1. Основные понятия реляционной алгебры.

*Лучше сначала прочитать оригинал(http://habrahabr.ru/post/145381/). Там картиночки и читабельнее***.**

Реляционной базой данных называется совокупность отношений, содержащих всю информацию, которая должна храниться в базе.

Каждая строка в таблице является кортежем в реляционной теории. Множество упорядоченных кортежей называется отношением. Применительно к таблице домены – это столбцы.

Строгое определение отношения: Пусть даны N множеств D1,D2, …. Dn (домены), отношением R над этими множествами называется множество упорядоченных N-кортежей вида <d1,d1,...dn>, где d1 принадлежит D1 и тд. Множества D1,D2,..Dn называются доменами отношения R.

Каждый элемент кортежа представляет собой значение одного из атрибутов соответствующего одному из доменов.

В отношении требованием является то, что все кортежи должны различаться. Для однозначной идентификации кортежа существует первичный ключ. Первичный ключ – это атрибут или набор из минимального числа атрибутов, который однозначно идентифицирует конкретный кортеж и не содержит дополнительных атрибутов.

В реляционной БД таблицы взаимосвязаны и соотносятся друг с другом как главные и подчиненные. Связь главной и подчиненной таблицы осуществляется через первичный ключ главной таблицы и внешний ключ подчиненной таблицы.

Основные восемь операций реляционной алгебры были предложены Э.Коддом: объединение, пересечение, вычитание, декартово произведение, выборка, соединение, деление.

Для начала рассмотрим самую простую операцию — имя отношения. Её результатом будет такое же отношение, то есть выполнив операцию PRODUCTS, мы получим копию отношения PRODUCTS.

Проекция является операцией, при которой из отношения выделяются атрибуты только из указанных доменов, то есть из таблицы выбираются только нужные столбцы, при этом, если получится несколько одинаковых кортежей, то в результирующем отношении остается только по одному экземпляру подобного кортежа.

Выборка — это операция, которая выделяет множество строк в таблице, удовлетворяющих заданным условиям. Условием может быть любое логическое выражение.

Умножение или декартово произведение является операцией, производимой над двумя отношениями, в результате которой мы получаем отношение со всеми доменами из двух начальных отношений. Кортежи в этих доменах будут представлять из себя все возможные сочетания кортежей из начальных отношений.

Операция соединения обратна операции проекции и создает новое отношение из двух уже существующих. Новое отношение получается конкатенацией кортежей первого и второго отношений, при этом конкатенации подвергаются отношения, в которых совпадают значения заданных атрибутов. Естественное (натуральное) соединение отличается лишь тем, что при соединении двух таблиц, например, по ID, в результирующем отношении будет один столбец ID.

Результатом операции пересечения будет отношение, состоящее из кортежей, полностью входящих в состав обоих отношений.

Результатом вычитания будет отношение, состоящее из кортежей, которые являются кортежами первого отношения и не являются кортежами второго отношения.

2. Архитектура системы баз данных, блоки данных, сегменты, экстенты.

Лучше всего оригинальная документация:
http://docs.oracle.com/cd/B19306_01/server.102/b14220/logical.htm#g23847

Можно ещё вот тут вот почитать: http://citforum.ru/database/oracle/kyte/02.shtml
Всё довольно-таки няшно и понятно расписано.

Oracle – система управления базами данных, которая предоставляет открытый, всеобъемлющий, интегрированный подход к управлению информацией. Состоит из экземпляра и базы данных.

Экземпляр – средство доступа к базе данных Oracle. Состоит из набора процессов операционной системы и используемой ими памяти. Открывает только одну базу данных в один момент времени.

Физическая структура БД Oracle определяется файлами операционной системы, которые обеспечивают физическое хранение информации в базе данных. (Control files, Data files, Redo log files). Логическая структура представлена иерархией: tablespaces, segments, extends, blocks.

SYSTEM tablespace. Создается вместе с базой данных. Содержит словарь данных. Содержит сегмент отката SYSTEM. Non-SYSTEM tablespaces. Содержит пользовательские сегменты. Управляются администратором БД.

Типы данных Oracle: пользовательские и встроенные: скалярные (CHAR, VARCHAR2, NUMBER и др.), коллекции (VARRAY, TABLE), отношения (REF).

Таблицы в Oracle: обычные таблицы, секционированные таблицы, организованные по индексу, кластерные.

Redo Log. В журналы повторного выполнения записываются все изменения в базе данных для проведения восстановления в случае сбоя. Redo log объединяются в группы. Необходимо как минимум 2 группы. Данные в файлах внутри группы идентичны. Группы Redo log используются циклично. Когда файлы группы заполнены, LGWR переключается на следующую группу. Это называется переключением логов. В этот момент происходит Checkpoint. Информация записывается в control file.

Undo. Перед тем как сделать изменения в блоке данных Oracle копирует его в Undo. Это необходимо для Rollback и Read Consistency.

Структура памяти Oracle состоит из двух основных областей: System Global Area (SGA): выделяется при старте экземпляра, базовый компонент экземпляра Oracle. Program Global Area (PGA): выделяется при запуске серверного процесса.

SGA состоит из следующих структур: Shared pool, Database buffer cache, Redo log buffer. Опциональные структуры SGA: Large pool, Java pool. PGA – это память, зарезервированная для процессов подключающихся к базе данных.

Взаимодействие между физическими и логическими структурами памяти осуществляется с помощью различных фоновых процессов.

Oracle использует неявную транзакционную модель. Транзакция начинается сразу за окончанием предыдущей. COMMIT или ROLLBACK заканчивают транзакцию. Во время создания базы данных сервер Oracle создает дополнительные объектные структуры в файлах данных: таблицы словаря данных и таблицы dynamic performance.

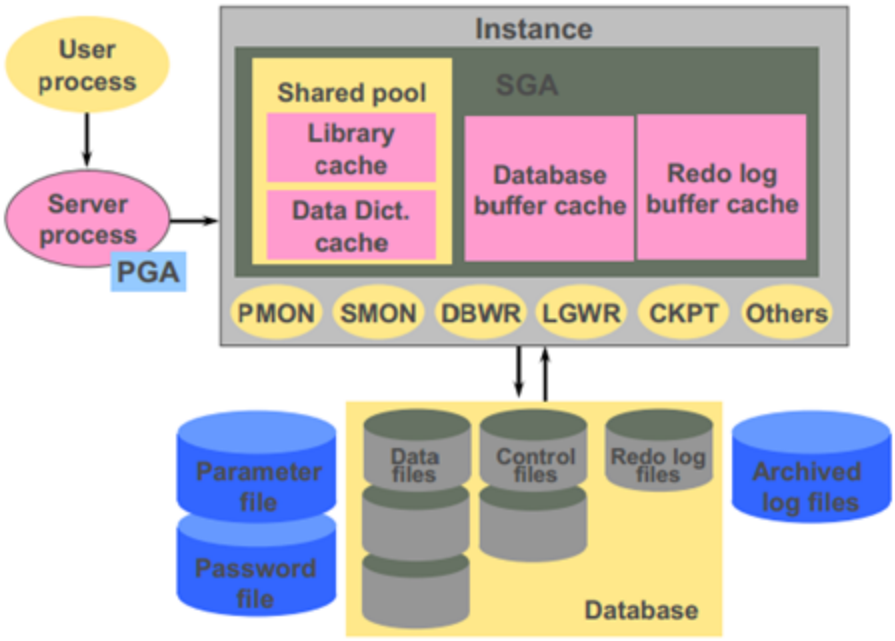
Словарь данных – это набор read-only таблиц и представлений, которые содержат информацию о базе данных. Описывает базу данных и ее объекты. Словарь данных предоставляет

информацию о: логической и физической структуре БД, свойствах и размерах объектов, ограничениях целостности, пользователях, ролях, привилегиях, аудите.

Блоки данных (Data Block) - мельчайший строительный блок базы данных Oracle, состоящий из определенного количества байт на диске. Блок данных Oracle - логический компонент базы данных. Диски на которых располагаются блоки Oracle, сами делятся на блоки данных. Обычно блоки данных диска соответствуют блокам данных Oracle. Размер блока базы данных Oracle устанавливается параметром DB_BLOCK_SIZE в файле init.ora. Размер блока следует воспринимать, как минимальную единицу обновления, выбора или вставки данных.

Экстенты (extent) - это два или более последовательных блоков данных Oracle, представляющий собой единицу выделения места на диске. Когда комбинируется несколько непрерывных блоков данных, они называются экстентом. Когда вы создаете объект базы данных вроде таблицы или индекса, вы выделяете им некоторый начальный объем пространства, называемый начальным экстентом, и, кроме того, указываете размер следующего экстента.

Сегменты (segments) - набор экстентов, которые вы выделяете логической структуре, такой как таблица или индекс (или некоторый другой объект). Набор экстентов формирует следующую более крупную единицу хранения, именуемую сегментом. Oracle называет сегментом все пространство, выделенное любому конкретному объекту базы данных.



3. Реляционная и объектно-реляционная структура данных.

Реляционная база данных – база данных, основанная на реляционной модели данных. Для работы с реляционными БД применяют реляционные СУБД.

Объектно-реляционная база данных – база данных, поддерживающая некоторые технологии, реализующие объектно-ориентированный подход.

Реляционные базы данных используют набор таблиц, представляющих простые данные. Дополнительная или связанная информация хранится в других таблицах. Часто для хранения одного объекта в реляционной базе данных используется несколько таблиц.

Однако работать приходится именно с объектами. Для решения этой проблемы и созданы реляционные системы управления базами данных. Они должны уметь как обрабатывать данные в объектно-ориентированном виде, так и уметь сохранить эти данные в реляционной форме. ORM

(object-relational mapping) – объектно-реляционное отображение, как раз та самая технология программирования. Вообще вопрос не об этом. См. вопрос 15.

