Курсовая по теме: **«Основы реляционных баз данных. MySQL»**

Выполнил: Кузнецов Сергей
(Факультет Geek University Python-разработки)

Оглавление

1. Установка дистрибутива.

Создаем в табличном пространстве mySQL БД с именем «facebook» командой «*CREATE facebook;*» (можно выбрать любое др. наименование);

Для установки дистрибутива БД «facebook» можно пойти двумя путями.

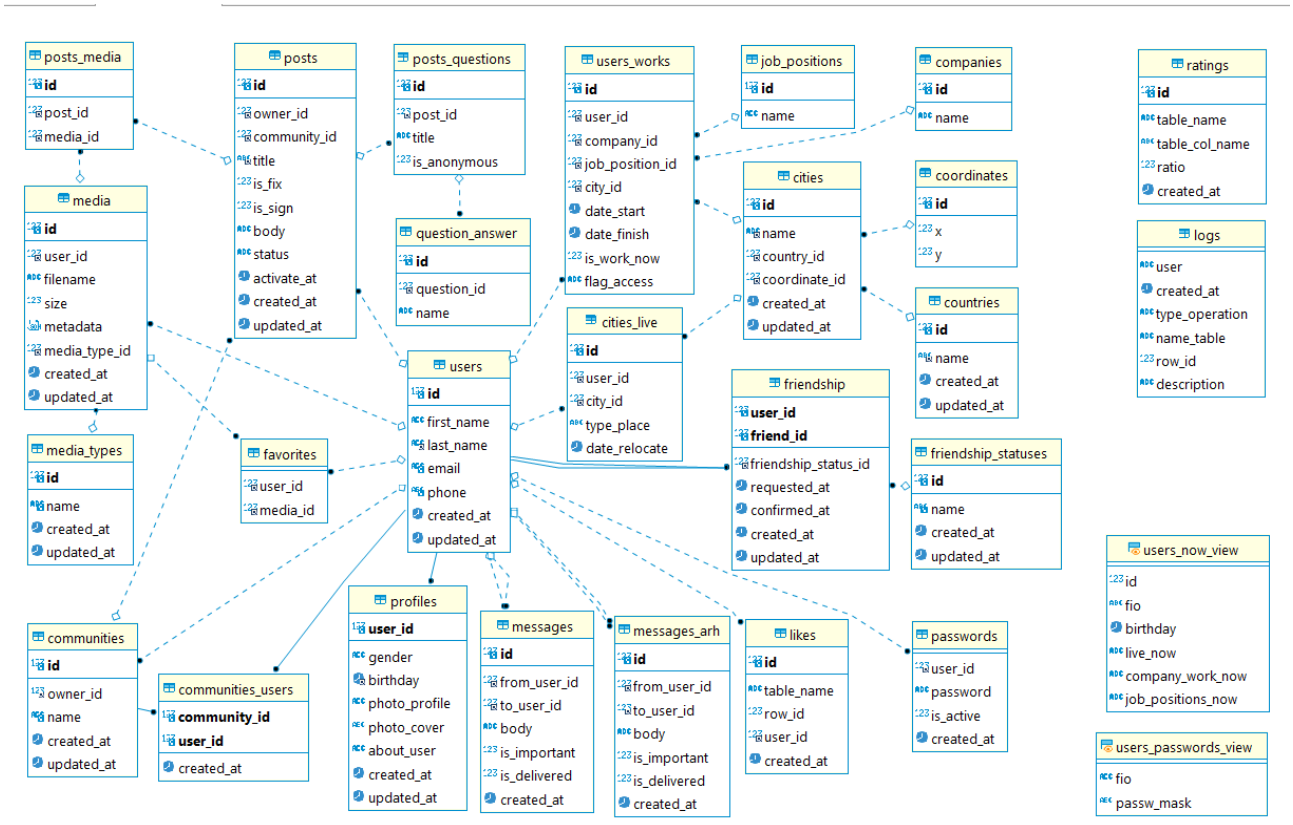
1) Последовательно:
   - Загружаем в терминале скрипт создания таблиц и представлений «FaceBook_tables_01.sql»
     «*mysql -u root -p facebook < FaceBook_tables_01.sql*» или
     «*mysql facebook < FaceBook_tables_01.sql*»
   - Добавляем индексы, триггеры, хранимые процедуры и функции
     «*mysql facebook < FaceBook_index_trigger_func_proc_02.sql*»
   - Загружаем в терминале скрипт с «сырыми» данными, сформированными с использованием сервиса http://filldb.info «*mysql facebook < FaceBook_dump_03.sql*»
   - Приводим «сырые» данные к более достоверному виду
     «*mysql facebook < FaceBook_convert_BD_04.sql*»
2) За одну итерацию:
   - Прогружаем дамп БД «*mysql facebook < FaceBook_dump_01_04.sql*»

2. Описание БД и решаемых ею задач.

