Компьютерные технологии в физике элементарных частиц.

Базы данных

Для чего нужны базы данных?

- В современном мире производится огромное количество данных. Чтобы данные стали информацией, их необходимо структурировать и поместить в контекст. Именно для этого и нужны базы данных.
- Типичные базы данных в экспериментах в физике высоких энергий: заходов, файлов, состояния детектора, калибровок, заданий и много-много других.
- Системы управления базами данных (СУБД) – пакет программного обеспечения, который предназначен для управления большими объемами связанных между собой данных. Помимо управления данными СУБД помогает решить множество других вопросов: управление пользователями, ограничение доступа, расширяемость,…

Реляционная модель

Подавляющее количество существующих СУБД основаны на реляционной модели данных.

А какие еще модели бывают?

- сетевая, иерархическая, объектная,…

Принципы реляционной модели были впервые описаны в классической статье E.F. Codd. A Relational Model of Data for Large Shared Data Banks. Communications of the ACM, Volume 13, Number 6, June, 1970. Название «реляционная» проистекает от английского слова “relation”, которое в данном контексте означает «таблица».

Упрощенно, реляционная база данных – это коллекция двумерных таблиц. Оказалось, что такая модель отлично подходит для работы со структурированными данными, и, поэтому, со временем реляционные СУБД заняли абсолютно доминирующее положение.

С развитием Интернета появилось огромное количество неструктурированных данных, это дало толчок развитию других подходов. О них мы поговорим позже.

Распространенные решения

Существует множество коммерческих и свободных СУБД:

Коммерческие СУБД	Свободные СУБД
Oracle (1979-) – лидер рынка. В частности, используется в CERN для экспериментов на LHC. IBM DB2 (1983-) – язык SQL появился именно в экспериментальной версии предшественника DB2. Microsoft SQL Server (1988-) Microsoft Access – настольная СУБД.	PostgreSQL (1997-) – самая полнофункциональная свободная СУБД. Используется многими небольшими и средними экспериментами. MySQL (1998-) – изначально упрощенная, но быстрая СУБД, популярная при разработке web-приложений. Сейчас полнофункциональная (в зависимости от типа таблицы). В настоящее время куплена Oracle.

реляционной модели, максимальными объемами базы данных, уровнем реализации стандарта SQL, возможностью работы с множеством клиентов, возможностью работы в распределенном режиме и т.п.

Язык SQL

Для работы с базой данных используется язык SQL (Structured Query Language).

- SQL – декларативный язык программирования, т.е. пользователь описывает не порядок действий («программу»), а описывает то, что он хочет получить. Сервер БД сам «решает», как достичь требуемого результата.
- SQL позволяет описать структуру таблиц, связи между ними, накладываемые ограничения.
- SQL содержит большое количество механизмов для манипулирования данными.

Существует множество диалектов SQL – каждая СУБД на этапе своего становления развивала собственную версию. Первый стандарт SQL появился в 1986, после чего стандарт регулярно обновлялся. Последний стандарт был опубликован в 2008.

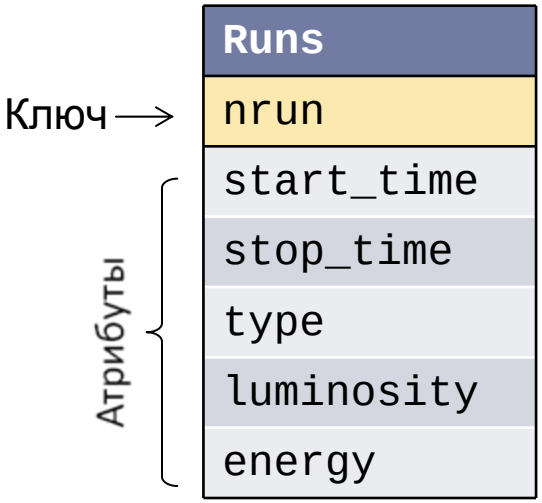
Все примеры в этой лекции используют вариант SQL, поддерживаемый СУБД PostgreSQL.

Проектируем базу данных заходов

Задача: для проведения эксперимента потребовалось спроектировать базу данных заходов.

Требования: Заход – минимальный блок записанной статистики, набранный в стабильных условиях. Заход идентифицируется номером (это первичный ключ). Заход характеризуется несколькими атрибутами: время начала и конца, тип, интеграл светимости.

Модель данных ("сущность")



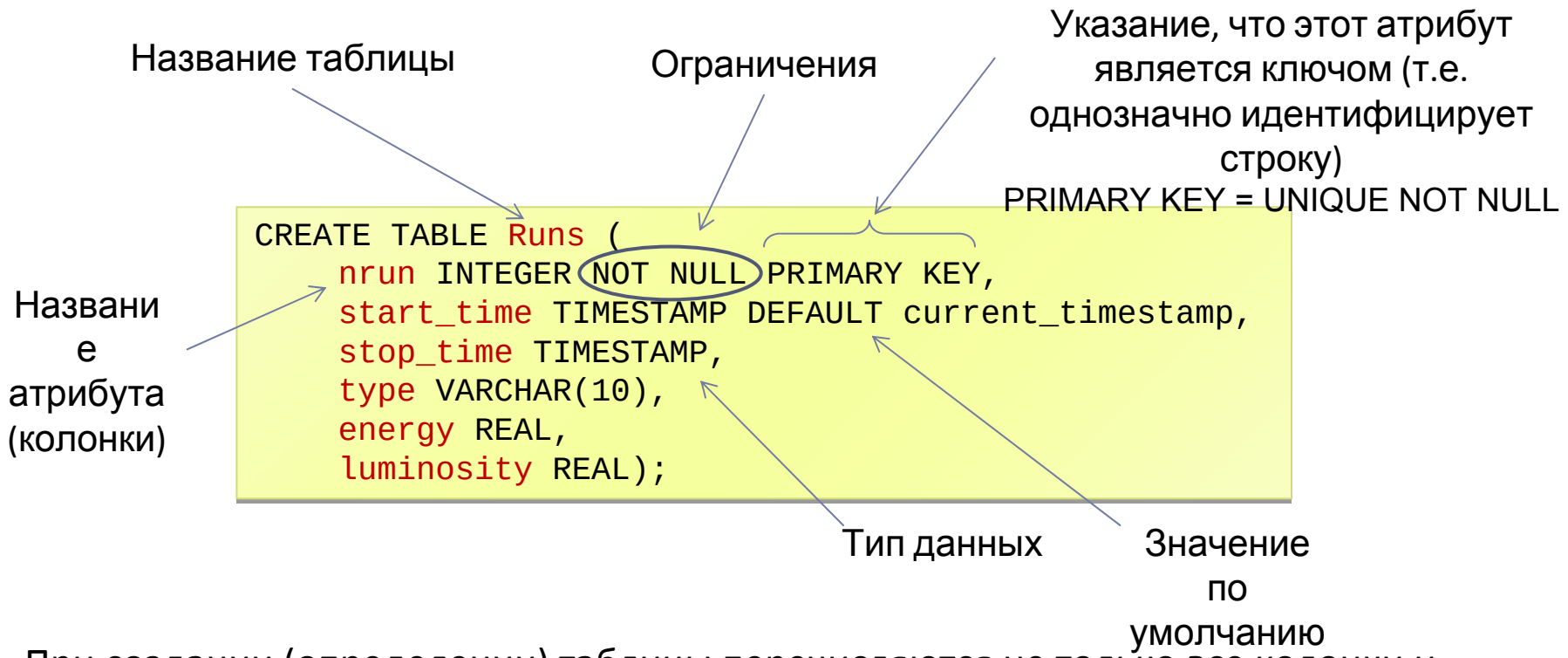
На языке SQL

```
CREATE TABLE Runs (
    nrun INTEGER NOT NULL PRIMARY KEY,
    start_time TIMESTAMP DEFAULT current_timestamp,
    stop_time TIMESTAMP,
    type VARCHAR(10), energy REAL,
    luminosity REAL);
```

Вот что получилось:

nrun	start_time	stop_time	type	luminosity
1234	'2012-03-08 01:02:03'	'2012-03-08 02:01:00'	'beam'	12.34
1235	'2012-03-08 02:02:01'	'2012-03-08 02:30:43'	'cosmic'	0.0

Посмотрим внимательнее на команду CREATE



При создании (определении) таблицы перечисляются не только все колонки и типы данных, которые в них хранятся, но и дополнительная информация, позволяющая более точно описать свойства таблицы: какие колонки являются ключами; ограничения на возможные значения (уникальность, диапазоны значений,…); значения по умолчанию; и другие.