Почему же тогда не сделать объектно-ориентированную базу данных? Потому что как-то с 80-х годов так и не удалось. Причины:

- **Популярность**. Под реляционные базы создано множество продуктов, которые необходимо поддерживать. В них уже вложено много денег. А с использованием ООБД разработано сравнительно мало серьезных коммерческих продуктов.

- **Язык запросов и его стандартизация.** В 1986 году был принят первый стандарт SQL-86, который определил всю судьбу реляционных БД. Для объектно-ориентированных баз данных пока стандарта языка запросов нет.

- **Математический аппарат.** Для реляционных БД используется математический аппарат реляционной алгебры. Он объясняет, как должны выполняться основные операции над отношениями в базе данных, доказывает их оптимальность. Для ООБД пока нет такого аппарата.

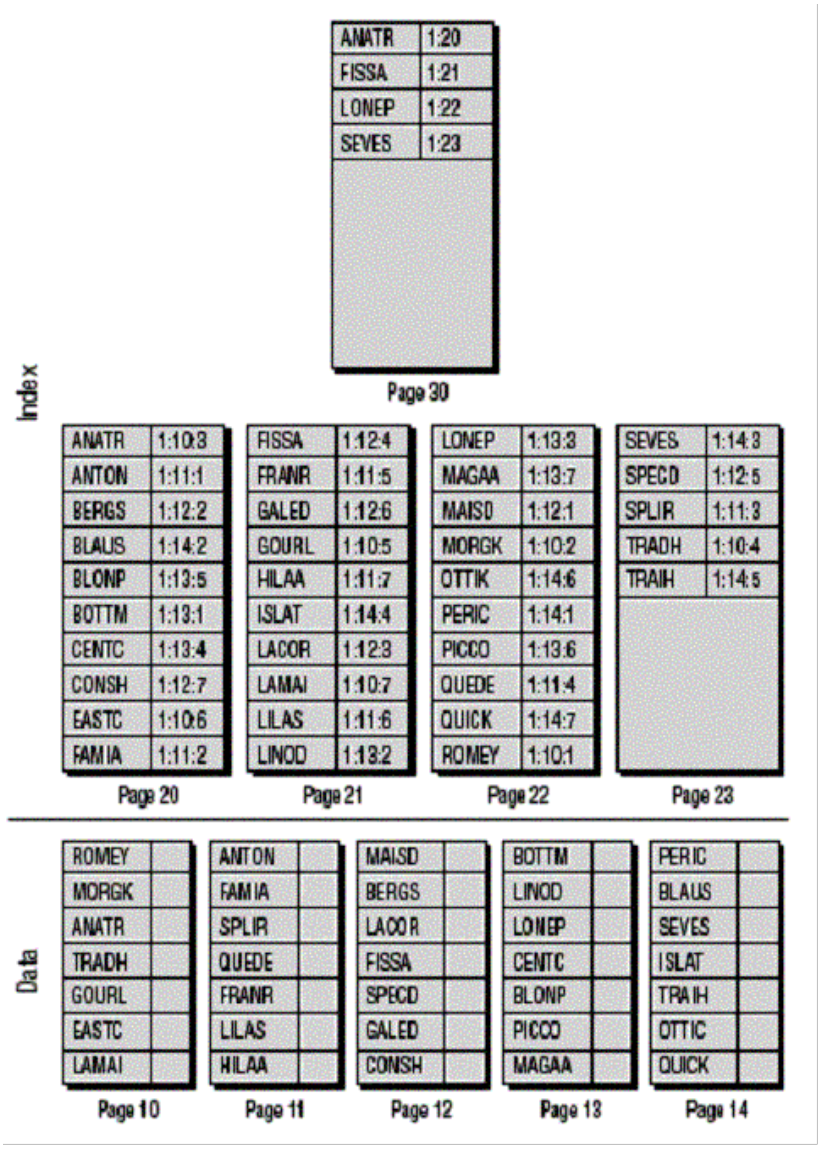
- **Проблема хранения данных и методов.** В реляционных БД хранятся только голые данные. Что с ними будет делать приложение, зависит уже от приложения. В ООБД же, напротив должны храниться объекты, а объект - это совокупность его свойств и методов. Здесь так же нет единого мнения, как ООБД должна осуществлять хранение объектов и как разработчик должен эти объекты разрабатывать и проектировать. Здесь же возникает и проблема хранения иерархии объектов, хранение абстрактных классов и т.п.

4. Методы оптимизации производительности БД, DML-операций (индексы, рефакторинг, настройка, IOT).(Кравцов)

При проектировании и реализации базы данных следует выделить большие таблицы и наиболее сложные процессы. При их разработке необходимо уделить особое внимание производительности. Кроме того, следует определить, какое влияние на производительность будет оказывать увеличивающееся количество пользователей, обращающихся к таблицам.
Ниже продемонстрировано несколько примеров изменения структуры базы данных для улучшения производительности.

- Если требуется ежедневно вычислять сводные данные по таблице, содержащей сотни тысяч строк, в нее можно добавить столбец или столбцы для хранения предыдущих данных, подвергшихся статистической обработке, и использовать их только при подготовке отчета.
- Базы данных могут быть излишне нормализованы. Это означает, что база данных содержит несколько небольших взаимосвязанных таблиц. При обработке данных в этих таблицах требуются дополнительные действия по объединению взаимосвязанных данных. Дополнительная обработка уменьшает производительность базы данных. В этом случае за счет денормализации базы данных можно упростить обработку и слегка увеличить производительность.

Индекс — объект базы данных, создаваемый с целью повышения производительности поиска данных. Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному критерию путем последовательного просмотра таблицы строка за строкой может занимать много времени. Индекс формируется из значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы и, таким образом, позволяет искать строки, удовлетворяющие критерию поиска. Ускорение работы с использованием индексов достигается в первую очередь за счёт того, что индекс имеет структуру, оптимизированную под поиск — например, сбалансированного дерева.



Рефакторинг — это простое изменение в схеме базы данных, которое способствует улучшению её проекта при сохранении функциональной и информационной семантики. Следствием рефакторинга базы данных не может быть добавление новых функциональных возможностей или ограничение уже существующих, равно как и добавление новых данных или же изменение смысла существующих.

Выделяют следующие категории рефакторинга реляционных баз данных:

- Рефакторинг структуры

  Изменения в структуре таблиц или представлений.

Методы: замена связи "один ко многим" ассоциативной таблицей; замена столбца; переименование представления / столбца / таблицы; перемещение столбца; разбиение столбца / таблицы; слияние столбцов / таблиц; удаление представления / столбца / таблицы.

- Рефакторинг качества данных

Изменения, направленные на улучшение качества хранимой в базе данных информации.

Методы: введение заданного по умолчанию значения; введение общего формата; введение ограничения столбца; добавление поисковой таблицы; перемещение данных; преобразование столбца в недопускающий NULL-значения; применение стандартного типа; применение стандартных кодовых обозначений; уничтожение значения, заданного по умолчанию; уничтожение ограничения столбца; уничтожение столбца, не допускающего NULL-значений.

- Рефакторинг ссылочной целостности

Изменения, направленные на поддержание ссылочной целостности в базе данных.

Методы: введение каскадного удаления; введение программного удаления; введение триггера для накопления исторических данных; введение физического удаления; добавление ограничения внешнего ключа; добавление триггера для вычисляемого столбца; уничтожение ограничения внешнего ключа.

- Рефакторинг архитектуры

Изменения, направленные на улучшение взаимодействия внешних программ с базой данных.

Методы: введение вычислительного метода; введение индекса; введение таблицы только для чтения; добавление метода чтения; замена метода (методов) представлением; замена представления методом (методами); инкапсуляция таблицы в представление; перенос метода в базу данных; перенос метода из базы данных.

- Рефакторинг методов

Методы рефакторинга кода, применимые к триггерам и хранимым процедурам.

Index Organized Tables (IOT) - это особый тип таблиц у Oracle. Обычная таблица, создаваемая в вашей базе данных, имеет тип «кучи» – записи навалены кучей вне зависимости от содержимого. Когда записи вставляются, то они помещаются в первый доступный блок, который найдет оракл, не придерживаясь никакого определенного порядка. В индексно-организованных таблицах записи размещаются в порядке определенном первичным ключом, который вы установили.

Пример

```
CREATE TABLE admin_docindex(
      token char(20),
      doc_id NUMBER,
      token_frequency NUMBER,
      token_offsets VARCHAR2(2000),
      CONSTRAINT pk_admin_docindex PRIMARY KEY (token, doc_id))
   ORGANIZATION INDEX
   TABLESPACE admin_tbs
   PCTTHRESHOLD 20
   OVERFLOW TABLESPACE admin_tbs2;
```

http://my-oracle.it-blogs.com.ua/post-408.aspx

5. Нормальные формы (1-6, Бойса-Кодда).

Первая нормальная форма (1NF)

- Все строки должны быть различными.
- Все элементы внутри ячеек должны быть атомарными (не списками). Другими словами, элемент является атомарным, если его нельзя разделить на части, которые могут использоваться в таблице независимо друг от друга.

Методы приведения к 1NF:

- Устраните повторяющиеся группы в отдельных таблицах (одинаковые строки)
- Создайте отдельную таблицу для каждого набора связанных данных

Вторая нормальная форма (2NF)

- Таблица должна находиться в первой нормальной форме
- Любое ее поле, не входящее в состав первичного ключа, функционально полно зависит от первичного ключа.

Методы приведения к 2NF:

- Создайте отдельные таблицы для наборов значений, относящихся к нескольким записям
- Свяжите эти таблицы с помощью внешнего ключа

Третья нормальная форма (3NF)

- Таблица находится во второй нормальной форме
- Любой ее не ключевой атрибут функционально зависит от первичного ключа

Проще говоря, второе правило требует выносить все не ключевые поля, содержимое которых может относиться к нескольким записям таблицы в отдельные таблицы (справочники).

Нормальная форма Бойса-Кодда

Эта форма почти то же, что и третья. С одним небольшим дополнительным условием.

