SQL , общий индекс на несколько таблиц:

1, решение 1

Я использую основную таблицу с общими свойствами и основным ID, на которую посредством foreign key ссылаются таблицы детализирующие разные типы объектов. Кроме того есть таблица описывающая каждый возможный тип объекта, включающая имя таблицы для данного типа и имена VIEW исполльзующихся в общих отчетах. В этих таблицах как PRIMARY KEY используется тот же ключ что и в общей.  
В принципе можно сделать VIEW который покажет все поля всех типов данных - только имхо это \_никогда\_ не нужно. (Что толку ГЛОБАЛЬНО искать если не знаешь, в каком поле искать-то?)

* [[https://habrastorage.org/r/w48/getpro/habr/avatars/e24/f0a/ed5/e24f0aed57701bdb72ec4a7252e16667.jpg](https://habr.com/ru/users/Vile55/)**Vile55**](https://habr.com/ru/users/Vile55/) 27 июня 2008 в 02:40

**0**

А, и еще один вариант, который используется в том же продукте - это таблица со свойствами объектов (свойств оных много, они все разные, и вообще могут различаться даже для объектов одного типа).   
Вот для них используется таблица с тремя основными полями - ID\_объекта, ID\_свойства, Значение\_свойства\_в\_виде\_XML\_данных

2, решение 2 –

Мне оптимальным кажется наследование объектов, как в Postgres.  
  
И вообще — задача, о которой о сути идёт речь — организация хранения наследования. Рассмотрено, кстати, детально в книге Роберта Мюллера «Базы данных и UML. Проектирование». А не это неуклюжее «один к разным».

3. решение – PosgreSQL специфичное но не работает тк нет уникальности индекса

<https://www.postgresql.org/docs/8.3/ddl-inherit.html>

4. Из своего опыта работы с разными СУБД могу сказать, что всегда нужно учитывать особенностии конкретной СУБД. Решение SQL задач из общих соображений (без учета особенностей СУБД) к практической реализации имеет мало отношения. Более того в постановке задачи ничего не сказано о профиле данных (частоте изменений свойств обьектов, частоте появления новых обьектов, частоте появления новых типов обьектов и т.д.) Если решать SQL зачади в самом общем виде, то естесвенно надо полагаться на теорию реляционных баз данных, а не выдумывать велосипед. Но в большинстве случаев это не выгодно с точки зрения производительлности.  
  
Примеры:   
Postgres:   
Особенность: на каждый update поля в строке в действительностии добавляется новая строка, старая остается не видимой для сессии.   
Это особенность реализации версионности. Поэтому для Postgres'а оптимально разделять таблицы на более мелкие (вертикальное партицирование). Где в одной будут содержаться редко изменяющиеся поля и таких полей может быть много, в другой часто изменяющиеся и таких полей в одной аблице должно быть как можно меньше. Причем таким образом можно делить даже одну сущность (например "корабль"), что уже не верно с точки зрения реляционной теории.  
Oracle:   
Особенность: Наличие возможности более гибко влиять на построение плана запроса(oracle hints). Что дает возможность лучше подстраиваться под профиль данных и оптимизировать сложные запросы. Можно выбирать решения больше удовлетворяющие реляционным бд, что дает большую гибкость при выборке данных.   
Особенность: наличие очень хорошей имплементации bitmap индексов, сущесвенно ускоряющих выборку по однородным данным и уменьшающим обьем хранимых индексов. Это приводит к необходимости создания буферов. Например в буфферные таблицы с разных клиентов складывается информация об изменении координат обьектов, раз в минуту все данные забираются из буферов и append'ом (оракловый метод вставки большого кол-ва строк) вставляются в основное хранилище, что дает оптимальное построение bitmap индексов.

5. Нормализованная БД: (**Наши объекты:** object\_id, name) -< (**Наш космос:** id, x, y, object\_id) -< (**Свойства объектов:** id, var\_id, value) >- (**Наши свойства:** var\_id, name[, type]).  
  
Далее можно перейти к денормализации БД за счет избыточности названий объектов и их свойств : (**Наш космос:** id, x, y, object\_name) -< (**Свойства объектов:** id, name[, type,] value).  
  
Денормализация за счет использования XML (PostgresSQL хвалят) и JSON:  
(**Наш космос:** id, x, y, object\_name, name[, type,] xml\_or\_json\_value).  
  
Нужно учесть функциональность ПО а именно, к каким полям будут выполняться запросы, т.к. в случае с использованием XML или JSON, нужно индексировать большое количество данных и осуществлять медленный поиск (xml\_or\_json\_value LIKE '%значение%').  
  
P.S. Есть множество решений построения архитектуры БД. Для достижения высокой производительности и максимальной зажатости данных, нужно рассматривать впервую очередь поведения самого ПО, на вооружение можно взять примеры UML-диаграмм. И не забываем про постреляционные БД.

6. решение:

одна таблица, которая хранит сущности, которые имеют свойства: тип, идентификатор внутри типа, имя, описание, дату создания, модификации и тп  
одна таблица, которая хранит "космические объекты", которые имеют такие свойства как масса, координаты, ориентация и тп. каждый объект идентифицируется идентификатором+типом  
одна таблица, которая хранит "звёзды", которые характеризуются... ну чем-то там характеризуются ;-) идентификатор в этой таблице является уникальным, а поле "тип" всегда равно "звезда" или вообще отсутствует.  
  
итого, информация о кажом объекте распределена по трём таблицам: сущность, космический объект, звезда. связь 1 к 1 по идентификатору и типу.

Итого: одно из лучших:

Я использую основную таблицу с общими свойствами и основным ID, на которую посредством foreign key ссылаются таблицы детализирующие разные типы объектов. Кроме того есть таблица описывающая каждый возможный тип объекта, включающая имя таблицы для данного типа и имена VIEW исполльзующихся в общих отчетах. В этих таблицах как PRIMARY KEY используется тот же ключ что и в общей.  
В принципе можно сделать VIEW который покажет все поля всех типов данных - только имхо это \_никогда\_ не нужно. (Что толку ГЛОБАЛЬНО искать если не знаешь, в каком поле искать-то?)

* [[https://habrastorage.org/r/w48/getpro/habr/avatars/e24/f0a/ed5/e24f0aed57701bdb72ec4a7252e16667.jpg](https://habr.com/ru/users/Vile55/)**Vile55**](https://habr.com/ru/users/Vile55/) 27 июня 2008 в 02:40

**0**

А, и еще один вариант, который используется в том же продукте - это таблица со свойствами объектов (свойств оных много, они все разные, и вообще могут различаться даже для объектов одного типа).   
Вот для них используется таблица с тремя основными полями - ID\_объекта, ID\_свойства, Значение\_свойства\_в\_виде\_XML\_данных