Типы данных в SQL

В реляционной базе данных могут храниться данные различного типа. Распространенные типы данных в Postgres:

Числовые	
INTEGER	Целые числа, 4 байта
REAL	Вещественные числа, 4 байта
NUMERIC	Числа с заданной точностью
SERIAL	Целые числа с автоинкрементом

Строковые	
VARCHAR(n)	Строки до n символов
TEXT	Строки произвольной длины

Дата/Время	
DATE	Дата
TIME	Время в течение дня
TIMESTAMP	Дата и время

Битовые	
BOOLEAN	«Да» или «Нет»
BIT(n)	Битовая маска из n бит
BYTEA	Байтовая строка произвольной длины. Аналог BLOB в других СУБД.

Разное	
XML	Текстовые данные в формате XML
MACADDR	MAC-адрес, например '08:00:2b:01:02:03'
INET	IP-адрес, например '192.168.100.128/25'

Добавляем информацию о файлах

Пусть нам потребовалось добавить в БД информацию о том, где лежат файлы заходов.

Требования: Каждому заходу соответствует один уникальный файл. Несколько копий этого файла может лежать на разных компьютерах.

Вот что мы хотим получить:

nrun	start_time	stop_time	type	luminosity	md5	location
1234	'2012-03-08 01:02:03'	'2012-03-08 02:01:00'	'beam'	12.34	595f44fec1e92a71d3e9e77456ba80d1	'host1:/dir1/1234.root'
1235	'2012-03-08 02:02:01'	'2012-03-08 02:30:43'	'cosmic'	0.0	71f920fa275127a7b60fa4d4d41432a3	'host1:/dir1/1235.root' 'host2:/dir2/1235.root'

Проблема: в некоторых ячейках хранится несколько значений – это невозможно.
Простое решение: создаем отдельную строчку для каждого их этих значений.

nrun	start_time	…	md5	host	dir	filename
1234	'2012-03-08 01:02:03'		595f44fec1e92a71d3e9e77456ba80d1	'host1'	'dir1'	'1234.root'
1235	'2012-03-08 02:02:01'		71f920fa275127a7b60fa4d4d41432a3	'host1'	'dir1'	'1235.root'
1235	'2012-03-08 02:02:01'		71f920fa275127a7b60fa4d4d41432a3	'host2'	'dir2'	'1235.root'

Недостаток: много информации дублируется

Нормализация таблиц

Для того, чтобы избежать дублирования информации, каждая таблица должна соответствовать одной «сущности». Процесс выделения независимых сущностей в отдельные таблицы называется «нормализацией».

Нормализация – разбиение одной таблицы на две и более таким образом, чтобы получившиеся таблицы обладали «лучшими» свойствами. Цель: добиться, чтобы любая информация появлялась в базе данных только в одном месте.

Уровни нормализации: первая нормальная форма (1НФ), вторая НФ (2НФ), 3НФ, 4НФ, 5НФ

1НФ	Строка и столбец в таблице однозначно определяют значение. Все таблицы в реляционной СУБД находятся в 1НФ.
2НФ	Таблица удовлетворяет 1НФ и все поля, не входящие в ключ таблицы, зависят только от полного значения ключа, а не от какого-то подмножества его составляющих.
3НФ	2НФ + поля, не входящие в ключ (атрибуты), не зависят напрямую друг от друга

Процесс нормализации: последовательно разбиваем таблицы на несколько отдельных таблиц, пока они все не будут находиться в 3НФ.

Нормализация таблицы заходов

Ключом в таблице заходов является (nrun, host, dir, filename), а start_time зависит только от nrun.

Наша таблица не удовлетворяет требованиям 2НФ!

nrun	start_time	…	md5	host	dir	filename
1234	‘2012-03-08 01:02:03’		595f44fec1e92a71d3e9e77456ba80d1	‘host1’	‘dir1’	‘1234.root’
1235	‘2012-03-08 02:02:01’		71f920fa275127a7b60fa4d4d41432a3	‘host1’	‘dir1’	‘1235.root’
1235	‘2012-03-08 02:02:01’		71f920fa275127a7b60fa4d4d41432a3	‘host2’	‘dir2’	‘1235.root’

Разбиваем ее на две

Ключ nrun

nrun	start_time	…	md5
1234	‘2012-03-08 01:02:03’		595f44fec1e92a71d3e9e77456ba80d1
1235	‘2012-03-08 02:02:01’		71f920fa275127a7b60fa4d4d41432a3
1235	‘2012-03-08 02:02:01’		71f920fa275127a7b60fa4d4d41432a3

Подумайте: почему поле md5 осталось в этой таблице?

nrun	host	dir	filename
1234	‘host1’	‘dir1’	‘1234.root’
1235	‘host1’	‘dir1’	‘1235.root’
1235	‘host2’	‘dir2’	‘1235.root’

Подумайте: что является ключом в этой таблице и удовлетворяет ли она требованиям 2НФ и 3НФ? Подсказка: ответ зависит от контекста.

Создадим новые таблицы в SQL

Указание, что этот атрибут является внешним ключом, т.е. описывает связь с другой сущностью. Т.е. мы явно указываем, что nrun в обеих таблицах несет один и тот же смысл.

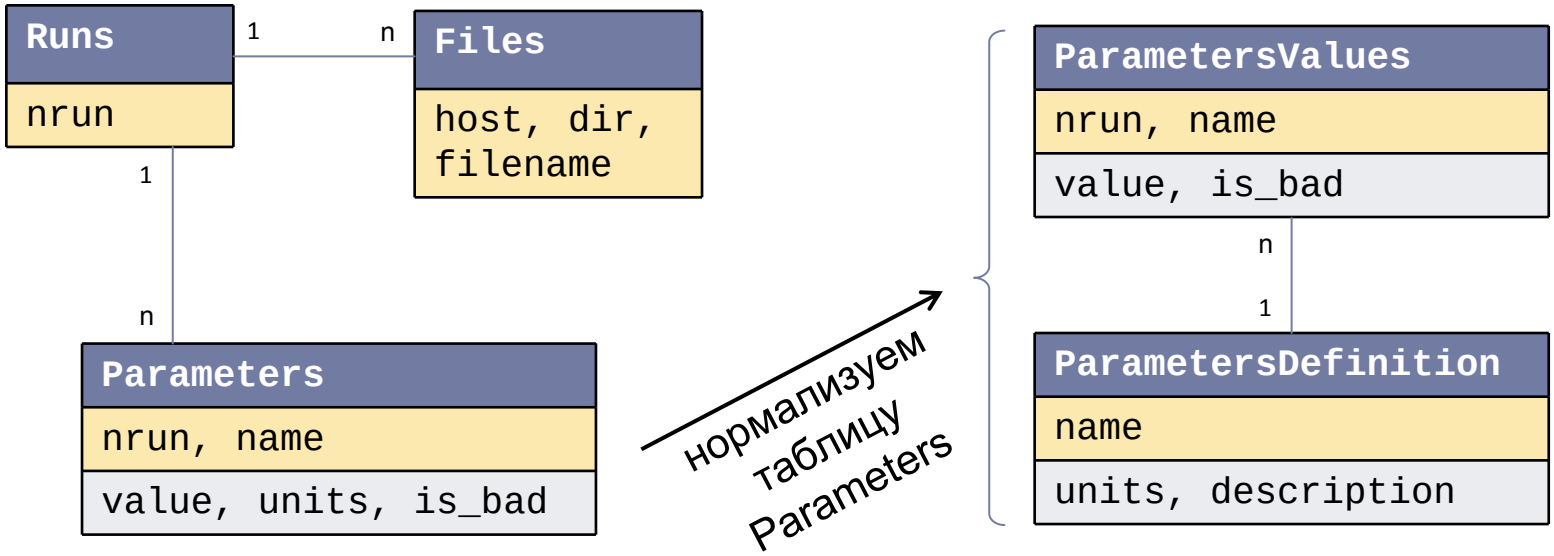
```
CREATE TABLE Runs (
     nrun INTEGER NOT NULL PRIMARY KEY,
     start_time TIMESTAMP DEFAULT current_timestamp,
     stop_time TIMESTAMP,
     type VARCHAR(10),
     luminosity REAL,
     energy REAL,
     md5 VARCHAR(32) );

CREATE TABLE Files (
     nrun INTEGER REFERENCES Runs(nrun),
     host VARCHAR(80),
     dir VARCHAR(80),
     filename VARCHAR(80),
     PRIMARY KEY (host, dir, filename) );
```