- Таблица находится в третьей нормальной форме
- В таблице должен быть только один потенциальный первичный ключ

Другими словами, в таблице должен быть только один первичный ключ и не должно быть других потенциальных вариантов (например, наборе не ключевых полей этой таблицы).

Четвертая нормальная форма (4NF)

Ну, тут как и во всех предыдущих формах требования, включают в себя требования всех предыдущих форм + что-то еще. В это форме дополнительное правило должно исключать многозначные зависимости. Другими словами, все строки таблицы должны быть независимыми друг от друга. В том смысле, что наличие какой-то строки X, не должно означать, что строка Y тоже где-то есть в этой таблице.

Пятая нормальная форма (5NF)

Если таблица не может быть разбита на две таблицы без потерь зависимости, то говорят о наличии в ней зависимости по соединению (или зависимость соединения).

Таблица, в которой имеется зависимость соединения, не находится в пятой нормальной форме. (Важно то, что нельзя разбить на ДВЕ таблицы без потерь зависимости, если можно разбить на большее количество таблиц, то после разбиения получим 5 нормальную форму)

Шестая нормальная форма (6NF)

Переменная отношения находится в шестой нормальной форме тогда и только тогда, когда она удовлетворяет всем нетривиальным зависимостям соединения. Из определения следует, что переменная находится в 6НФ тогда и только тогда, когда она неприводима, то есть не может быть подвергнута дальнейшей декомпозиции без потерь. Каждая переменная отношения, которая находится в 6НФ, также находится и в 5НФ.

6. ER-модель: характеристики связей, сущностей, ограничения целостности, первичные и внешние ключи.

Между двумя сущностями,например А и Б возможны четыре вида связей. Первый тип - связь “один к одному”(1:1): в каждый момент времени каждому представителю (экземпляру) сущности А соответствует 1 или 0 представителей сущности Б (студент может не заработать или получить одну из повышенных стипендий).
Второй тип - связь “один ко многим”(1:M): одному представителю сущности А соответствует 0,1 или несколько представителей сущности Б.(квартира может пустовать, в ней может жить один или несколько человек)
Третий тип - “многие к одному”(M:1) и четвертый - “многие ко многим”(M:M). Также есть рекурсивная связь (свиное ухо) для описания иерархий.

Стержневая сущность (стержень) — это независимая сущность, которая не является ни характеристикой, ни ассоциацией.
Ассоциативная сущность (ассоциация) — это связь вида "многие-ко- многим" ("*-ко-многим" и т. д.) между двумя или более сущностями или экземплярами сущности. Ассоциации рассматриваются как полноправные сущности: они могут участвовать в других ассоциациях точно так же, как стержневые сущности; могут обладать свойствами, т. е. иметь не только набор ключевых атрибутов, необходимых для указания связей, но и любое число других атрибутов, характеризующих связь.
Характеристическая сущность (характеристика) — это связь вида "многие-к-одной" или "одна-к-одной" между двумя сущностями (частный случай ассоциации). Единственная цель характеристики в рамках рассматриваемой предметной области состоит в описании или уточнении некоторой другой сущности. Необходимость в них возникает в связи с тем, что сущности реального мира имеют иногда многозначные свойства. Муж может иметь несколько жен; книга — несколько характеристик переиздания (исправленное, дополненное, переработанное и пр.) и т. д.
Существование характеристики полностью зависит от характеризуемой сущности: женщины лишаются статуса жен, если умирает их муж.

Напомним, что ключ или возможный ключ — это минимальный набор атрибутов, по значениям которых можно однозначно найти требуемый экземпляр сущности. Минимальность означает, что исключение из набора любого атрибута не позволяет идентифицировать сущность по оставшимся. Каждая сущность должна обладать хотя бы одним возможным ключом. Если же возникает ситуация, когда из состава атрибутов сущности не удается создать возможного ключа (естественного ключа), то создают, так называемый, суррогатный ключ — автоматически сгенерированное значение, никак не связанное с информационным содержанием сущности. Один из возможных естественных ключей или суррогатный ключ принимается за первичный ключ. При выборе первичного ключа следует отдавать предпочтение несоставным ключам или ключам, составленным из минимального числа атрибутов. Нецелесообразно также использовать ключи с длинными текстовыми значениями (предпочтительнее использовать целочисленные атрибуты). Так, для идентификации студента можно использовать либо уникальный номер зачетной книжки, либо набор из фамилии, имени, отчества, номера группы и может быть дополнительных атрибутов, так как не исключено появление в группе двух студентов (а чаще студенток) с одинаковыми фамилиями, именами и отчествами. Не допускается, чтобы первичный ключ стержневой сущности (любой атрибут, участвующий в

первичном ключе) принимал неопределенное значение. Иначе возникнет противоречивая ситуация: появится не обладающий индивидуальностью и, следовательно, не существующий экземпляр стержневой сущности. По тем же причинам необходимо обеспечить уникальность первичного ключа.

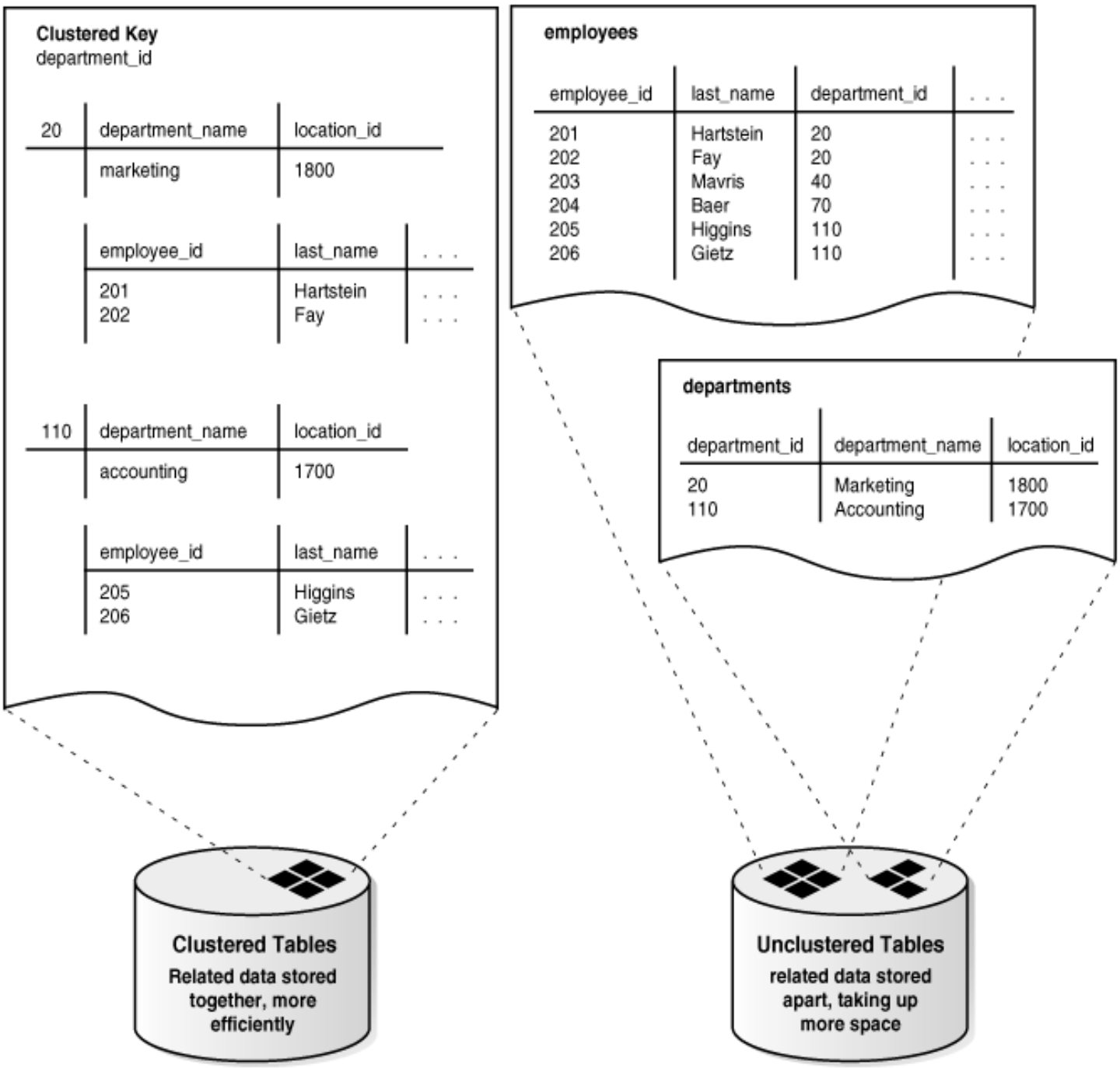
Теперь о внешних ключах: Если сущность В связывает сущности А и Б, то она должна включать внешние ключи, соответствующие первичным ключам сущностей А и Б. Например, ассоциативная сущность СОСТАВ (см. рис. 2.8) включает внешние ключи КОД_БЛЮДА и КОД_ПРОДУКТА, соответствующие первичным ключам связываемых стержневых сущностей БЛЮДА и ПРОДУКТЫ. Если сущность Б характеризует сущность А, то она должна включать внешний ключ, соответствующий первичному ключу сущности А. Например, характеристическая сущность РЕЦЕПТ (см. рис. 2.8 или 2.9) включает внешний ключ КОД_БЛЮДА, соответствующий первичному ключу характеризуемой сущности БЛЮДА.

Целостность — понимается как правильность данных в любой момент времени. Но эта цель может быть достигнута лишь в определенных пределах: СУБД не может контролировать правильность каждого отдельного значения, вводимого в базу данных (хотя каждое значение можно проверить на правдоподобность).
Выделяют три группы правил целостности.
Целостность по сущностям. Не допускается, чтобы какой-либо атрибут, участвующий в первичном ключе, принимал неопределенное значение.
Целостность по ссылкам. Значение внешнего ключа должно либо: быть равным значению первичного ключа ассоциируемой (характеризуемой) сущности; быть полностью неопределенным, т. е. каждое значение атрибута, участвующего во внешнем ключе, должно быть неопределенным.
Целостность, определяемая пользователем. Для любой конкретной базы данных существует ряд дополнительных специфических правил, которые относятся к ней одной и определяются разработчиком. Чаще всего контролируется: уникальность тех или иных атрибутов; диапазон значений (экзаменационная оценка от 2 до 5); принадлежность набору значений (пол "М" или "Ж").

7. Кластеры (в применении к таблицам в реляционных и объектно-реляционных БД).

Кластеры - опциональный метод хранения информации. Кластер это группа таблиц, которые делят одни и те же блоки данных, поскольку у них есть общие колонки и они часто используются вместе. К пример, таблицы **`employees`** и **`departments`** имеют общий **`department_id`** столбец. Когда мы создем кластер из таблиц **`employees`** и **`departments`**, Oracle физически сохраняет все строки для каждого отдела(department) из обеих таблиц в одних и тех же блоках данных.

Figure 5-10 Clustered Table Data



Профит:

- Время доступа и затраты памяти для объединений(join) этих таблиц уменьшается
- В кластере, каждый **cluster key value** сохраняется только один раз.В примере видно, как departmentID сохраняется всего один раз, независимо от того,сколько строк в таблицах имеют такое же значение. Это экономит дисковое пространство.

http://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_5001.htm

8. TCL, DML, DDL, DCL.

DDL

DDL - (Data Defenition Language) предложения для определения структуры базы данных или схемы. Примеры:
- CREATE - создает объекты базы данных (таблицы, представления и т.д.)
- ALTER - Изменяет структуру и объекты базы данных
- DROP - Удаляет объекты базы данных
- TRUNCATE - Удаляет все записи из таблицы
- COMMENT - Добавляет комментарии в словарь данных
- RENAME - Переименовывает объект (alter table <old_name> rename to <new_name>)

DML

DML - (Data Manipulation Language) предложения для управления данными. Примеры:
- SELECT - Возвращает данные из базы данных
- INSERT - Вставляет данные в таблицу
- UPDATE - Обновляет существующие данные в таблице
- DELETE - Удаляет все записи в таблице
- MERGE - UPSERT операция (insert или update)
- CALL - вызов подпрограммы PL/SQL или Java
- EXPLAIN PLAN - Предоставляет план запроса
- LOCK TABLE - Управление параллелизмом

DCL

DCL - Data Control Language. Примеры:
- GRANT - Дает пользователю привелегии доступа к базе данных и ее объектам
- REVOKE - Забирает у пользователя привелегии данные командой GRANT

TCL

TCL - (Transaction Control) предложения используемые для управления изменениями сделанными предложениями DML. Это позволяет объединять предложения DML в логические транзакции.
- COMMIT - Сохраняет изменения
- SAVEPOINT - Определяет точку транзакции до которой потом можно откатиться
- ROLLBACK - Восстанавливает базу данных на момент последней операции COMMIT. Откатывает транзакцию
- SET TRANSACTION - Изменяет опции транзакции, такие как: уровень изоляции и какой сегмент отката использовать

Предложения DML автоматически не сохраняются, т.е. вы можете использовать откат транзакции, но результьтаты DDL предложений сохраняются автоматически.

9. Предложения модификации данных (DML), иерархические SQL-запросы.(Кравцов)

Data Manipulation Language (DML) (язык управления (манипулирования) данными) — это семейство компьютерных языков, используемых в компьютерных программах или пользователями баз данных для получения, вставки, удаления или изменения данных в базах данных. На текущий момент

наиболее популярным языком DML является SQL, используемый для получения и манипулирования данными в РСУБД.
Функции языков DML определяются первым словом в предложении (часто называемом **запросом**), которое почти всегда является глаголом. В случае с SQL эти глаголы — «select» («выбрать»), «insert» («вставить»), «update» («обновить»), и «delete» («удалить»). Это превращает природу языка в ряд обязательных утверждений (команд) к базе данных.
Языки DML разделяются в основном на два типа:

- Процедурный DMLs — описывают действия над данными (**какие** данные нужны и **как** их получить).
- Декларативный DMLs — описывают сами данные (**какие** данные нужны, без указания способа их получения). SQL декларативный.

Иерархические запросы
Если таблица содержит иерархические данные, их можно выбрать при помощи конструкции:

Эта фраза употребляется в запросе после фразы WHERE.
Oracle формирует иерархическую выборку, выполняя следующие шаги:

1. Oracle выбирает корневую строку (строки) иерархии – ту строку, которая удовлетворяет условию в выражении START WITH.
2. Затем выбираются дочерние строки для каждой корневой строки. Каждая дочерняя строка должна удовлетворять условию в фразе CONNECT BY по отношению к одной из корневых строк.
3. Выбираются следующие поколения дочерних строк. Сначала выбираются потомки строк, выбранных на шаге 2, потом – их потомки и т.д.
4. Oracle всегда выбирает потомков, вычисляя условия CONNECT BY относительно текущей родительской строки.
5. Если запрос содержит фразу WHERE, исключаются все строки, которые не удовлетворяют условию в фразе WHERE. Oracle вычисляет эти условия для каждой строки, а не просто удаляет всех потомков строки, которая не удовлетворяет условию.

Оператор SELECT, выполняющий иерархический запрос, не может содержать соединение.
Выражение **START WITH** – задает строку/строки, лежащие в корне иерархии. Это выражение определяет условие, которому должны соответствовать корневые строки. Условие может содержать вложенные запросы. Если эта фраза не задана, то все строки таблицы являются корневыми.
CONNECT BY – задает отношение между родительскими и дочерними строками в иерархии. Отношение задается «P-условием», это может быть любое сравнение, но какая-то его часть должна содержать ключевое слово PRIOR, относящееся к родительской строке.
Чтобы найти дочерние строки, Oracle вычисляет **PRIOR**-выражение для родительской строки, а другое выражение – для каждой строки таблицы. Строки, для которых это выражение дает истину, являются дочерними. CONNECT BY может содержать и другие условия-фильтры. CONNECT BY не может содержать вложенных запросов.

Если CONNECT BY приводит к петле, Oracle возвращает ошибку

Если не понятно, нормальный пример тут http://habrahabr.ru/post/43955/

10. Блокирование и одновременный доступ. Механизм транзакций. (Полина)
Транзакция или логическая единица работы, - это в общем случае последовательность ряда таких операций, которые преобразуют некоторое непротиворечивое состояние базы данных в другое непротиворечивое состояние, но не гарантируют сохранения непротиворечивости во все промежуточные моменты времени.

Пример: необходимо перевести с банковского счёта номер 5 на счёт номер 7 сумму в 10 денежных единиц. Этого можно достичь, к примеру, приведённой последовательностью действий:

- Начать транзакцию

прочесть баланс на счету номер 5

уменьшить баланс на 10 денежных единиц

сохранить новый баланс счёта номер 5

прочесть баланс на счету номер 7

увеличить баланс на 10 денежных единиц

сохранить новый баланс счёта номер 7

- Окончить транзакцию

Эти действия представляют собой логическую единицу работы «перевод суммы между счетами», и таким образом, являются транзакцией. Если прервать данную транзакцию, к примеру, в середине, и не аннулировать все изменения, легко оставить владельца счёта номер 5 без 10 единиц, тогда как владелец счета номер 7 их не получит.

Корректное поддержание транзакций одновременно является основой обеспечения целостности баз данных, а также составляют базис изолированности пользователей во многопользовательских системах. Изолированность - такой режим функциональности, при котором каждому пользователю кажется, что база данных доступна только ему. Задача по обеспечению изолированности - параллелизм транзакций.

При выполнении транзакции СУБД должна придерживаться определенных правил обработки набора команд, в транзакцию входящих. Разработаны 4 правила, известные как требования ACID(atomicity, consistency,isolation,durability - неделимость,согласованность, изолированность, устойчивость), гарантирующие правильность и надежность работы системы:

- Атомарность гарантирует, что никакая транзакция не будет зафиксирована в системе частично. Будут либо выполнены все её подоперации, либо не выполнено ни одной. Поскольку на практике невозможно одновременно и атомарно выполнить всю последовательность операций внутри транзакции, вводится понятие «отката» (rollback): если транзакцию не удаётся полностью завершить, результаты всех её до сих пор произведённых действий будут отменены и система вернётся в исходное состояние.
- Транзакция является **согласованной**, потому что не нарушает бизнес-логику и отношения между элементами данных. Это свойство очень важно при разработке клиент-серверных систем, поскольку в хранилище данных поступает большое количество транзакций от разных

систем и объектов. Если хотя бы одна из них нарушит целостность данных, то все остальные могут выдать неверные результаты.

- Транзакция всегда **изолирована**, поскольку ее результаты самодостаточны. Они не зависят от предыдущих или последующих транзакций – это свойство называется сериализуемостью и означает, что транзакции в последовательности независимы.
- Транзакция долговременна. После своего завершения она сохраняется в системе, которую ничто не может вернуть в исходное (до начала транзакции) состояние, т.е. происходит фиксация транзакции, означающая, что ее действие постоянно даже при сбое системы. При этом подразумевается некая форма хранения информации в постоянной памяти как часть транзакции .

Указанные выше правила выполняет сервер. Программист лишь выбирает нужный уровень изоляции, заботится о соблюдении логической целостности данных и бизнес-правил.

Блокировки

Повышение эффективности работы при использовании небольших транзакций связано с тем, что при выполнении транзакции сервер накладывает на данные блокировки.

Блокировкой называется временное ограничение на выполнение некоторых операций обработки данных. Блокировка может быть наложена как на отдельную строку таблицы, так и на всю базу данных. Управлением блокировками на сервере занимается менеджер блокировок, контролирующий их применение и разрешение конфликтов. Транзакции накладывают блокировки на данные, чтобы обеспечить выполнение требований ACID. Без использования блокировок несколько транзакций могли бы изменять одни и те же данные.

Блокировка представляет собой метод управления параллельными процессами, при котором объект БД не может быть модифицирован без ведома транзакции, т.е. происходит блокирование доступа к объекту со стороны других транзакций.

Различают два вида блокировки:

- блокировка записи – транзакция блокирует строки в таблицах таким образом, что запрос другой транзакции к этим строкам будет отменен ;
- блокировка чтения – транзакция блокирует строки так, что запрос со стороны другой транзакции на блокировку записи этих строк будет отвергнут, а на блокировку чтения – принят.

В СУБД используют протокол доступа к данным, позволяющий избежать проблемы параллелизма. Его суть заключается в следующем:

- транзакция, результатом действия которой на строку данных в таблице является ее извлечение, обязана наложить блокировку чтения на эту строку;
- транзакция, предназначенная для модификации строки данных, накладывает на нее блокировку записи;
- если запрашиваемая блокировка на строку отвергается из-за уже имеющейся блокировки, то транзакция переводится в режим ожидания до тех пор, пока блокировка не будет снята;
- блокировка записи сохраняется вплоть до конца выполнения транзакции.

Решение **проблемы параллельной обработки** БД заключается в том, что строки таблиц блокируются, а последующие транзакции, модифицирующие эти строки, отвергаются и переводятся в режим ожидания. В связи со свойством сохранения целостности БД транзакции являются подходящими единицами изолированности пользователей.

Если в системе управления базами данных не реализованы механизмы блокирования, то при одновременном чтении и изменении одних и тех же данных несколькими пользователями могут возникнуть следующие проблемы одновременного доступа:

- проблема последнего изменения возникает, когда несколько пользователей изменяют одну и ту же строку, основываясь на ее начальном значении; тогда часть данных будет потеряна, т.к. каждая последующая транзакция перезапишет изменения, сделанные предыдущей. Выход из этой ситуации заключается в последовательном внесении изменений;
- проблема **"грязного" чтения** возможна в том случае, если пользователь выполняет сложные операции обработки данных, требующие множественного изменения данных перед тем, как они обретут логически верное состояние. Если во время изменения данных другой пользователь будет считывать их, то может оказаться, что он получит логически неверную информацию. Для исключения подобных проблем необходимо производить считывание данных после окончания всех изменений;
- проблема **неповторяемого чтения** является следствием неоднократного считывания транзакцией одних и тех же данных. Во время выполнения первой транзакции другая может внести в данные изменения, поэтому при повторном чтении первая транзакция получит уже иной набор данных, что приводит к нарушению их целостности или логической несогласованности;
- проблема чтения **фантомов** появляется после того, как одна транзакция выбирает данные из таблицы, а другая вставляет или удаляет строки до завершения первой. Выбранные из таблицы значения будут некорректны.

Для решения перечисленных проблем в специально разработанном стандарте определены четыре уровня блокирования. Уровень изоляции транзакции определяет, могут ли другие (конкурирующие) транзакции вносить изменения в данные, измененные текущей транзакцией, а также может ли текущая транзакция видеть изменения, произведенные конкурирующими транзакциями, и наоборот. Каждый последующий уровень поддерживает требования предыдущего и налагает дополнительные ограничения:

- уровень 0 – запрещение "загрязнения" данных. Этот уровень требует, чтобы изменять данные могла только одна транзакция ; если другой транзакции необходимо изменить те же данные, она должна ожидать завершения первой транзакции ;
- уровень 1 – запрещение "грязного" чтения. Если транзакция начала изменение данных, то никакая другая транзакция не сможет прочитать их до завершения первой;
- уровень 2 – запрещение неповторяемого чтения. Если транзакция считывает данные, то никакая другая транзакция не сможет их изменить. Таким образом, при повторном чтении они будут находиться в первоначальном состоянии;
- уровень 3 – запрещение фантомов. Если транзакция обращается к данным, то никакая другая транзакция не сможет добавить новые или удалить имеющие строки, которые могут быть считаны при выполнении транзакции. Реализация этого уровня блокирования выполняется путем использования блокировок диапазона ключей. Подобная блокировка накладывается не на конкретные строки таблицы, а на строки, удовлетворяющие определенному логическому 4условию.

11. Динамический SQL (1-3 методы).(Топалов, Переведу чуть позже)

Метод	Тип предложения	Требуемые вызовы пакета DBMS_SQL
1	Незапросные, нет базовых переменных, выполняется однократно	открытие курсора (open cursor), разборка (parse), выполнение (execute), закрытие курсора (close cursor)
2	Незапросные, известное число базовых переменных, выполняются один или несколько раз	открытие курсора, разборка, связывание переменных (bind variables), выполнение, закрытие курсора
3	Запросные, известное число операторов SELECT и базовых переменных	открытие курсора, разборка, связывание переменных, определение колонок (define columns), выполнение, выборка строк (fetch rows), получение значений колонок (get column values), обновление (refretch), ... закрытие курсора

Method 1

Этот метод позволяет вашей программе принимать или строить динмические SQL-предложения, затем мгновенно исполнить их с помощью команды EXECUTE IMMEDIATE. SQL предложение не должно быть запросом (SELECT-предложение) и не должно содержать мест для переменных функции. `'DELETE FROM EMP WHERE DEPTNO = 20'`
`'GRANT SELECT ON EMP TO scott'`
SQL-предложение парсится каждый раз, когда выполняется.

Method 2

Этот метод позволяет вашей программе принимать или строить динамические SQL-предложения, затем обрабатывать их с помощью PREPARE и EXECUTE команд. Предложение не должно быть запросом. Количество мест для переменных из программы должно быть известно заранее.
`'INSERT INTO EMP (ENAME, JOB) VALUES (:emp_name, :job_title)'`
`'DELETE FROM EMP WHERE EMPNO = :emp_number'`
SQL-предложение парсится однажды, но может быть запущено несколько раз с разными значениями переменных. SQL предложения, такие как CREATE и GRANT исполняются после того, как они PREPARE-d(готовы).

Method 3

Этот метод позволяет вашей программе принимать или строить динамические запросы, затем обрабатывать их, используя команду PREPARE и DECLARE, OPEN, FETCH, и CLOSE команды курсора. Количество переменных, типы переменных должныть быть извесстны до компиляции:
`'SELECT DEPTNO, MIN(SAL), MAX(SAL) FROM EMP GROUP BY DEPTNO'`
`'SELECT ENAME, EMPNO FROM EMP WHERE DEPTNO = :dept_number'`

12. Представления (предназначение, процедуры создания и обновления) и курсоры (SQL). (Полина)

Представление - это пустая именованная таблица, определяемая перечнем тех столбцов таблиц и признаками тех их строк, которые хотелось бы в ней увидеть. Представление является как бы "окном" в одну или несколько базовых таблиц. Оно создается с помощью предложения CREATE VIEW.
Представление не поддерживаются его собственными физическими хранимыми данными. Вместо этого в каталоге таблиц хранится определение, оговаривающее, из каких столбцов и строк других таблиц оно должно быть сформировано при реализации SQL-предложения на получение данных из представления или на модификацию таких данных.

```
CREATE VIEW имя_представления
        [(столбец[,столбец] ...)]
        AS подзапрос
        [WITH CHECK OPTION];
```

где подзапрос, следующий за AS и являющийся определением данного представления, не исполняется, а просто сохраняется в каталоге;
необязательная фраза "WITH CHECK OPTION" (с проверкой) указывает, что для операций INSERT и UPDATE над этим представлением должна осуществляться проверка, обеспечивающая удовлетворение WHERE фразы подзапроса;
список имен столбцов должен быть обязательно определен лишь в тех случаях, когда:
а) хотя бы один из столбцов подзапроса не имеет имени (создается с помощью выражения, SQL-функции или константы);
б) два или более столбцов подзапроса имеют одно и то же имя;
если же список отсутствует, то представление наследует имена столбцов из подзапроса.
уничтожение: DROP VIEW представление;

Операции выборки такие же,как и из обычных таблиц(select лала from представление where лала) СУБД преобразует его при выполнении в эквивалентную операцию над лежащими в основе базовыми таблицами (перед выполнением проводит слияние выданного пользователем SELECT с предложениями SELECT из описаний представлений, хранящихся в каталоге).

Обновление
обновляемыми являются представления, полученные из единственной базовой таблицы простым исключением некоторых ее строк и (или) столбцов, обычно называемые "представление-подмножество строк и столбцов", с ними можно сделать insert,update,delete, что приведет к изменению таблиц,лежащих в основе. Если представление образовано более сложным образом, система может не понять, какой столбец,где обновлять и выдать ошибку, такие представления обновляемыми не являются.

К теоретически обновляемым представлениям относятся представления-подмножества строк и столбцов. Однако существуют некоторые представления, которые не являются представлениями-подмножествами строк и столбцов, но также теоретически обновляемы. Хотя известно, что такие есть и можно привести их примеры, но невозможно дать их формального определения. В общем,все очень запутано и зависит от реализации.

Предназначение

Одна из основных задач, которую позволяют решать представления, - обеспечение независимости пользовательских программ от изменения логической структуры базы данных при ее расширении и

(или) изменении размещения столбцов, возникающего, например, при расщеплении таблиц. В последнем случае можно создать представление-соединение с именем и структурой расщепленной таблицы, позволяющее сохранить программы, существовавшие до изменения структуры базы данных.
Кроме того, представления дают возможность различным пользователям по-разному видеть одни и те же данные, возможно, даже в одно и то же время. Это особенно ценно при работе различных категорий пользователей с единой интегрированной базой данных. Пользователям предоставляют только интересующие их данные в наиболее удобной для них форме (окно в таблицу или в любое соединение любых таблиц).
Наконец, от определенных пользователей могут быть скрыты некоторые данные, невидимые через предложенное им представление. Таким образом, принуждение пользователя осуществлять доступ к базе данных через представления является простым, но эффективным механизмом для управления санкционированием доступа.

Курсоры

Курсор - это пустая именованная таблица, определяемая перечнем тех столбцов базовых таблиц и признаками тех их строк, которые хотелось бы в ней увидеть.
Отличие от представления:
Для пользователя представления почти не отличаются от базовых таблиц. Они могут использоваться как в интерактивном режиме, так и в прикладных программах. Курсоры же созданы для процедурной работы с таблицей в прикладных программах. Например, после объявления курсора:

```
DECLARE     Блюд_состав CURSOR FOR
SELECT      Блюдо,Продукт,Вес
FROM Состав,Блюда,Продукты
WHERE       Состав.БЛ = Блюда.БЛ
AND   Состав.ПР = Продукты.ПР
AND   Блюдо = 'Суп харчо';
```

и его активизации (OPEN Блюд_состав) будет создана временная таблица с составом блюда "Суп харчо" и специальным указателем, определяющим в качестве текущей первую строку этой таблицы. С помощью предложения FETCH (выбрать), которое обычно исполняется в программном цикле, можно присвоить определенным переменным значения указанных столбцов этой строки. Одновременно курсор будет передвинут к следующей строке таблицы. После обработки в программе полученных значений переменных выполняется следующее предложение FETCH и т.д. до окончания перебора всех продуктов Харчо.

13. Основные конструкции PL/SQL: анонимные блоки, процедуры, функции, типы (в т.ч. поля и методы). (Гуляев Павел)
Поля и методы типов (коды посмотрите):
http://docs.oracle.com/cd/B10501_01/appdev.920/a96595/dci05pls.htm

Более полно - http://docs.oracle.com/cd/B12037_01/appdev.101/b10807/10_objs.htm#i12435

Про блоки, процедуры, функции думаю все итак знают, лучше всего прочитать 18 главу Громова или здесь http://ivan-shamaev.ru/pl_sql_functions_procedures_variables_cursors_cycles/

14. Скалярные и пользовательские типы данных в PL/SQL.(Гуляев Павел)

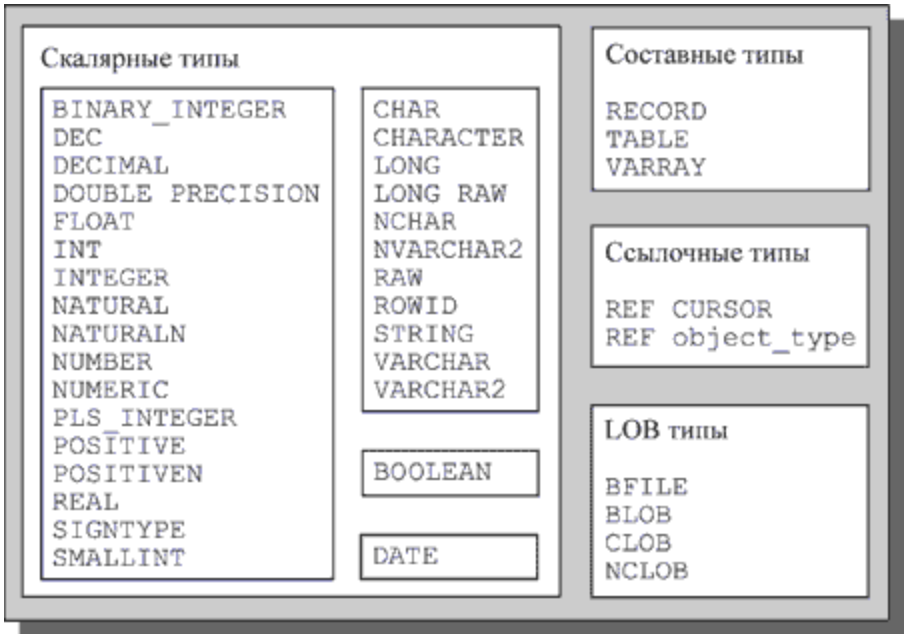
Типы данных

Язык PL/SQL поддерживает следующие категории типов:

- встроенные типы данных, включая коллекции и записи;
- объектные типы данных.

Встроенные типы данных

На рисунке приведен список встроенных типов PL/SQL.



Скалярные типы описывают простые значения, не имеющие внутренних составляющих.
Составные типы описывают структуры, в которых имеются внутренние компоненты.
Ссылочные типы содержат значения. LOB типы содержат значения, называемые локаторами, которые определяют расположение больших объектов хранимых данных (например, графические файлы).
В следующей таблице приведено описание некоторых типов данных языка PL/SQL.

Синтаксис	Диапазон значений
Числовые типы	
BINARY_INTEGER PLS_INTEGER (целое со знаком)	-2147483647 .. 2147483647 . Тип PLS_INTEGER требует меньше памяти и обрабатывается быстрее, чем другие *числовые типы*
NUMBER[(*precision*,scale)] (с плавающей точкой)	1.0E-130 .. 9.99E125
NUMERIC (с фиксированной точкой)	точность до 38 десятичных знаков
FLOAT (с плавающей точкой)	точность до 38 десятичных знаков
REAL (с плавающей точкой)	точность до 18 десятичных знаков
Символьные типы	
CHAR[(maximum_length)] (для строк постоянной длины) LONG (для строк переменной длины)	до 32767 байт Для столбца базы данных максимальный размер типа CHAR составляет 2000 байтов, а типа LONG – до 2 Гб
RAW(maximum_length) LONG *RAW* (для двоичных данных или строк байтов)	до 32767 байт Для столбца базы данных типа *RAW* максимальный размер – 2000 байт.
VARCHAR2 (maximum_length) (для строк символов переменной длины)	до 32767 байт

Все встроенные типы данных являются базовыми типами.
Любой базовый тип PL/SQL определяется как набор значений и набор операций, выполнимых над этими значениями.
Язык PL/SQL позволяет определять новые подтипы как подмножество значений некоторого базового типа с тем же набором операций. Подтип не вводит никаких дополнительных операций над данными и не определяет никакого нового типа.
Определение подтипа может иметь следующее формальное описание:
SUBTYPE subtype_name IS base_type;
В пакете STANDARD базы данных Oracle, автоматически подключаемом для любого блока, определено несколько подтипов.
Пользователь может определить свой тип как некоторый подтип в секции объявлений блока, подпрограммы или пакете PL/SQL.
Например:

```
DECLARE
  SUBTYPE MyDate IS DATE;
    - Основан на типе DATE
  TYPE MyRec IS RECORD (time1 INTEGER,
                        time2 INTEGER);
  SUBTYPE MyInterval IS MyRec;
    - Основан на типе RECORD
  SUBTYPE ID_N IS tbl1.f1%TYPE;
    - Основан на типе столбца
```

Типы, используемые как базовые, не могут содержать ограничений длины. Для создания пользовательского типа с ограничением длины предварительно объявляется переменная такого типа и уже на ее основе определяется пользовательский тип.
Например:

```
DECLARE
  var1 VARCHAR2(6);
    - Объявление переменной
    - с ограничением длины
  SUBTYPE string_6 IS var1%TYPE;
    - Объявление
```

LOB-типы

LOB-типы используются для хранения больших объектов (Large Object). Стандарт SQL-99 ввел поддержку LOB-типов для расширенного уровня соответствия. Однако в Oracle реализован более полный набор LOB-типов.

В Oracle8 позволяется хранить данные LOB-типа до 4 Гбайт.

Типы LOB от типа LONG отличаются, главным образом, тем, что при выборе значения любого LOB-типа посредством оператора SELECT возвращается указатель, а не само значение; кроме того, типы LOB могут быть и внешними.

Oracle поддерживает следующие четыре типа для больших объектов:

- BFILE - для внешнего двоичного файла;
- BLOB - для внутреннего двоичного объекта;
- CLOB - для внутреннего символьного объекта;
- NCLOB - для внутреннего символьного объекта, учитывающего национальный набор символов.

Любой объект LOB состоит из двух частей: данных и указателя на эти данные, называемого локатором.

Типы BLOB, CLOB или NCLOB могут использоваться как для столбца базы данных, так и для переменной PL/SQL.

Для загрузки объекта LOB предусмотрен пакет PL/SQL DBMS_LOB.

Пакет DBMS_LOB для работы с LOB-типами содержит процедуры и функции, некоторые из которых приведены в следующей таблице.

Синтаксис	Описание
`APPEND (d1,d2)`	Добавляет `d2` к `d1`
`COMPARE(d1,d2,n,pos1,pos2)`	Сравнивает `n` байт значений `d1` и `d2`
`COPY (d,s,n,dp,sp)`	Копирует `n` байт из `d` в `s`.
`FILEOPEN (bdata,m)`	Открывает объект типа `BFILE` в режиме, указанном параметром `m`
`LOADFROMFILE (bdata1,data2,n,pos1,pos2)`	Копирует `n` байт объекта типа `BFILE bdata1` в любой объект LOB `data2`
`GETLENGTH (data)`	Возвращает длину указанного объекта LOB
`READ (data,n,pos,buf)`	Читает из объекта `data n` байт
`WRITE (data,n,pos,buf)`	Копирует из буфера `buf n` байт
`EMPTY_CLOB (), EMPTY_BLOB ()`	Создают "пустой" объект указанного типа

Например:

```
DECLARE
  pf1 CLOB;
  pf2 BLOB;
  buf varchar2;
BEGIN
```

```
CREATE TABLE tbl1 ( f1 CLOB, f2 BLOB);
INSERT INTO tbl1 VALUES
  (empty_clob(),empty_blob() );
  SELECT f1 INTO pf1 FROM tbl1 FOR UPDATE;
  buf := 'Текст, который будет вставлен
  в объект LOB';
  DBMS_LOB.write (pf1, length(buf), 0, buf);
END;
```

(http://www.intuit.ru/studies/courses/4/4/lecture/114?page=2)

15. ООП в СУБД, поля, методы, основные свойства, наследование типов (Крихели Артём)
http://www.oraclebi.ru/files/presentations/imelnikov/oop_plsql.pdf

16. Функции, возвращающие объекты и таблицы объектов, конвейерные функции. (Полина)
я взяла то,что подходит под вопрос,но вообще,в этой статье больше информации,почитайте, если интересно http://www.fors.ru/upload/magazine/07/http_text/w_dev_pipelined_tf.html

Табличные функции используются для возврата PL/SQL-коллекций, которые имитируют таблицы. Они могут быть запрошены как обычные таблицы с помощью функцию TABLE во фразе FROM. Обычные табличные функции требуют, чтобы коллекции перед возвращением были полностью наполнены (населены). Так как коллекции хранятся в памяти, это может стать проблемой, поскольку на большие коллекции впустую тратится много памяти и времени в ожидании возвращения первой строки. Эти узкие возможности делают обычные табличные функции непригодными в случаях масштабных ETL-операций (ETL — Extraction Transformation Load — Извлечение-Преобразование-Загрузка). Обычные табличные функции требуют создания именованной строки и табличных типов как объектов базы данных.

```
-- Создание типов для функции.
DROP TYPE t_tf_tab;
DROP TYPE t_tf_row;

CREATE TYPE t_tf_row AS OBJECT (
  id          NUMBER,
  description  VARCHAR2(50)
);
/

CREATE TYPE t_tf_tab IS TABLE OF t_tf_row;
/

-- Build the table function itself.
CREATE OR REPLACE FUNCTION get_tab_tf (p_rows IN NUMBER) RETURN t_tf_tab AS
  l_tab  t_tf_tab := t_tf_tab();
BEGIN
  FOR i IN 1 .. p_rows LOOP
    l_tab.extend;
    l_tab(l_tab.last) := t_tf_row(i, 'Description for ' || i);
  END LOOP;
```

```
  RETURN l_tab;
END;
```

Конвейерные табличные функции)
(Pipelined Table Functions
Конвейерная обработка отменяет надобность в создании огромных наборов, передавая строки по каналу из функции по мере их создания, сохраняя память и позволяя запустить последующую обработку еще до окончания генерации всех строк.
Конвейерные табличные функции включают фразу PIPELINED и используют вызов PIPE ROW, чтобы вытолкнуть строки из функции, как только они создадутся, вместо построения табличной коллекции. Заметим, что вызов RETURN пустой, поскольку нет никакой коллекции, возвращаемой из функции.

```
-- Построение конвейерной табличной функции.
CREATE OR REPLACE FUNCTION get_tab_ptf (p_rows IN NUMBER) RETURN t_tf_tab PIPELINED AS
BEGIN
  FOR i IN 1 .. p_rows LOOP
    PIPE ROW(t_tf_row(i, 'Description for ' || i));
  END LOOP;

  RETURN;
END;
/
```

Когда ETL-операции проводятся на большом хранилище данных, наблюдается существенное повышение производительности, поскольку загрузка данных из внешних таблиц производится табличными функциями непосредственно в таблицы хранилища, избегая промежуточного размещения данных.

Исключение NO_DATA_NEEDED
(NO_DATA_NEEDED Exception)
Конвейерная табличная функция может создать больше данных, чем необходимо запросившему её процессу. Когда такое происходит, конвейерная табличная функция останавливает выполнение, порождая исключение NO_DATA_NEEDED. Оно не должно явно обрабатываться, если только в процедуру не включен обработчик исключений OTHERS.
Приведенная ниже функция возвращает 10 строк, но запрос потребовал от нее только первые 5 строк. В этом случае функция прекращает выполнение, вызывая исключение NO_DATA_NEEDED.

```
-- Построение конвейерной табличной функции.
CREATE OR REPLACE FUNCTION get_tab_ptf (p_rows IN NUMBER) RETURN t_tf_tab PIPELINED AS
BEGIN
  FOR i IN 1 .. p_rows LOOP
    DBMS_OUTPUT.put_line('Row: ' || i);
    PIPE ROW(t_tf_row(i, 'Description for ' || i));
  END LOOP;

  RETURN;
END;
/
```

Если имеется обработчик исключений OTHERS, то он захватит исключение NO_DATA_NEEDED и выполнит некоторый код обработки ошибок, что не нужно.

```
-- Построение конвейерной табличной функции.
CREATE OR REPLACE FUNCTION get_tab_ptf (p_rows IN NUMBER) RETURN t_tf_tab PIPELINED AS
BEGIN
  FOR i IN 1 .. p_rows LOOP
    DBMS_OUTPUT.put_line('Row: ' || i);
    PIPE ROW(t_tf_row(i, 'Description for ' || i));
  END LOOP;

  RETURN;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.put_line('OTHERS Handler');
    RAISE;
END;
/
```

Если вы планируете использовать обработчик исключений OTHERS, то для исключения NO_DATA_NEEDED необходимо задействовать специальное прерывание.

```
-- Построение конвейерной табличной функции.
CREATE OR REPLACE FUNCTION get_tab_ptf (p_rows IN NUMBER) RETURN t_tf_tab PIPELINED AS
BEGIN
  FOR i IN 1 .. p_rows LOOP
    DBMS_OUTPUT.put_line('Row: ' || i);
    PIPE ROW(t_tf_row(i, 'Description for ' || i));
  END LOOP;

  RETURN;
EXCEPTION
  WHEN NO_DATA_NEEDED THEN
    RAISE;
  WHEN OTHERS THEN
    DBMS_OUTPUT.put_line('OTHERS Handler');
    RAISE;
END;
/
```

17. Коллекции и работа с ними.

Коллекцией называется упорядоченная группа элементов одного типа. Язык PL/SQL поддерживает три вида коллекций: вложенные, индексированные таблицы и varray-массивы

Доступ к любому элементу вложенной таблицы или varray-массива осуществляется по его индексу, который указывается в скобках после имени переменной типа коллекции.

Для создания коллекции следует определить тип коллекции – TABLE или VARRAY – и объявить переменную этого типа. Определение типа выполняется в секции объявлений блока PL/SQL, подпрограммы или пакета.

TYPE type_name IS TABLE OF element_type [NOT NULL];

Параметр type_name указывает имя определяемого типа, а element_type - это любой допустимый тип данных PL/SQL, исключая некоторые типы, в том числе VARRAY, TABLE, BOOLEAN, LONG, REF CURSOR и т.п.

Вложенную таблицу можно рассматривать как одномерный массив, в котором индексами служат значения целочисленного типа в диапазоне от 1 до 2147483647. Вложенная таблица может иметь пустые элементы, которые появляются после их удаления встроенной процедурой DELETE. Вложенная таблица может динамически увеличиваться.

Индексированные таблицы позволяют работать со столбцами как с единой переменной - массивом.Определение индексированной таблицы (index-by tables):

TYPE type_name IS TABLE OF element_type [NOT NULL] INDEX BY BINARY_INTEGER;

Индексированная таблица - это вариант вложенной таблицы, в которой элементы могут иметь произвольные целочисленные значения индексов. Такой тип данных очень удобен, если в качестве индекса использовать значение первичного ключа.

Определение типа Varray-массива может иметь следующее формальное описание:
TYPE type_name IS {VARRAY | VARYING ARRAY} (size_limit) OF element_type [NOT NULL];

Параметр type_name указывает имя определяемого типа, size_limit - максимальное количество элементов, а element_type - это любой допустимый тип данных PL/SQL, исключая некоторые типы, такие как VARRAY, TABLE, BOOLEAN, LONG, REF CURSOR и т.п.

Максимальное количество элементов в Varray-массиве указывается при определении типа и не может изменяться динамически. Доступ к каждому элементу Varray-массива осуществляется по индексу. Varray-массивы можно передавать в качестве параметров. Varray-массивы не могут иметь пустот, так как для них нет операции удаления произвольного элемента массива.

```
DECLARE
  TYPE d1 IS VARRAY(365) OF DATE;
  TYPE rec1 IS RECORD (v1 VARCHAR2(10), v2 VARCHAR2(10));
  - Массив записей
  TYPE arr_rec IS VARRAY(250) OF rec1;
  - Вложенная таблица
  TYPE F1T1 IS TABLE OF tbl1.f1%TYPE;
  CURSOR c1 IS SELECT * FROM tbl1;
  - Массив записей, основанный на курсоре
  TYPE t1 IS VARRAY(50) OF c1%ROWTYPE;
  - Индексированная таблица
  TYPE t2 IS TABLE OF tbl1%ROWTYPE INDEX BY BINARY_INTEGER;
  - Объявление переменной
  rec_t2 t2;
BEGIN
   /* Использование переменной типа "индексированная таблица" */
   SELECT * INTO rec_t2(120) WHERE f1 = 120;
END;
```

Для инициализации коллекции используется конструктор - автоматически создаваемая функция, одноименная с типом коллекции. Конструктор без параметров – пустая коллекция.

В PL/SQL реализован ряд встроенных методов для работы с коллекциями. Эти методы вызываются как collection_name.method_name[(parameters)]

Методы: EXISTS(n), COUNT, LIMIT (для вложенных таблиц возвращает NULL), DELETE (m,n) (указываем только n – удаляется элемент n; m и n – с m до n; без аргументов – удалить всё), FIRST, LAST (наименьший и наибольший индекс элементов), PRIOR(n) (индекс элкмента, предшествующего n), NEXT(n) (индекс следующего), EXTEND (n,i) (Если параметров не задано, то в коллекцию добавляется один null-элемент, а если

указан только параметр n, то добавляются n null-элементов. Если задано оба параметра, то добавляются n элементов, являющихся копиями i-го элемента коллекции), TRIM(n) (Если параметры не указаны, то удаляется один последний элемент, а при задании параметра удаляются n последних элементов коллекции. Если значение параметра превышает реальное количество элементов, возвращаемое функцией COUNT, то инициируется исключение. Если элемент был ранее удален функцией DELETE, то он все равно будет входить в число удаляемых функцией TRIM элементов).

18. Управление правами пользователей: объектные, системные привилегии, роли.

DDL-операторы GRAND и REVOKE. Эти операторы нельзя использовать непосредственно в PL/SQL. Существуют привилегии двух различных видов: объектные и системные. Объектная привилегия (object privilege) разрешает выполнение определенной операции над конкретным объектом (например, над таблицей). Системная привилегия (system privilege) разрешает выполнение операций над целым классом объектов. Существует множество системных привилегий, соответствующих практически всем возможным операциям DDL. Например, системная привилегия CREATE TABLE, позволяет ее обладателю создавать таблицы. А, вот системная привилегия CREATE ANY TABLE дает возможность создавать таблицы в других схемах.

```
GRANT system-privilege/role to user/role/public WITH ADMIN OPTION
```

system_privilege - предоставляемое системное полномочие.
role - предоставляемая роль.
TO - определяет пользователей или роли, которым предоставляются системные полномочия.
PUBLIC - указывает что, системные полномочия определяемые администратором предоставляются всем пользователям.
WITH ADMIN OPTION - позволяет получившему системные полномочия или роль предоставлять их в дальнейшем другими пользователям или ролям. Такое решение в частности включает и возможность изменение или удаления роли.
Основных операций в языке DDL три - это CREATE, ALTER, DROP.
Группа ALTER:
`ALTER DATABASE` - Позволяет изменять саму БД.
`ALTER USER` - Позволяет изменять пользователя и его параметры (пароль, профиль, роль)
`ALTER PROFILE` - Позволяет изменять профили.
`ALTER TABLESPACE` - Позволяет изменять табличные пространства.
Для любого объекта - ANY:
`ALTER ANY PROCEDURE` - Разрешает изменение любой хранимой функции процедуры или пакета в любой схеме.
`ALTER ANY ROLE` - Разрешает изменение любой роли БД.
`ALTER ANY SEQUENCE` - Разрешает изменение любой последовательности в БД.
`ALTER ANY TABLE` - Разрешает изменение любой таблицы или вида в схеме БД.
`ALTER ANY TRIGGER` - Позволяет разрешать, запрещать компилировать любой триггер в любой схеме БД.
`ALTER ANY INDEX` - Разрешает изменение любого индекса в любой схеме.
Группа CREATE:
Позволяет создавать в [любой] схеме соответствующий объект:

```
CREATE [ANY] PROCEDURE/ SEQUENCE/ TABLE/ TRIGGER/ VIEW/ INDEX;
CREATE SESSION/ROLE;
```

Удаление объектов в [любой] схеме, а так же очистка таблиц:

```
DELETE [ANY] TABLE/ PROCEDURE/ SEQUENCE/ TABLE/ TRIGGER/ VIEW/ INDEX;
```

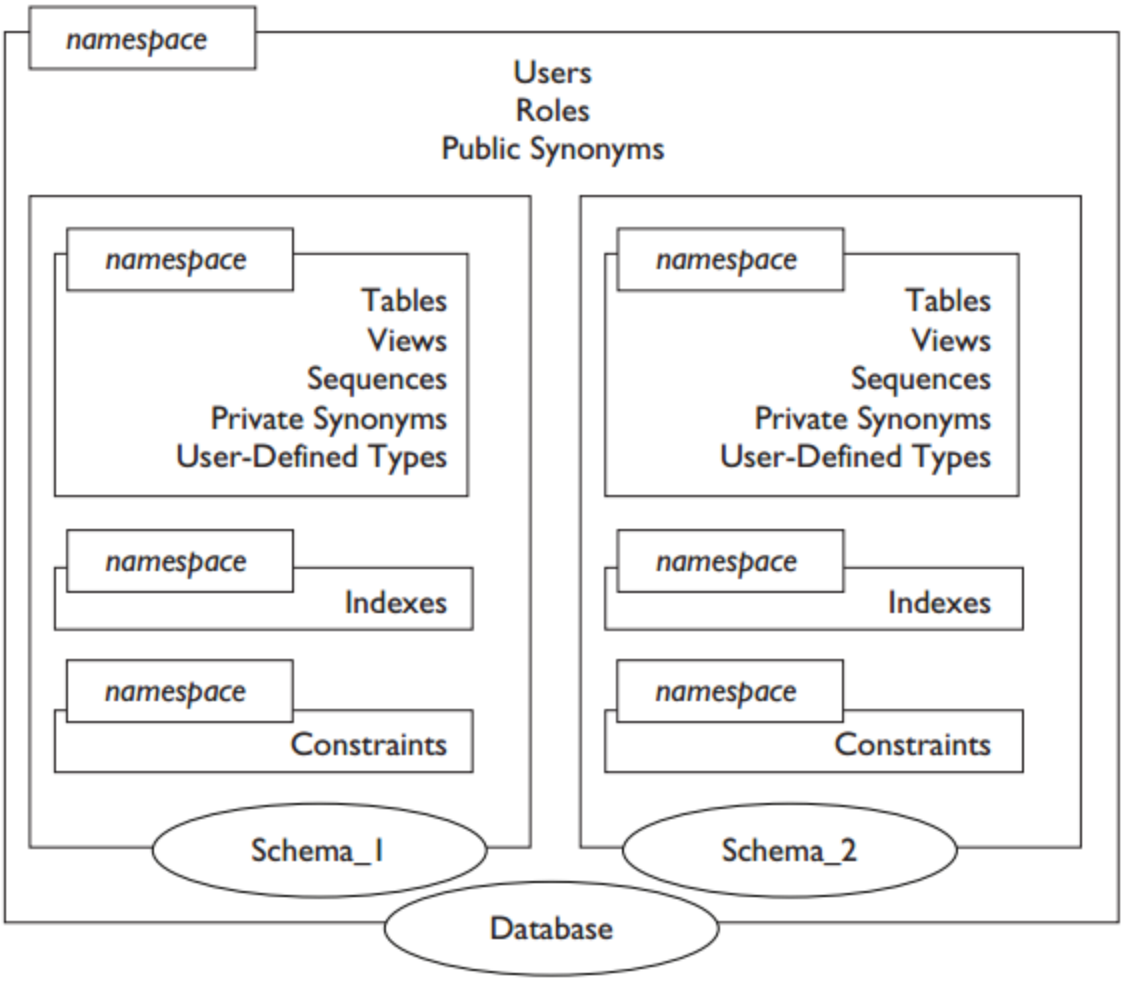
И еще полезные системные привилегии:

```
GRANT EXECUTE [ANY] PROCEDURE
GRANT ANY PRIVILEGE/ ROLE; INSERT ANY TABLE; LOCK ANY TABLE;
```

SELECT ANY TABLE; SELECT ANY SEQUENCE;
Объектные привилегии:
GRANT object_privilege/all/privilege column ON schema.object to USER/Role/Public WITH ADMIN OPTION
object_privilege: ALTER,SELECT,UPDATE,DELETE,INSERT,EXECUTE,INDEX,REFERENCES
Роль - это комплексный набор полномочий. Для разграничения доступа роль можно наделить привелегиями и назначить пользователям. CREATE ROLE role1 NOT IDENTIFIED;
GRANT role1 to usr0;

19. Пространства имен.

Пространство имен – некоторое множество, под которым подразумевается модель, абстрактное хранилище или окружение, созданное для логической группировки уникальных идентификаторов (т.е. имен).



20. Синонимы.

Синоним (Synonym) – это альтернативное имя (псевдоним) для объекта схемы. Если для какого-либо объекта базы данных Oracle существует синоним, то к объекту из SQL запроса можно обращаться либо по его настоящему имени, либо по синониму. Так же они обеспечивают некоторый уровень безопасности, поскольку скрывают имя объекта и его владельца, а также делают прозрачным местоположение удаленных объектов распределенных баз данных.

Синонимы позволяют переименовывать и перемещать базовые объекты. При том переопределяется только синоним, а приложение не требует никаких модификаций.

Различают два типа синонимов:

Частный (PRIVATE)- синонимы содержаться в схеме конкретного пользователя и доступны только самому пользователю, и тем, кому он предоставил соответствующие права доступа.

Общий (PUBLIC)- этими синонимами владеет специальная группа пользователей – PUBLIC, в результате чего эти синонимы доступны всем пользователям базы данных.

Чтобы создать частный синоним необходима привилегия CREATE SYNONYM, для того чтобы иметь права на создание синонима в схеме другого пользователя, необходима привилегия CREATE ANY SYNONYM. Для создания общих синонимов, необходима привилегия CREATE PUBLIC SYNONYM.

Для создания синонимов используется конструкция CREATE SYNONYM или CREATE PUBLIC SYNONYM, в зависимости от типа создаваемого синонима. При создании наличие реального объекта и привилегий доступа к нему не требуется.

CREATE SYNONYM [имя_синонима] FOR[имя_объекта]

CREATE PUBLIC SYNONYM [имя_синонима] FOR[имя_объекта]

В своей схеме пользователь может удалять любые частные синонимы. Для удаления частных синонимов в схеме другого пользователя, необходима привилегия DROP ANY SYNONYM, для удаления общих – DROP PUBLIC SYNONYM.

Для удаления используется оператор – DROP SYNONYM. Для удаления общего синонима - DROP PUBLIC SYNONYM.

При удалении синонима, его определение удаляется из словаря данных. Все объекты ссылающиеся на удаленный синоним остаются, однако они становятся недействительными и их использование невозможно. Объекты, для которых создавался синоним остаются без изменений и доступны для использования.

Если удален объект, для которого создавался синоним, то синоним остается, но при обращении к нему будет сообщено об ошибке. Если объект, для которого создавался синоним, был пересоздан, то синоним требуется перекомпилировать. Для этого используется предложение:

ALTER SYNONYM [имя_синонима] COMPILE;