Целью создания БД являлась демонстрация имеющихся навыков. Учитывая это, ряд полей ограничены типом ENUM, вместо ссылки на доп. таблицы. Какие-то поля отсутствуют, дабы не усложнять восприятие примера. Не все запросы обработки данных используют оптимальные форматы и практическую реализацию. Механизмы поддержания целостности БД (триггеры) приведены не на всех таблицах по тем-же причинам.

Стоит упомянуть, что в ходе разработки БД, был использован базисный функционал mySQL, такой как представления, первичные и вторичные ключи, триггеры, хранимые функции и процедуры, индексы, временные таблицы, курсоры, динамические запросы, CRUD-операции, в том числе window-функции, обработчики ошибок, генерации сигналов ошибки, администрирование доступом и т.д.

2.1. ERDiagram



2.2. Перечень таблиц

users	Пользователи системы	100
profiles	Продолжение таблицы users, включающая менее часто использующиеся поля *(разделены для оптимизации работы CRUD-операций)*	100
passwords	История изменений паролей пользователей	300
countries	справочник стран	20
coordinates	Координаты локаций (города, …)	50
cities	Справочник городов	50
cities_live	Города проживания пользователей по датам	500
companies	Справочник компаний	50
job_positions	Справочник должностей	50
users_works	Места работы пользователей (по периодам)	500
media_types	Типы медиаконтента (справочник)	3
media	Медиаконтент (содержимое в файлах)	500
favorites	Избранное	500
friendship_statuses	статусы дружеских отношений	3
friendship	Друзья	1000
communities	Группы	20
communities_users	Таблица связи пользователей и групп	1000
messages	Сообщения	1000
messages_arh	Сообщения (архивные)	0
posts	Посты	500
posts_media	связь Поста с медиаконтентом	1000
posts_questions	связь Поста с опросом	100
questions_answers	Варианты ответов на опросы	300
likes	Таблица лайков	1000
logs	логгирование добавления/изменения/удаления записей в таблицы БД (*ENGINE=ARCHIVE*)	0

2.3. Триггеры, хранимые процедуры и функции.

Наименование	Примечание
Триггеры	
post_ins_check BEFORE INSERT ON post	для отслеживания того, что в группе может быть только один закрепленный пост (is_fix) в активном статусе
post_upd_check BEFORE UPDATE ON post	
likes_validation BEFORE INSERT ON likes	для проверки на допустимость, отвечающий за добавление записей в таблицу лайков… если в таблице (table_name), для которой добавляем Лайк нет строки (row_id), генерим сигнал ошибки для предотвращения вставки
cities_live_insert_check BEFORE INSERT ON cities_live	для отслеживания того, что у пользователя может быть только один город 'city_born' и 'live_now'
cities_live_update_check BEFORE UPDATE ON cities_live	
users_insert AFTER INSERT ON users	В целях логгирования
users_update AFTER UPDATE ON users	
users_delete AFTER DELETE ON users	
profiles_insert AFTER INSERT ON profiles	
profiles_update AFTER UPDATE ON profiles	
profiles_delete AFTER DELETE ON profiles	
passwords_insert AFTER INSERT ON passwords	
passwords_update AFTER UPDATE ON passwords	
passwords_delete AFTER DELETE ON passwords	