Составной первичный ключ можно описать таким образом. Этот пример не очень хороший, т.к. не рекомендуется в качестве ключей использовать строковые переменные.

Модель «сущность-связь» (Entity-Relation)

Нормализация используется как правило на последнем этапе проектирования БД. Начинается проектирование с построения концептуальной модели базы данных, для этого удобно использовать диаграмму «сущность-связь».

В качестве примера сформируем модель нашей БД. Информацию о заходах и файлах мы уже обсудили. Добавим теперь еще информацию о параметрах захода. Параметры – это любые величины, которые характеризуют заход. Их список может отличаться для разных заходов. Параметры характеризуются именем, значением, единицей измерения, статусом (плохое или хорошее значение).



Создание таблиц на основе модели «сущность-связь»

Существуют простые эмпирические правила, как перевести модель «сущность-связь» в набор таблиц реляционной БД (этап построения логической модели БД):

- Для каждой сущности создается отдельная таблица.
- Каждый тип связи описывается внешними ключами или таблицей связи.

Если для каждой строки в таблице A есть ровно одна строка в таблице B, то можно или создать единую таблицу, в которой объединить колонки из A и B, или использовать решение для связи «1-в-n»

Если для каждой строки в таблице A может быть несколько строчек в таблице B, ключ таблицы A должен быть определен в таблице B как внешний ключ.

Если для каждой строки в таблице A может быть несколько строчек в таблице B, и для каждой строки в таблице B может быть несколько строчек в таблице A, то нужно создать отдельную таблицу связей AB, в которой хранятся ключи обеих таблиц.

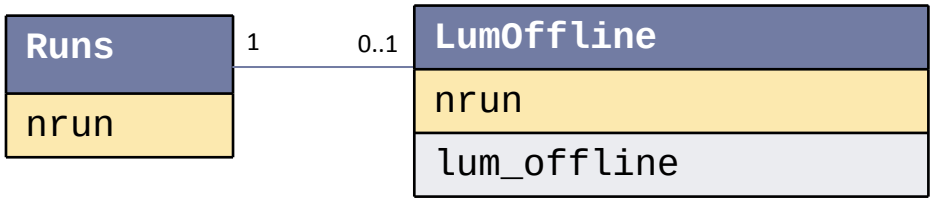
Расширяемость базы данных

Важным свойством реляционных баз данных является расширяемость.

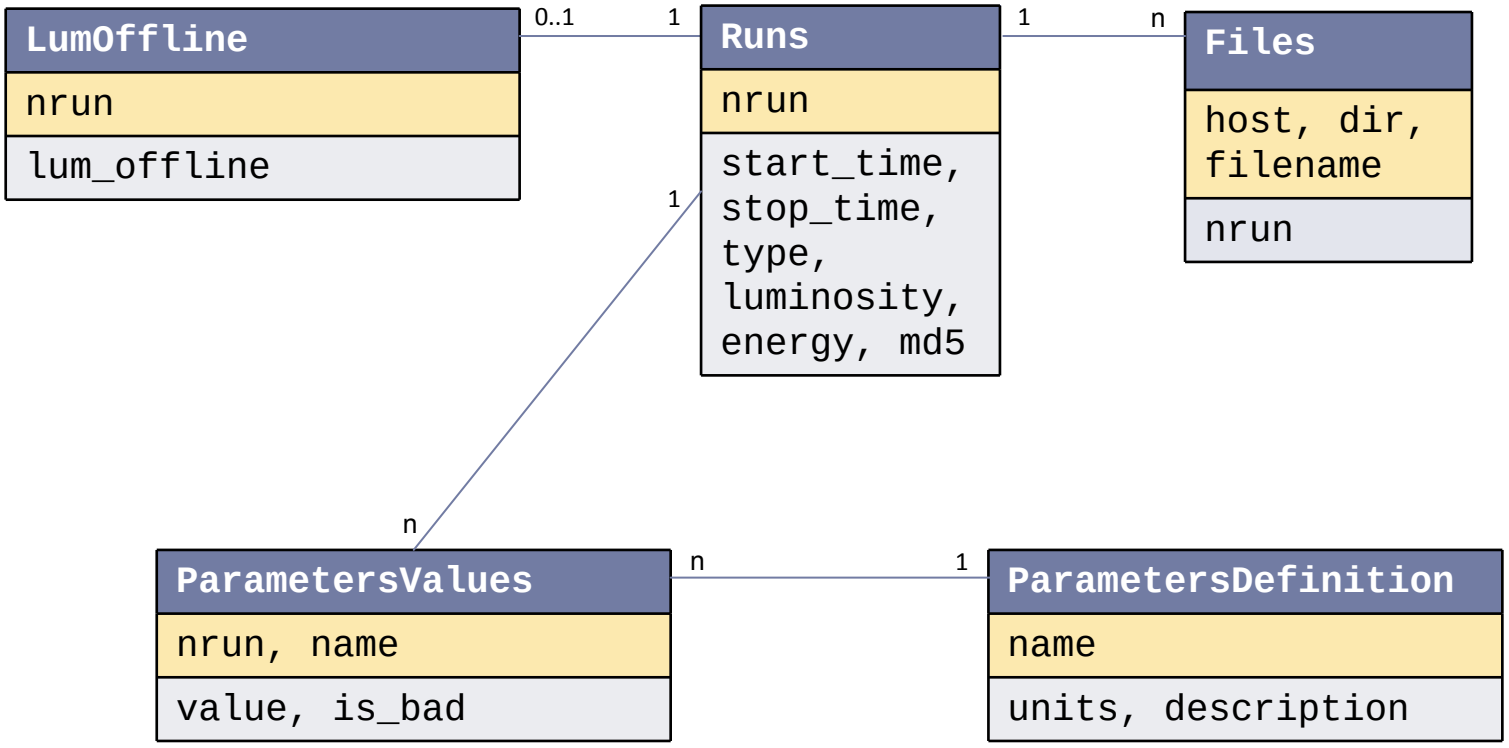
Например, поле luminosity в таблице Runs содержит интеграл светимости, измеренный с помощью монитора светимости. Представим, что через какое то время коллаборация эксперимента разработала новую методику измерения светимости, используя полную реконструкцию записанных данных. Как добавить эту информацию в базу данных?

Можно изменить исходную таблицу Runs, но не всегда такое решение не оптимально – например, большое количество программного обеспечения уже использует старое определение таблицы, и, возможно, изменение таблицы приведет к переписыванию программ (замечание: если это произошло, то ПО написано неправильно).

Решение, естественное с точки зрения реляционной БД: добавляем новую таблицу, которая будет содержать новое значение светимости (поле lum_offline) и связываем ее с исходной таблицей Runs. Дополнительные преимущества этого решения: так можно сохранить не только значение светимости, но и сопутствующие параметры; такой подход легко повторяется при появлении еще одной методики.



