного запроса.

41. Алгоритмы доступа к данным: полное (последовательное) сканирование, сканирование на основе битовой карты.

Полное (последовательное) сканирование:

- Чтение всех строк в таблице.
- Используется, когда индекс отсутствует или запрос должен обработать все строки.

Сканирование на основе битовой карты:

- Быстрое сканирование столбцов с использованием битовых масок.
- Подходит для столбцов с низкой кардинальностью (например, «пол» или «статус»).

Пример битовой карты:

```
Статус:   ['Active', 'Inactive', 'Pending']
Битовая карта: 101 (Active), 010 (Inactive), 001 (Pending)
```

42. Алгоритмы доступа к данным: доступ к таблицам на основе индексов, сканирование только индекса.

Доступ к таблицам на основе индексов:

- Использование индексов для быстрого поиска строк.
- Особенно эффективно для запросов с условиями (WHERE) и JOIN.

Сканирование только индекса:

- Выполнение запросов, когда все необходимые данные могут быть получены из индекса без доступа к основной таблице.

Пример:

```
sql

SELECT OrderID, OrderDate
FROM Orders
WHERE CustomerID = 1;
```

Если индекс создан по `CustomerID`, `OrderID` и `OrderDate`, запрос может быть выполнен только по индексу.

43. Способ соединения наборов строк: вложенный цикл, хеширование.

Вложенный цикл (Nested Loop Join):

- Итерирует по каждой строке первой таблицы и ищет совпадения во второй таблице.

- Эффективен для небольших наборов данных или когда одна таблица мала.

Пример:

```
FOR each row in Table1
    FOR each row in Table2
        IF Table1.key = Table2.key THEN
            OUTPUT row
```

Хеширование (Hash Join):

- Строит хеш-таблицу для одной таблицы и использует ее для поиска совпадений во второй таблице.
- Эффективен для больших наборов данных.

Пример:

```
1. Построить хеш-таблицу по ключам из первой таблицы.
2. Проверить каждую строку из второй таблицы на соответствие в хеш-таблице.
```

44. Способ соединения наборов строк: слияние.

Слияние (Merge Join):

- Обе таблицы сортируются по ключу соединения, затем соединение выполняется путем сканирования отсортированных наборов данных.
- Эффективен для предварительно отсортированных данных или когда сортировка может быть выполнена быстро.

Пример:

```
1. Отсортировать обе таблицы по ключу.
2. Итерировать по обеим таблицам одновременно и соединять совпадающие строки.
```

45. Планы выполнения запроса.

Планы выполнения запроса:

- План выполнения запроса (query execution plan) показывает, как SQL сервер выполняет запрос.
- Используется для анализа и оптимизации производительности запросов.

Пример запроса на отображение плана выполнения:

```
sql

EXPLAIN SELECT * FROM Orders WHERE CustomerID = 1;
```

План выполнения включает:

- Порядок выполнения операций.
- Методы доступа к данным (сканирование, индексирование).
- Оценки затрат на каждую операцию.

46. Управление параллелизмом.

Управление параллелизмом:

- **Параллелизм** позволяет выполнять несколько операций одновременно, улучшая производительность.
- Включает механизмы блокировок и согласованности для предотвращения конфликтов.

Методы управления параллелизмом:

1. **Блокировки**: Защита данных от конфликтов с помощью блокировок (эксклюзивные, разделяемые).
2. **Транзакции**: Обеспечение целостности данных через ACID-свойства транзакций.
3. **Планировщик задач**: Управление порядком выполнения параллельных операций.
4. **Версионность**: Использование версий данных для поддержки параллельных операций без блокировок.

47. Изоляция транзакций: уровни изоляции.

Уровни изоляции транзакций:

1. **Read Uncommitted**:
   - Транзакция может читать незавершенные изменения других транзакций.
   - Возможны «грязные» чтения.
2. **Read Committed**:
   - Транзакция читает только завершенные изменения других транзакций.
   - Защита от «грязных» чтений.
3. **Repeatable Read**:
   - Гарантируется, что данные, прочитанные в транзакции, не изменятся до её завершения.
   - Предотвращение «неповторяющихся» чтений.
4. **Serializable**:
   - Самый высокий уровень изоляции.
   - Транзакции исполняются как если бы они были последовательными.
   - Предотвращение «фантомных» чтений.