Хранимые процедуры и функции	
insert_friendship(IN num_str INT)	Процедура заполнения таблицы friendship. Учитывая составной первичный ключ таблицы, проще заполнить таблицу и игнорируя ошибки дублирования *(для наполнения БД)*
log_insert (IN name_table VARCHAR(20), IN type_operation VARCHAR(10), IN row_id BIGINT, IN description VARCHAR(255))	Процедура записи в лог-таблицу информации по изменению сущностей БД *(вызывается из триггеров)*
get_rating_user (IN user_id INT, OUT user_rating_out FLOAT)	Процедура вычисления рейтинга пользователя на основе построения динамического SQL-запроса с использованием Курсора и обработчика ошибок.
fill_ratings_all ()	Процедура заполнения временной таблицы с рейтингами всех пользователей
messages_move_in_arh()	Процедура переносит "старые" данные из таблицы messages, в предназначенную для архивных сообщений таблицу messages_arh, оставив только 100 самых "свежих" строк. *(транзакция)*
is_row_exists (row_id INT, table_name VARCHAR(40))	Функция получает в качестве параметров имя сущности (таблицы) и ИД строки. Возвращает наличие такой записи в запрошенной таблице *(для поддержания целостности БД, вызывается из триггера таблицы likes)*

Контроль целостности БД при невозможности использования FK (на примере Лайков).

Таблица likes позволяет содержать лайки для различных сущностей (messages, users, media, posts). Естественно, на уровне FK нет возможности поддерживать целостность БД, т.к. поле «row_id» может ссылаться на различные таблицы.

Создадим функцию «is_row_exists», получающую в качестве параметров имя сущности (таблицы) и ИД строки и возвращающую наличие такой записи в запрошенной таблице.

Вызов данной функции производится из триггера, отвечающего за добавление записей в таблицу лайков (*likes_validation BEFORE INSERT ON likes*). Если функция возвращает FALSE, триггер генерит сигнал для предотвращения вставки «*SIGNAL SQLSTATE "45000"*»

Естественно, для полноценного закрытия вопроса целостности, желательно добавить триггеры на всех выше упомянутых таблицах, к примеру, удаляющих каскадно записи из likes, при удалении родительской записи.

2.4. Скрипты характерных выборок (GRUD-операции).

Первоначальное заполнение БД было осуществлено при помощи сервиса http://filldb.info. Естественно, полученные данные весьма далеки от реальных. Чтобы привести «сырые» данные к более достоверному виду, мы воспользовались массой скриптов GRUD-операций (*FaceBook_convert_BD_04.sql*).

Примеры скриптов последующих характерных выборок из БД приведены в «*FaceBook_GRUD_05.sql*».

Агрегация данных (GROUP BY).

- Количество дней рождения, которые приходятся на каждый из дней недели (учесть, что необходимы дни недели текущего года, а не года рождения).
- Определить кто больше поставил лайков (всего) - мужчины или женщины?
- Подсчитать общее количество лайков десяти самым молодым пользователям (сколько лайков получили 10 самых молодых пользователей).
- Найти 10 пользователей, которые проявляют наименьшую активность в использовании социальной сети.

Сложные запросы (JOIN).

- Определить кто больше поставил лайков (всего) - мужчины или женщины?
- Подсчитать общее количество лайков десяти самым молодым пользователям (сколько лайков получили 10 самых молодых пользователей).
- Найти 10 пользователей, которые проявляют наименьшую активность в использовании социальной сети.
- Вывести все дни текущего месяца. Если есть именинник (даты в profiles) - указать его ФИО (users). Если именинников нет - фраза "именинников нет".

Оконные функции.

Запрос выводит следующие столбцы:
- имя группы;
- среднее количество пользователей в группах;
- самый молодой пользователь в группе;
- самый старший пользователь в группе;
- общее количество пользователей в группе;
- всего пользователей в системе;
- отношение в процентах (общее количество пользователей в группе / всего пользователей в системе) * 100

2.5. Транзакции.

Учитывая высокую продуктивность пользователей при написании сообщений, наступает момент, когда становится целесообразным «почистить» таблицу messages. Для реализации функционала служит процедура «messages_move_in_arh()». Она переносит "старые" данные из таблицы messages, в предназначенную для архивных сообщений таблицу messages_arh, оставив только 100 самых "свежих" строк. Принимая во внимание возможную высокую нагруженность системы, блокировки и т.д., процедура использует встроенный механизм поддержания транзакций.

2.6. Администрирование.

Создадим нескольких пользователей и предоставим различные виды доступа к БД:

- user1_read - доступны только запросы на чтение данных
- user2_all - любые операции в пределах базы данных
- user3_read_view - нет доступа к таблице passwords, однако, может извлекать записи из представления users_passwords_view.

2.7. Логгирование.

Для хранения результатов логгирования используется файл «logs» (*ENGINE=ARCHIVE*).

В целях демонстрации подхода к логгированию выбран «простейший» вариант с записью в БД всех действий, с таблицами (users, profiles, passwords). Фиксируется пользователь, произведший изменения и дата-время. Естественно, пароли в ЛОГ-файле записываются с маскирующей маской. При изменении полей в таблице users, указываются «старые» и «новые» значения…

Для включения дополнительно отслеживаемых таблиц, достаточно на них включить соответствующие триггеры с вызовом процедуры «*log_insert*» .

В «серьезных» проектах, к информации относятся как в GIT-е, т.е. ничто не удаляется и процесс изменения полностью логгируется и/или у строк ставится признак активности «на дату», выделяются «архивные» таблицы для редко-используемых или устаревших данных …

2.8. Динамические запросы, Курсоры и обработчик ошибок (на примере расчета Рейтинга).

Определим критерии активности пользователя используя сущность «ratings». В таблице определим весовые коэффициенты для каждого вида активности. Можно указывать любую таблицу, для которой указываем наименование поля со значением ID пользователя и вес (коэффициент значимости активности в данной сущности). Так, к примеру, таблице «friendship» мы поставили два разных веса оценки: «1» - для инициатора дружбы и «0.5» - для «получателя» приглашения дружить.

Для вычисления рейтинга пользователей, применяется хранимая процедура «get_rating_user», использующая Курсор и построение динамического SQL-запроса. Процедура просматривает все оцениваемые сущности, указанные в таблице ratings (такие, как posts, media, friendship,...). В общем, любые таблицы по любому полю связи с user_id и произвольному весовому коэффициенту (разные активности оцениваются индивидуально). На входе – «user_id», на выходе итоговый рейтинг пользователя с учетом весовых коэффициентов.

Также, используется процедура «*fill_ratings_all*» заполнения временной таблицы с рейтингами всех пользователей. Процедура последовательно перебирает всех пользователей и рассчитывает по каждому его рейтинг путем запуска процедуры «*get_rating_user*». Результат записывается во временную таблицу «*ratings_tmp*».

2.9. Индексы.

При принятии решения о целесообразности добавления индекса, в условиях невозможности применения классического метода учитывающего статистику частоты запросов к БД и накладные расходы по временным задержкам психологически «мешающим» usability, обратим внимание на субъективно часто-использующийся функционал. При этом, не будем строить индексы, дублирующие уже имеющиеся (все первичные, вторичные ключи и поля с признаком UNIQUE).

Проанализировав интерфейс «facebook» и отталкиваясь от имеющегося описания БД, приходит на ум целесообразность в обеспечении следующих таблиц индексами:

- users (last_name, first_name) – поиск пользователя
- profiles (birthday) – частая опция поиска
- cities (name) – поиск города
- countries (name) – поиск страны
- posts (title) - просмотр/поиск постов по заголовкам
- media – не нуждается (по внешним ключам и так есть индексы, а в остальном - слабо структурированная информация)
- media_types – не нуждается (минимум записей)
- friendship – не нуждается (внешние ключи и так закрывают все потребности по индексам)
- friendship_statuses - не нуждается (минимум записей)
- communities (name) - поиск на уникальность, но такой индекс уже есть, как у всех полей с признаком UNIQUE
- communities_users - не нуждается (внешние ключи…)
- likes – не нуждается (таблица связей)
- target_types - не нуждается (минимум записей)
- ratings - не нуждается (минимум записей)
- messages – не нуждается

3. Состав дистрибутива.

- MySQL CourseWork (read_me).docx – этот файл
- FaceBook_tables_01.sql – скрипт создания таблиц БД и представлений
- FaceBook_index_trigger_func_proc_02.sql – скрипт добавления индексов, триггеров, хранимых процедур и функций
- FaceBook_dump_03.sql – дамп «сырых» данных БД (не включает описания таблиц)
- FaceBook_convert_BD_04.sql – скрипт конвертации данных для придания более натурального вида
- FaceBook_dump_01_04.sql – дамп БД, включающий все предыдущие этапы работы с данными
- FaceBook_GRUD_05.sql – скрипт примеров вызова функций, процедур, запросов и т.д.