Окончательная модель базы данных заходов

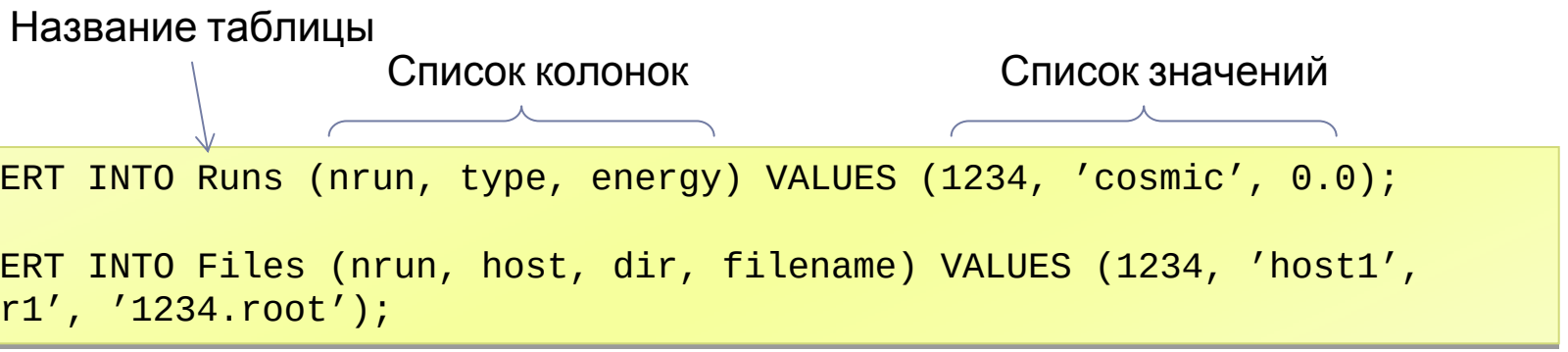


Окончательное описание базы данных на языке SQL

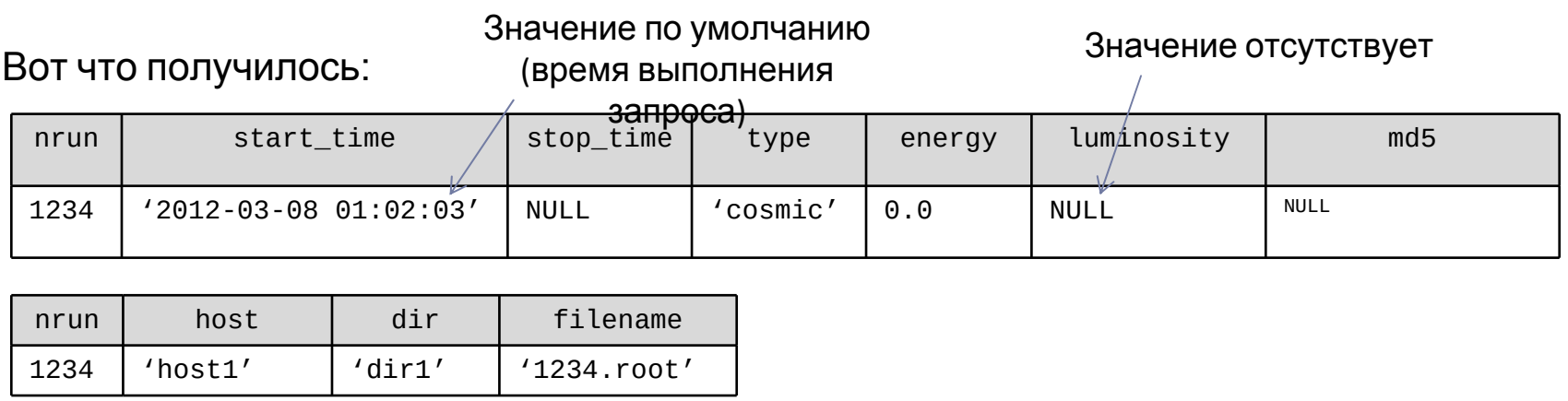

```
CREATE TABLE Runs (
     nrun INTEGER NOT NULL PRIMARY KEY,
     start_time TIMESTAMP DEFAULT current_timestamp, stop_time TIMESTAMP,
     type VARCHAR(10), luminosity REAL, energy REAL, md5 VARCHAR(32) );

CREATE TABLE Files (
     nrun INTEGER REFERENCES Runs(nrun),
     host VARCHAR(80), dir VARCHAR(80), filename VARCHAR(80),
     PRIMARY KEY (host, dir, filename) );

CREATE TABLE ParametersDefinition(
     name VARCHAR(20) PRIMARY KEY,
     units VARCHAR(20), description TEXT );

CREATE TABLE ParametersValues(
     nrun INTEGER REFERENCES Runs(nrun),
     name VARCHAR(20) REFERENCES ParametersDefinition (name),
     value REAL, is_bad BOOLEAN,
     PRIMARY KEY (nrun, name) );

CREATE TABLE LumOffline(
     nrun INTEGER PRIMARY KEY REFERENCES Runs(nrun),
     lum_offline REAL);
```

Заполняем базу данных (INSERT)

Для заполнения базы данных (создания строк) используется команда INSERT:



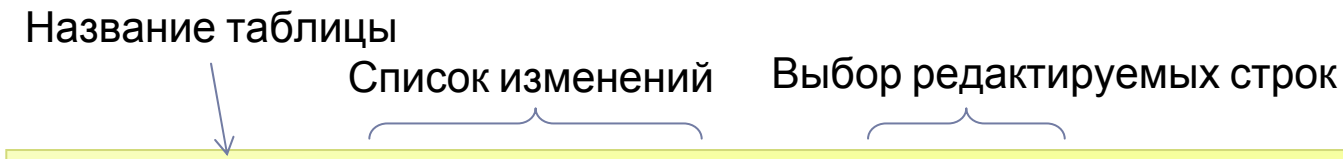
Вот что получилось:



Как правило, база данных заполняется с помощью специальных программ, которые получают информацию из внешних источников (электроники, веб-формы,...).

Редактируем базу данных (INSERT)

Для изменения базы данных (редактирования строк) используется команда UPDATE:




```
UPDATE Runs SET stop_time=NOW() WHERE nrun=1234;
```

Вот что получилось:

Runs

nrun	start_time	stop_time	type	energy	luminosity	md5
1234	‘2012-03-08 01:02:03’	‘2012-03-08 02:01:01’	‘cosmic’	0.0	NULL	NULL

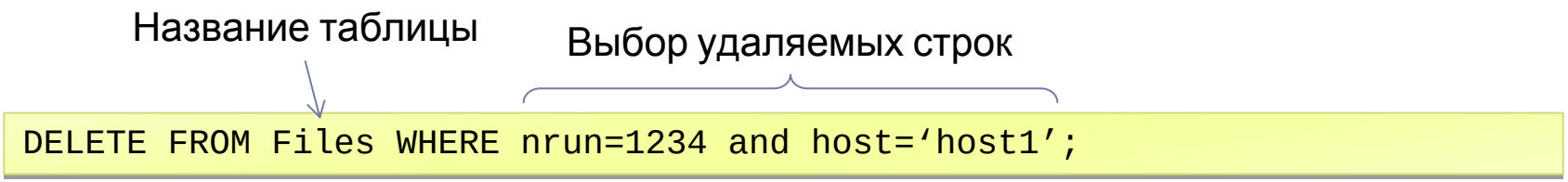
С помощью команды UPDATE можно редактировать много строчек одновременно.

Например, пусть на машине ‘host1’ все файлы были перенесены из директории ‘dir1’ в директорию ‘newdir’. В этом случае обновить информацию в базе данных можно следующим образом:

```
UPDATE Files SET dir=‘newdir’ where host=‘host1’ and dir=‘dir1’;
```

Удаляем записи из базы данных (DELETE)

Для удаления записей из базы данных используется команда DELETE:



Все строки, которые удовлетворяют условиям отбора, указанным после ключевого слова WHERE, будут удалены.

Осторожно! `DELETE FROM Files;` удалит все записи из таблицы Files!

Что произойдет, если мы удалим информацию о заходе 1234 из таблицы Runs? Ведь этот номер захода используется в таблице Files как внешний ключ.

В зависимости от настроек таблицы (см. подробное описание команды CREATE в документации по СУБД), возможны следующие сценарии:

- СУБД не позволит удалить эту запись, пока есть хоть одна связанная с ней запись в других таблицах;
- СУБД удалит все связанные с этой записью записи в других таблицах.

В любом случае, целостность СУБД будет сохранена.

Чтение информации из базы данных (SELECT)

Для извлечения информации из базы данных используется команда SELECT.

Например: получаем список заходов, записанных 8 марта 2012 года:

В команде SELECT указывается, какую информацию мы хотим получить (т.е. названия колонок и название таблицы) и условия отбора. В качестве списка колонок можно указывать *, тогда будут выданы все колонки, определенные в таблице:

```
SELECT * FROM Runs WHERE date(start_time)='2012-03-08';
```

В команде SELECT можно использовать встроенные функции SQL, например, функцию date(). Существует множество встроенных функций, которые позволяют приводить типы, оперировать со строками, временем, датами и временными интервалами, вычислять математические выражения, работать с XML и т.п.

Объединение таблиц

Очень часто требуется объединить информацию из нескольких таблиц. Например, если мы хотим получить список файлов для заходов, записанных 8 марта 2012 года, нам необходимо извлечь данные из двух таблиц: Runs и Files.

Полное объединение двух таблиц (декартово произведение):

A

A1	A2
1	One
2	Two

B

B1	B2
1	Uno
2	Due

```
SELECT * FROM A, B;
```

A1	A2	B1	B2
1	One	1	Uno
1	One	2	Due
2	Two	1	Uno
2	Two	2	Due

Обычно требуется не полное объединение, а добавление колонок к одной из таблиц. Для этого необходимо указать, какие строки относятся к одной и той же сущности:

A

A1	A2
1	One
2	Two

B

B1	B2
1	Uno
2	Due

```
SELECT A1,A2,B2 FROM A,B WHERE A1=B1;
```

A1	A2	B2
1	One	Uno
2	Two	Due

INNER JOIN

Более удобным способом объединения таблиц является использования ключевого слова JOIN. Такой подход позволяет не смешивать в одном выражении WHERE условия, относящиеся к объединению таблиц, и критерии отбора.

Синтаксис: `FROM A INNER JOIN B ON A.A1=B.B1` (вместо FROM A,B WHERE A.A1=B.B1)

Ключевое слово INNER можно опускать.

Если таблицы склеиваются по одноименной колонке, то можно использоват `FROM Runs INNER JOIN Files USING (nrun)`

Пример:

```
SELECT F.host,F.dir,F.filename
        FROM Files F INNER JOIN Runs R USING (nrun)
        WHERE date(start_time)='2012-03-08';
```

8 марта 2012 года

```
SELECT host,dir,filename FROM Files NATURAL JOIN Runs WHERE date(start_time)='2012-03-08';
```

OUTER JOIN

Если для какой то строки в одной из склеиваемых таблиц нет парной строки в другой таблице, то такая строка не будет выбрана INNER JOIN. Если такие строки должны быть выбраны, то для склейки нужно использовать одну из форм OUTER JOIN.

A

A1	A2
1	One
2	Two

B

B1	B2
1	Uno
3	Tre

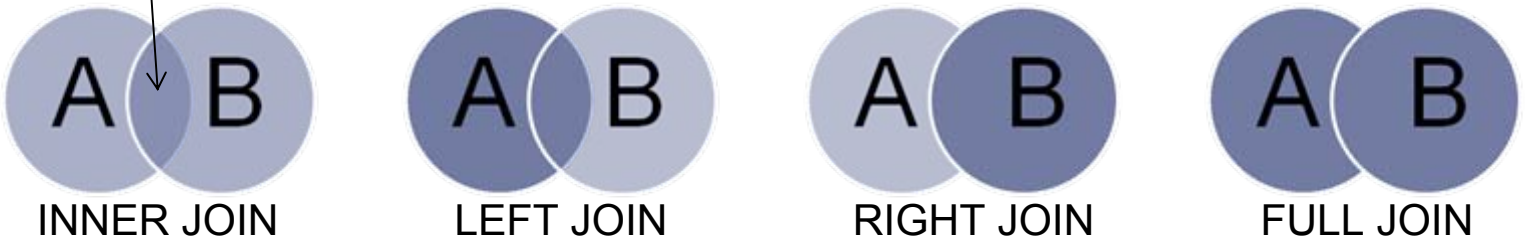
```
                          LEFT  JOIN
SELECT A1,A2,B2 FROM A RIGHT JOIN B ON A.A1=B.B1
                          FULL  JOIN
```

A1	A2	B2
1	One	Uno
2	Two	

A1	A2	B2
1	One	Uno
		Tre

A1	A2	B2
1	One	Uno
2	Two	
		Tre

Строки, прошедшие отбор



Примеры чтения данных из нескольких таблиц

1. Получим список всех заходов, записанных 8 марта 2012 года, и если файл захода храниться на host1, получим его местоположение:

```
SELECT R.nrun,dir,filename
   FROM Runs R LEFT JOIN Files F ON R.nrun=F.nrun AND F.host='host1'
   WHERE date(start_time)='2012-03-08';
```

2. Получим информацию о заходах, для которых есть параметры с «плохими» значениями

Многократный JOIN

```
SELECT R.nrun,P.name,P.value,D.units
   FROM Runs R JOIN ParametersValues P USING (nrun)
   LEFT JOIN ParametersDefinition D USING (name)
   WHERE is_bad;
```

3. Получим список заходов, которые были запущены более чем через час после остановки предыдущего захода

JOIN таблицы со своей копией

```
SELECT R2.nrun,R2.start_time
   FROM Runs R1 JOIN Runs R2 ON R2.nrun=R1.nrun+1
   WHERE R2.start_time>(R1.stop_time+INTERVAL '1 hour');
```

Представления (VIEW)

Построение эффективных многоуровневых запросов к БД требует от пользователя экспертных знаний. Для упрощения доступа к БД в SQL существует механизм представлений.

Представление – «замороженный» SELECT-запрос, с помощью которого создается виртуальная таблица. Конечный пользователь работает с представлением ровно так же, как с настоящей таблицей (только в режиме чтения; изменение данных в представлении возможно только в определенных случаях). Использование представлений позволяет сделать нормализованную БД удобной для конечных пользователей.

Пример: создадим представление, в котором будет интегрирована вся информация про светимость для «пучковых» заходов:

```
CREATE VIEW RunsLuminosity AS
   SELECT nrun, luminosity as lonline, lum_offline as loffline
      FROM Runs LEFT JOIN LumOffline USING (nrun)
      WHERE type='beam';
```

Сравним online и offline светимость:

```
SELECT nrun,(loffline/lonline-1) FROM RunsLuminosity WHERE lonline>0;
```

Получение статистической информации

Часто из базы данных требуется получить информацию, объединенную по каким либо параметрам. Для этого в SELECT-запросе используются агрегатные функции и ключевые слова GROUP BY и HAVING.

Агрегатные функции: MIN(), MAX(), SUM(), COUNT(), AVG(),…

Примеры:

1. Для каждого «пучкового» захода посчитаем, сколько копий файла захода храниться в системе

```
SELECT nrun, COUNT(filename) as nfile
   FROM Runs LEFT JOIN Files USING (nrun) WHERE type='beam'
   GROUP BY nrun;
```

2. Отберем только те заходы, для которых существует более 2 копий:

```
SELECT nrun, COUNT(filename)
   FROM Runs LEFT JOIN Files USING (nrun) WHERE type='beam'
   GROUP BY nrun HAVING count(filename)>2;
```

Проверяется до группирования

Проверяется после группирования

3. Интеграл светимости, набранный в течении каждого дня:

```
SELECT DATE(start_time) as date, SUM(luminosity) as daylum
   FROM Runs GROUP BY date;
```

Вложенные запросы

Современные реляционные базы данных позволяют использовать вложенные SELECT-запросы внутри других команд.

Примеры:

1. Для каждого дня выберем заход с максимальным интегралом светимости:

```
SELECT date(start_time) as date,nrun,luminosity FROM Runs R1
  WHERE luminosity = (SELECT MAX(luminosity) from Runs R2
                      WHERE
date(R2.start_time)=date(R1.start_time));
```

2. Получим список заходов, у которых нет плохих параметров

```
SELECT nrun FROM Runs
   WHERE nrun NOT IN (SELECT DISTINCT nrun FROM ParametersValues
                      WHERE is_bad);
```

3. Отберем те заходы, для которых существует более 2 копий в файловой системе

```
SELECT nrun, count
   FROM (SELECT nrun, COUNT(filename)
         FROM Runs LEFT JOIN Files USING (nrun) GROUP BY nrun) T
   WHERE count>2;
```

О чем я умолчал

Технологии реляционных СУБД развиваются уже несколько десятилетий. Приведенные выше сведения достаточны для первоначального знакомства с основными понятиями и для использования БД на уровне простого пользователя. Существует множество аспектов реляционных СУБД, которые не были упомянуты:

- транзакции
- хранимые функции и процедуры
- триггеры
- управление пользователями и правами доступа
- администрирование
- индексы и оптимизация
- …

Для более глубокого знакомства с технологией рекомендую обратиться к многочисленной литературе, часть из которой перечислена в конце лекции.

Задания

- Задание 1

  Расширьте базу данных заходов, добавив туда информацию о точках по энергии. Точка по энергии – период времени, когда ускоритель работает в одинаковых условиях (при одной энергии) и происходит набор экспериментальной статистики.

- Задание 2

  Спроектируйте базу данных электроники системы сбора данных. Требования: электроника представляет собой отдельные платы, установленные в крейтах. Каждая плата обладает собственным сетевым интерфейсом, который идентифицируется MAC-адресом. Платы логически объединяются в группы по подсистемам детектора. Необходимо иметь возможность описывать множество различных конфигураций системы сбора данных, которые отличаются списком включенных подсистем.

Структурированные и неструктурированные данные.

Становление технологии реляционных СУБД происходило в 1980-1990 годы. В ту эпоху, как правило, владельцем данных был тот, кто произвел эти данные, поэтому данные были структурированы – т.е. их структура была четко описана и предопределена. Именно такие данные описывает реляционная модель.

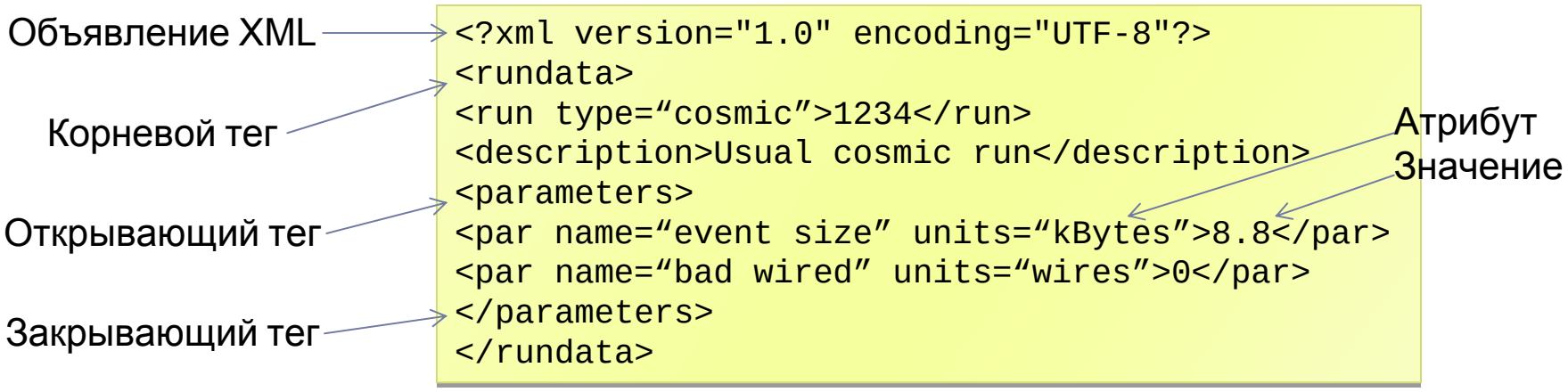
С конца 1990-х годов количество данных стало нарастать взрывным образом, в первую очередь, благодаря развитию Интернета. Кроме того, поменялся и характер данных – они стали гораздо более многообразными. Огромное количество данных сегодня неструктурированны – т.е. они не укладываются в какую то предопределенную структуру. Несмотря на термин, у неструктурированых данных есть структура – но она очень гибкая. Пример неструктурированных данных: база данных веб-страниц (например, Google); журнал эксперимента и т.п.

В последнее десятилетие технологии реляционных СУБД активно развиваются в направлении обработки неструктурированных данных. Параллельно развиваются и другие технологии – «нереляционные» СУБД, позволяющие обрабатывать большие объемы неструктурированных данных.

XML

Один из самых распространенных способов описания текстовых документов с гибкой структурой – использование языка XML (eXtensible Markup Language). XML широко используется описания документов (современные форматы хранения файлов Microsoft Word и Open Office основаны на XML), для обмена информацией между программами.

Документ в формате XML представляет собой иерархическую структуру элементов, каждый идентифицируется тегом, может принимать некоторое значение и характеризоваться рядом атрибутов:

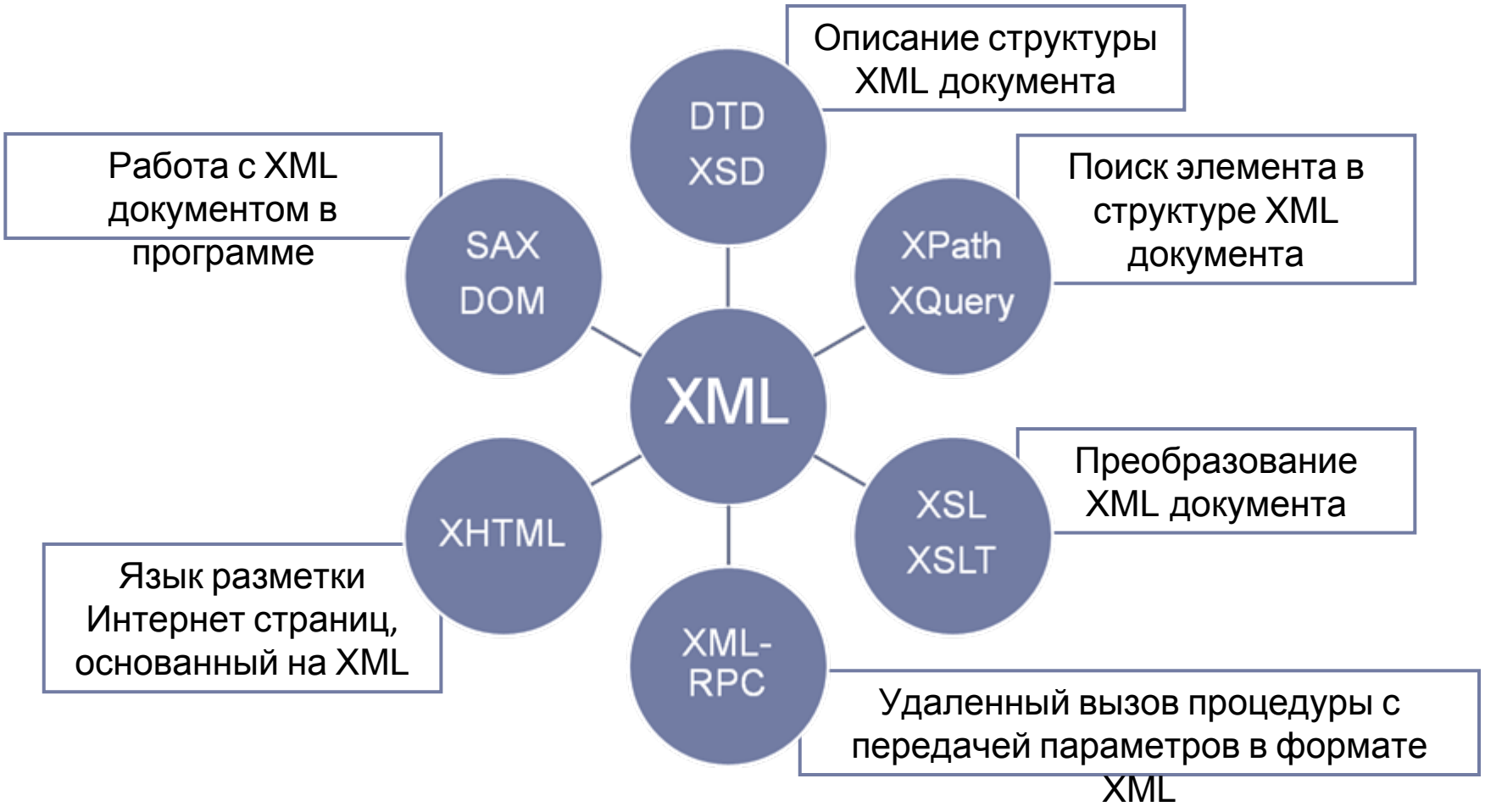



```
<?xml version="1.0" encoding="UTF-8"?>
<rundata>
<run type="cosmic">1234</run>
<description>Usual cosmic run</description>
<parameters>
<par name="event size" units="kBytes">8.8</par>
<par name="bad wired" units="wires">0</par>
</parameters>
</rundata>
```

Каждый документ в формате XML фактически представляет собой базу данных.

Технологии, связанные с XML

XML – это не только язык описания, но и целое семейство технологий, связанных с применением XML.



XML и SQL

Почему более гибкий XML не вытесняет реляционную модель и РСУБД?

- Реляционные СУБД на много порядков более эффективны при работе с большими объемами данных. Реляционные БД размером в несколько ТБ вполне обычны; с XML БД размером в несколько ГБ уже очень трудно работать.
- В реляционных СУБД значительно лучше решены вопросы расширяемости, параллельной обработки, безопасности, работы многих пользователей и т.п.

В большинстве современных РСУБД присутствует возможность непосредственной работы с XML документами (стандарт SQL/XML):

- Возможность хранить XML документ в специальной колонке таблицы
- Специальные функции, которые позволяют работать со структурой XML документа и извлекать нужную информацию (содержимое элементов)
- Специальные функции для выдачи результатов поиска в виде XML документа

Т.е. используется комбинированный подход – часть данных с предопределенной структурой хранится в СУБД «обычным» образом, а «гибкая» часть хранится в виде XML документа. Например, таким образом можно добавить в БД заходов журнал событий, произошедших во время захода.

Пример работы с XML в PostgreSQL

Добавим в нашу базу заходов поле, содержащее журнал событий:

```
ALTER TABLE Runs ADD COLUMN log XML;
```

1. Добавим журнал для захода 1234:

```
UPDATE Runs SET log=XMLPARSE(CONTENT
'<runlog><onshift><person>Ivan</person><person>John</person></onshift>
</runlog>') WHERE nrun=1234;
```

2. Добавим событие в журнал для захода 1234:

```
UPDATE Runs SET log=XMLCONCAT(log,
   XMLELEMENT(NAME event, XMLATTRIBUTES(now() AS time),'Beam is off'))
WHERE nrun=1234;
```

теперь поле log выглядит так

```
<onshift><person>Ivan</person><person>John</person></onshift>
<event time="2012-08-16 17:00:11.412376+07">Beam is off</event>
```

3. Получим список дежурных для каждого захода:

```
SELECT nrun, XPATH('/onshift/person/text()',log) FROM Runs
   WHERE log IS NOT NULL;
```

РСУБД и бинарные данные

В реляционной СУБД можно хранить не только текстовые документы в формате XML, но и любые большие бинарные объекты. В большинстве СУБД для этого существует тип данных BLOB (Binary Large Object). В PostgreSQL эту роль выполняет тип данных BYTEA. Используя BLOB в базе данных можно хранить картинки, аудиозаписи, сырые экспериментальные данные и т.п.

Возможности работы с BLOB ограничены (по очевидным причинам). Фактически, BLOB можно только записать в таблицу и прочитать из таблицы. Вся ключевая информация при этом должна содержаться в других колонках.

Использовать BLOB имеет смысл, когда общий объем данных, содержащийся во всех BLOB в СУБД, не превышает ожидаемого размера базы данных. Например, это подходящее решение для хранения картинок для небольшого веб-сервера, или для хранения калибровочной информации детектора.

Использовать BLOB для хранения данных эксперимента, как правило, неразумно – объем данных слишком большой. В этом случае в СУБД следует хранить только указатели на местоположение файлов в файловой системе, а не сами файлы (именно так устроена наша база данных заходов). Этот подход используется в подавляющем количестве экспериментов.

NoSQL решения

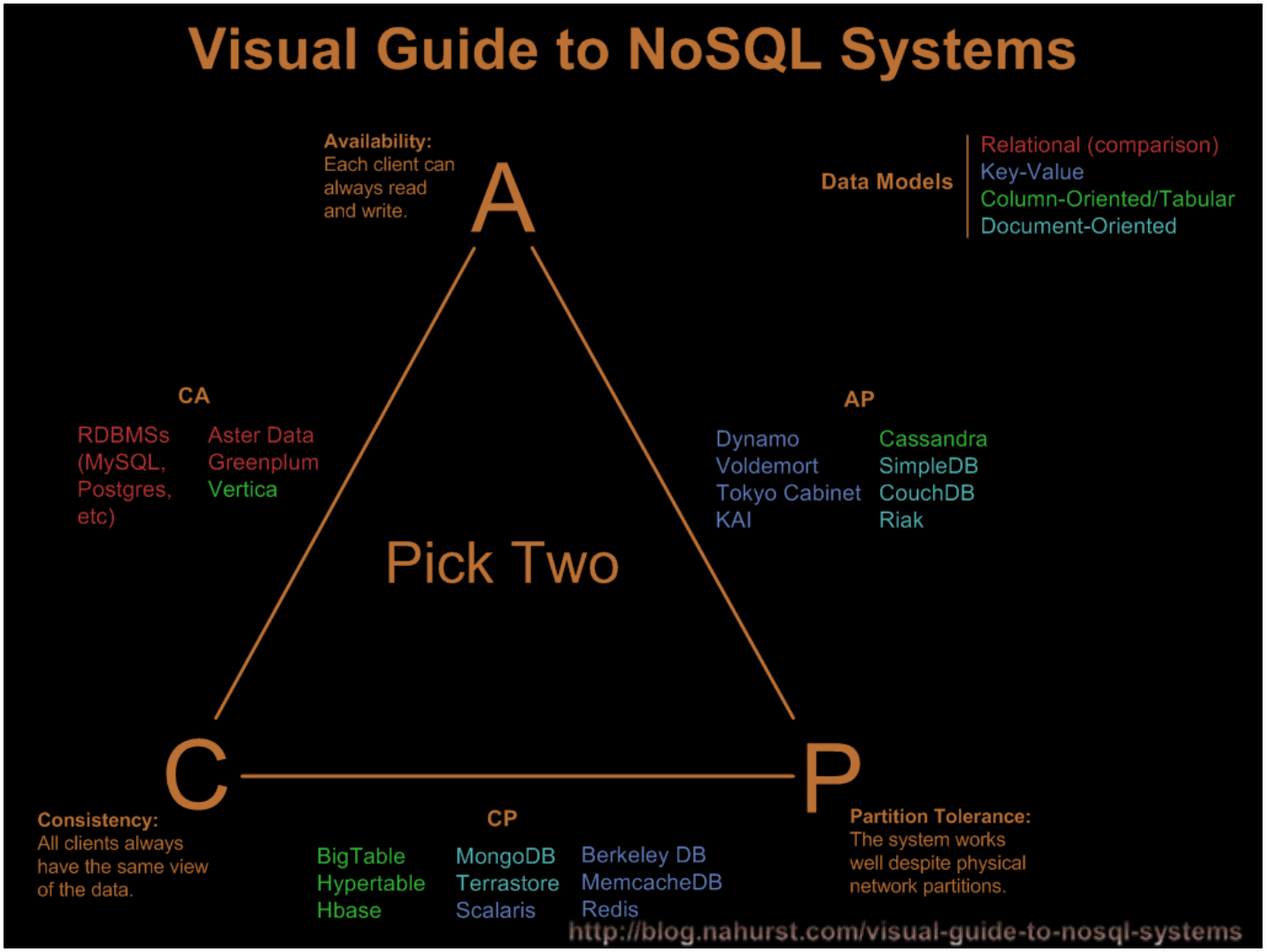
В последнее десятилетие активно развиваются новые, не основанные на реляционной модели, подходы к хранению и анализу очень больших объемов данных. Для таких технологий появилось общее название NoSQL (которое чаще всего переводится как “Not-only-SQL”). Общие характеристики таких технологий:

- Отступление от реляционной модели (хотя какие то ее черты продолжают присутствовать): возможность хранить не-атомарные значения в ячейках таблицы, использование модели данных «ключ-значение», отсутствие четкой схемы (модели) данных, язык запросов, не основанный на SQL, хранение многих версий данных, возможность временной несогласованности базы данных,…
- Хранение и анализ данных не на выделенном сервере, а на вычислительном кластере или «в облаке». Многие NoSQL решения ориентируются на обработку петабайтов данных, и для них характерна параллельная работа на 1000-10000 узлах

Все большие интернет-компании используют NoSQL для предоставления своих сервисов: Google, Amazon, Ebay, Twitter,…

Основные производители РСУБД также активно развивают это направление.

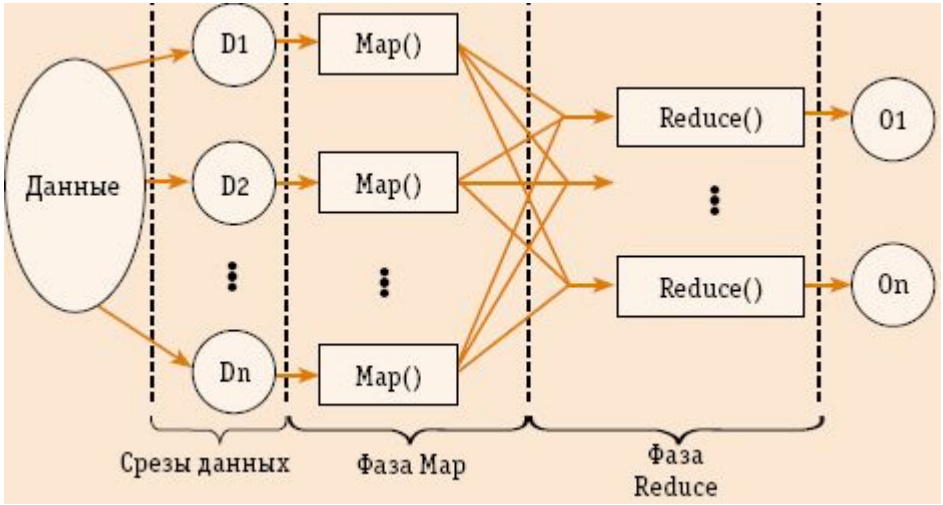
Спектр NoSQL решений



MapReduce

Во многих NoSQL решениях технология поиска (или, в более общем случае, обработки) данных основана на модели MapReduce. Название модели проистекает из названия функций map() и reduce(), часто применяемых в функциональном программировании.

Идея модели состоит в разделении процесса обработки на две стадии – предварительной обработки (map) и свертки (reduce). При этом для управления данными (поиск, обмен и т.п.) используется модель данных «ключ-значение».

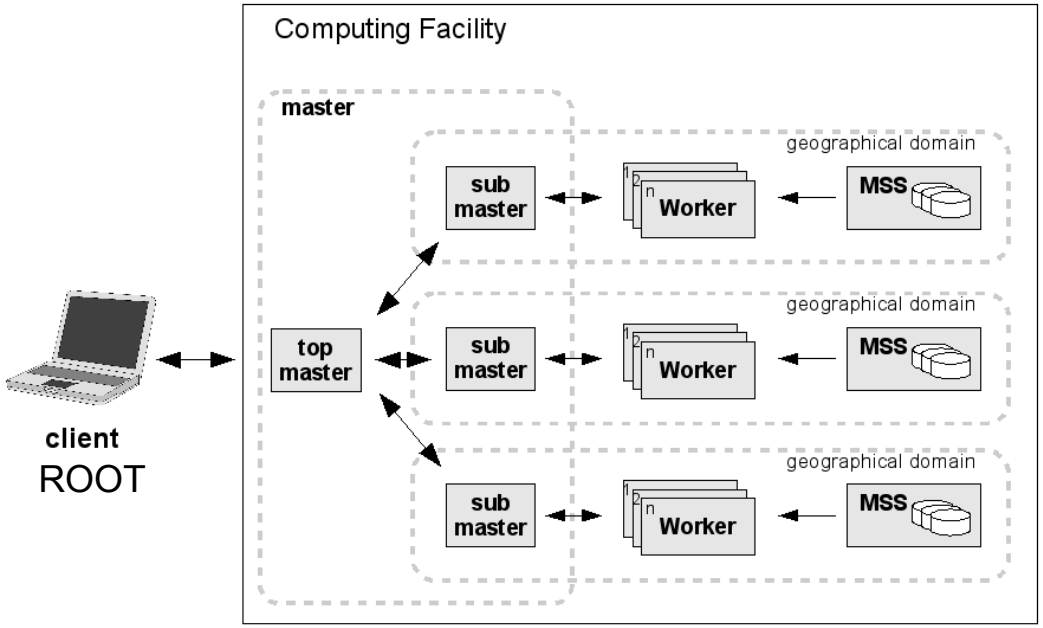


http://www.osp.ru/os/2009/02/7322603/

PROOF

Аналог MapReduce присутствует в фреймворке ROOT – компонент PROOF. Он позволяет пользователю производить параллельный анализ ROOT-файлов, которые могут храниться в распределенном файловом хранилище XROOTD.

Типичный сценарий: в дисковом хранилище хранятся ROOT-файлы для всех набранных заходов (10-100 тысяч файлов). Пользователь пишет программу, которая анализирует один файл и формирует гистограммы, и использует PROOF для того, чтобы проанализировать все файлы.



Что почитать

Технология реляционных баз данных развивается уже несколько десятилетий, поэтому существует огромное количество литературы на эту тему.

- Дейт К. Дж. Введение в системы баз данных, 8-е издание / Пер. с англ. — М.: Издательский дом "Вильямс", 2005. — 1328 с.: ил. – «фундаментальное» введение в технологии баз данных
- Линн Бейли, Изучаем SQL / Пер. с англ. — СпБ.: Питер, 2012. — 592 с.
- Энтони Молинаро, SQL. Сборник рецептов / Пер. с англ. — М.: Символ-Плюс, 2009. — 672 с.

Документация по СУБД на соответствующих интернет-сайтах

- PostgreSQL. http://www.postgresql.org/docs
- MySQL. http://dev.mysql.com/doc/
- Oracle. http://www.oracle.com/technetwork/database/enterprise-edition/documentation/index.html

Документация по NoSQL решениям

- http://nosql-database.org/ - ссылки на все основные NoSQL БД