Пример установки уровня изоляции:

```
sql

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
BEGIN TRANSACTION;
-- SQL operations
COMMIT;
```

48. Управление параллелизмом. Управление блокировками: виды блокировок.

Виды блокировок:

1. **Эксклюзивная блокировка (X)**:
    - Запрещает другим транзакциям доступ к заблокированным данным.
    - Используется для операций записи.
2. **Разделяемая блокировка (S)**:
    - Позволяет другим транзакциям читать данные, но запрещает запись.
    - Используется для операций чтения.
3. **Обновляемая блокировка (U)**:
    - Предотвращает «дедлоки» при обновлении данных.
    - Разрешает только одну транзакцию иметь обновляемую блокировку до завершения обновления.
4. **Блокировка намерения (IX, IS)**:
    - Указывает намерение транзакции установить более строгие блокировки на уровне таблицы или страницы.

Управление блокировками:

- **Эскалация блокировок**: Автоматический переход от блокировки строк к блокировке страницы или таблицы при большом числе заблокированных строк.
- **Истечение времени блокировки**: Установка тайм-аутов для предотвращения бесконечного ожидания блокировок.
- **Мониторинг блокировок**: Анализ и управление текущими блокировками для оптимизации работы базы данных.

49. Блокировки: уровни блокировок. Продолжение транзакций.

Уровни блокировок:

1. **Строка (Row)**:
    - Самый низкий уровень блокировки.
    - Позволяет высокую степень параллелизма, но требует много ресурсов.
2. **Страница (Page)**:
    - Блокировка группы строк (обычно 8 KB).
    - Компромисс между параллелизмом и ресурсами.
3. **Таблица (Table)**:
    - Блокировка всей таблицы.
    - Используется при крупных обновлениях или когда параллелизм не важен.
4. **База данных (Database)**:
    - Блокировка всей базы данных.
    - Редко используется, обычно при административных операциях.

Продолжение транзакций:

- **Транзакции** могут быть приостановлены и продолжены после разрешения блокировок или ожидания ресурсов.

Пример продолжения транзакции:

```
sql

BEGIN TRANSACTION;
-- SQL operations
WAITFOR DELAY '00:00:05'; -- Wait for 5 seconds
COMMIT;
```

50. Ведение журнала транзакций. Чекпойнты.

Ведение журнала транзакций:

- Журнал транзакций записывает все изменения данных для обеспечения восстановления и отката транзакций.
- Включает записи о начальных, измененных и окончательных состояниях данных.

Чекпойнты:

- **Чекпойнт** — это момент, когда все текущие изменения данных записываются на диск.
- Используются для ускорения восстановления после сбоя и уменьшения объема журнала транзакций.

Пример выполнения чекпойнта:

```
sql

CHECKPOINT;
```

51. Протокол двухфазной фиксации.

Протокол двухфазной фиксации (2PC):

- Обеспечивает атомарность распределенных транзакций через двухэтапный процесс:
  - **Фаза подготовки** (Prepare): Все участники готовятся к выполнению транзакции и сообщают о своей готовности.
  - **Фаза фиксации** (Commit): Если все участники готовы, транзакция фиксируется; иначе, выполняется откат.

Пример выполнения 2PC:

```
1. Координатор отправляет запрос на подготовку (Prepare) всем участникам.
2. Участники выполняют локальные операции и сообщают «готов» или «не готов».
3. Если все «готовы», координатор отправляет запрос на фиксацию (Commit).
4. Участники фиксируют свои изменения.
```

52. Архитектура серверной обработки базы данных. Компоненты сервера базы данных.

Компоненты сервера базы данных:

1. **Планировщик запросов**:
   - Разрабатывает план выполнения запросов и оптимизирует их.
2. **Менеджер памяти**:
   - Управляет распределением и использованием памяти для кэша данных и выполнения операций.
3. **Менеджер файлов**:
   - Управляет физическим хранением данных на диске.
4. **Контроллер транзакций**:
   - Управляет началом, фиксацией и откатом транзакций.

5. **Система управления блокировками**:
    - Управляет блокировками для обеспечения целостности данных.
6. **Журнал транзакций**:
    - Записывает изменения данных для обеспечения восстановления и отката.
7. **Модуль безопасности**:
    - Управляет доступом пользователей и обеспечивает защиту данных.
8. **Менеджер соединений**:
    - Управляет клиентскими соединениями и распределяет запросы к базе данных.

Пример архитектуры серверной обработки: