Bases de datos 1

ÍNDICES

Bases de datos 1

- **Índices: existen básicamente dos tipos de ítems**
  - Ordenados:
    - estos índices se basan en un ordenamiento de los valores.
  - Basados en hash:
    - se basan en la distribución uniforme de los valores en un rango de “espacios” o “buckets”.
    - El espacio asignado a cada valor es determinado por una función, denominada “función de hash”.
  - Independientemente del tipo de índice, hay que recordar
    - Hace más lenta la escritura (más trabajo por hacer).
    - Más espacio utilizado.

Bases de datos 1

- **Índices:**
  - No existe una técnica de indización que siempre resulte mejor que otra. Se deben evaluar aspectos tales como:
    - Tipo de acceso: se suelen buscar registros con un valor específico o que esté dentro de un rango dado?
    - Tiempo de acceso: cuanto se tarda en encontrar uno o más registros.
    - Tiempo de inserción: se considera tanto el tiempo que lleva encontrar el lugar correcto donde insertar el registro, como el tiempo que lleva actualizar el índice.
    - Tiempo de borrado: se considera el tiempo que lleva encontrar el ítem a eliminar, así como el tiempo que lleva actualizar el índice.
    - Espacio extra: cuanto espacio extra ocupa la estructura de un índice.

Bases de datos 1

- **Índices:**
  - Comparación entre índices ordenados y hash.

  - Si las consultas son del tipo

    select $A_1, A_2, \ldots, A_n$

    from r

    where $A_i = c$

    Entonces los índices basados en funciones de hash son los indicados.
  - Si las consultas son del tipo

    select $A_1, A_2, \ldots, A_n$

    from r

    where $A_i <= c_2$ **and** $A_i >= c_1$

    Entonces los índices ordenados tendrán más performance.

Bases de datos 1

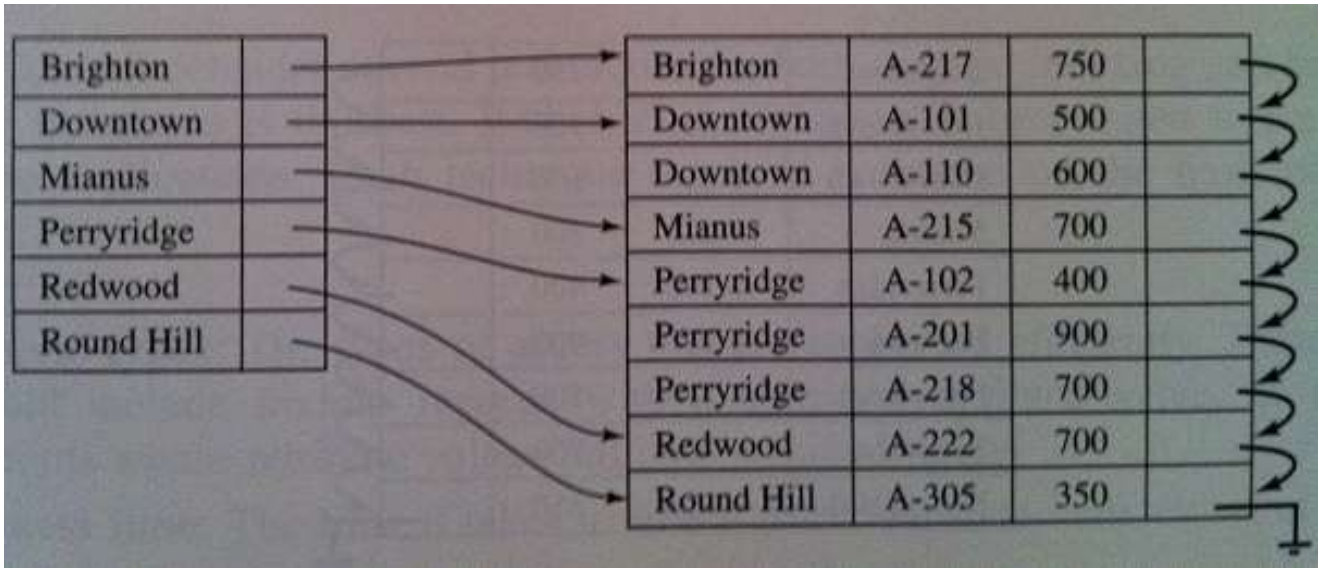
- **Scan vs. Seek**
  - El optimizador de consultas debe decidir entre usar un scan o un seek, dependiendo:
    - El costo relativo de cada uno (función interna)
    - El número de registros que estima se encontrarán
      - Si es de aproximadamente el 30% del total de los registros, entonces el optimizador utilizará probablemente un scan.
    - Ventajas del comando ANALYZE TABLE.

Bases de datos 1

- **Índices ordenados: nuevamente podemos tener dos variedades:**
  - Densos:
    - Cada uno de los valores de búsqueda de los registros es representado en el índice.
    - Usualmente más rápidos.
  - Dispersos:
    - Se representan solamente algunos de los valores de búsqueda.
    - Requieren menos espacio y tienen menos mantenimiento a causa de las inserciones y borrados.

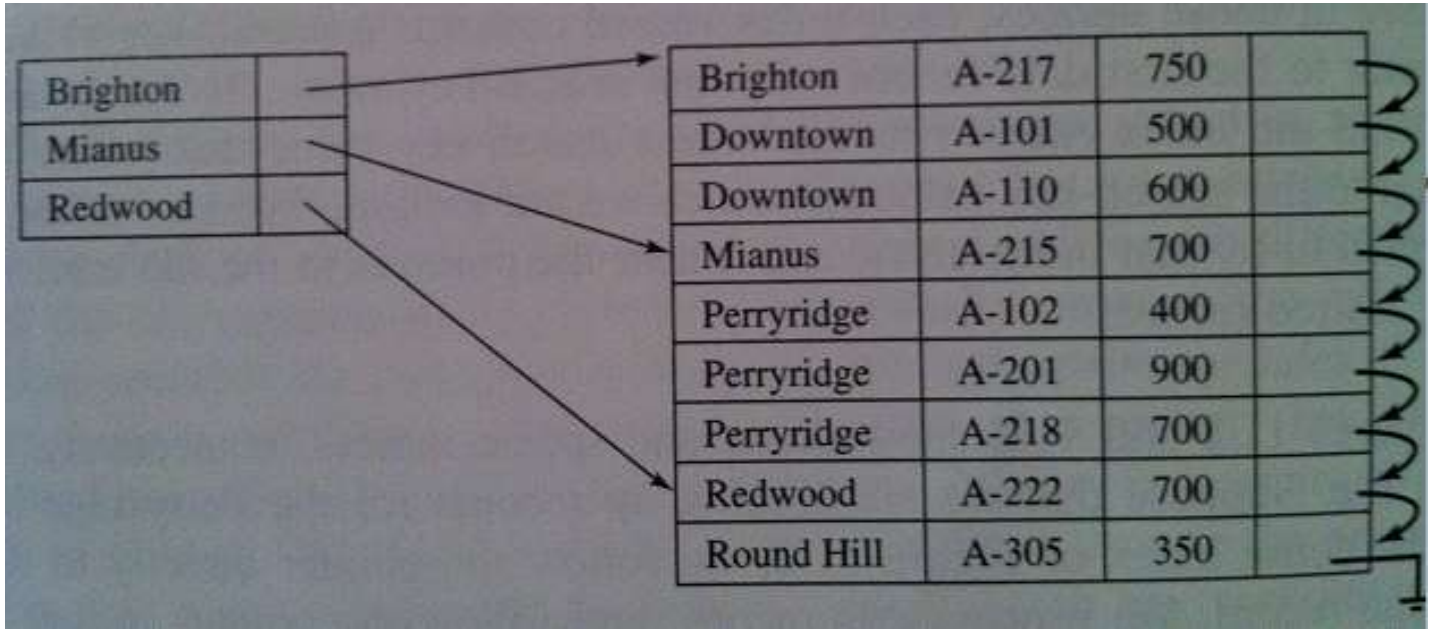
Bases de datos 1

- Índice denso



Bases de datos 1

- Índice disperso



Bases de datos 1

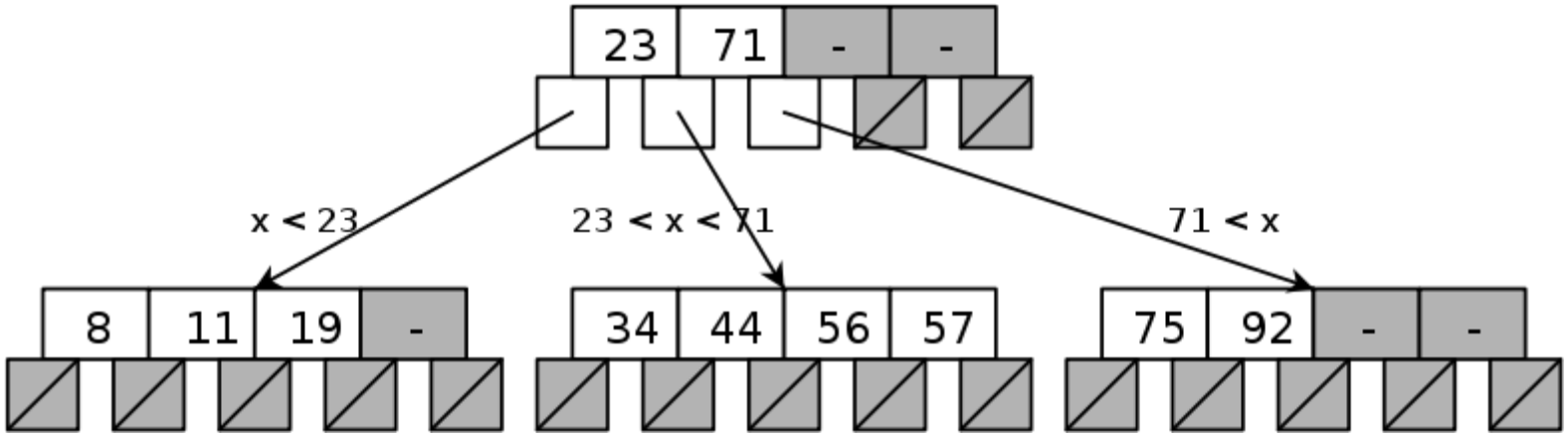
- **Índices en MySQL**
  - Todos los tipos de datos de MySQL puede ser indizados.
  - La utilización de índices en las columnas más relevantes representa la operación de optimización más importante que se puede realizar.
  - Cada storage engine define la cantidad máxima de índices y su tamaño.
    - Cada storage engine soporta al menos 16 índices por tabla.

Bases de datos 1

- **MySQL utiliza diferentes tipos de índices dependiendo de las tablas y su formato**
  - B-tree
    - Primary key
    - Unique
    - Index
    - Fulltext (utilizado para realizar búsquedas de texto en columnas de tipo CHAR, VARCHAR y TEXT).
      - Solamente para MyISAM.
  - R-tree
    - Datos espaciales
  - Hash
    - Memory engine por defecto.

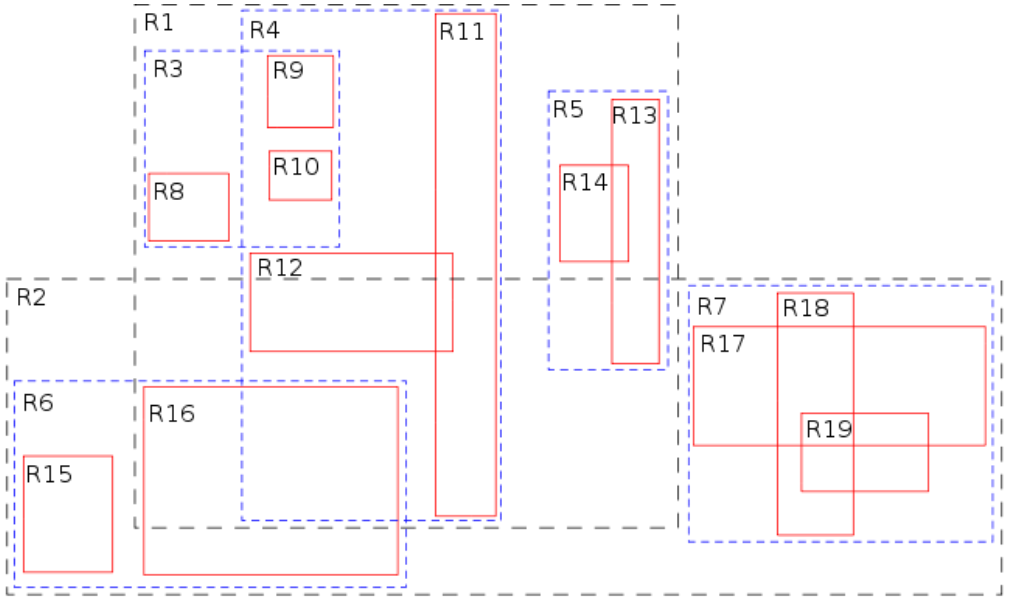
Bases de datos 1

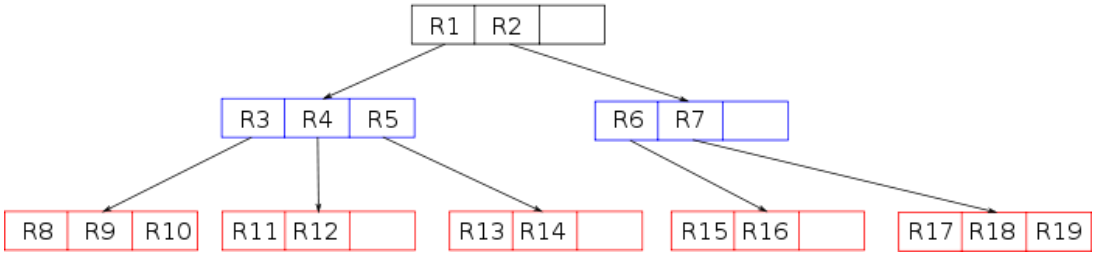
- Ejemplo de un árbol B



Bases de datos 1

- Ejemplo de un R-tree





Bases de datos 1

- **Para qué utiliza MySQL los índices?**
  - Para encontrar rápidamente las filas que concuerdan con una cláusula where.
  - Para no tomar en consideración algunas filas (si hay varios índices disponibles, se utiliza el que menos filas retorna).
  - Para recuperar filas de una tabla al ejecutar sentencias “join”.
    - Para este tipo de comparación se sugiere utilizar los mismos tipos en ambas columnas (ejemplo de numéricos y strings).
  - Para recuperar el Min() y Max().
  - Para realizar un ordenamiento (order by).
  - En algunos casos, se pueden realizar optimizaciones con las columnas numéricas para utilizar los valores de los índices en vez de retornar los de las tablas.

Bases de datos 1

- **Índices basados en árboles B en MySQL**
  - Un índice de este tipo se puede utilizar cuando las consultas utilizan expresiones =, >, >=, <, <=, o BETWEEN.
  - Este tipo de índice también se puede utilizar en comparaciones con LIKE contra una constante de tipo string que no comience con un “wildcard”.

```
SELECT * FROM tbl_name WHERE key_col LIKE 'Patrick%';

SELECT * FROM tbl_name WHERE key_col LIKE 'Pat%_ck%';

SELECT * FROM tbl_name WHERE key_col LIKE '%Patrick%';

SELECT * FROM tbl_name WHERE key_col LIKE other_col;
```

Bases de datos 1

- **Índices hash en MySQL**
  - Se los utiliza para comparaciones por igualdad (=).
  - El optimizador no los puede utilizar para acelerar las cláusulas order by (ya que no existe un orden establecido por la función de hashing).
  - El optimizador no tiene forma de calcular la cantidad de registros entre dos valores y por lo tanto no puede decidir qúe índice usar. Esto puede afectar el rendimiento negativamente al pasar de MyISAM a Memory.
  - El índices basados en B-tree pueden utilizarse en forma parcial, mientras que los basados en hash requieren la entrada completa de la clave.

Bases de datos 1

- **Creación de índices en MySQL**

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
    [index_type]
    ON tbl_name (index_col_name,...)
    [index_type]

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH}
```

  - Esta sentencia permite agregar índices con posterioridad a la creación de la tabla.
  - Se mapea a una sentencia ALTER TABLE….
  - No permite crear un índice Primary Key (utilizar *alter*).

Bases de datos 1

- Storage engines e índices

Storage Engine	Permissible Index Types
MyISAM	BTREE
InnoDB	BTREE
MEMORY/HEAP	HASH, BTREE
NDB	HASH, BTREE (see note in text)

Bases de datos 1

- **Algunas características adicionales de los índices**
  - Es posible crear índices que utilicen solamente una parte de una columna:
    - col_name(length)
    - Ejemplo CREATE INDEX part_of_name ON customer (name(10));
    - Si los nombres suelen diferir en los 10 primeros elementos, entonces este índice será muy rápido y permitirá:
      - Ahorrar espacio
      - Mejorar el tiempo en las inserciones
  - Un índice UNIQUE crea una restricción para que ningún otro valor igual se pueda insertar en dicha columna. Si se intenta, MySQL devuelve un error.

Bases de datos 1

- **Como identificar campos candidatos a ser indizados?**
  - Prestar atención a los filtros.
  - Claves foráneas (usualmente la clave primaria está indizada, pero por las dudas verificar).
  - Cláusulas GROUP BY. El orden de los campos es importante.
  - Eliminar índices cubiertos por otros índices.

Bases de datos 1

- **Otras opciones para optimizar el rendimiento**
  - Utilizar el tipo correcto para las columnas.
    - Principio fundamental: cuanto más pequeño el tipo de dato, más datos entran en un bloque del índice. Cuando más registros entran en cada bloque, menos lecturas se requieren.
  - No utilizar el COUNT(*). Son más rápidas las tablas de resumen.
  - Al requerir consultas, por defecto utilizar el tipo MyISAM.
  - Utilizar el comando SHOW STATUS y SLOW QUERY.
  - Utilizar los mismos tipos de datos en los JOINS.
  - Evitar NOT IN o <>.
  - Utilizar SELECT SQL_CACHE.
  - Utilizar el LIMIT.

Bases de datos 1

CONSULTAS EN MYSQL

Bases de datos 1

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [PROCEDURE procedure_name(argument_list)]
    [INTO OUTFILE 'file_name' export_options
      | INTO DUMPFILE 'file_name'
      | INTO var_name [, var_name]]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

Bases de datos 1

- **Características de la cláusula select**
  - Las opciones ALL y DISTINCT trabajan especificando si deberían devolverse todos los registros, o si deberían eliminarse los repetidos.
    - DISTINCTROW es un sinónimo de DISTINCT.
  - HIGH_PRIORITY otorga alta prioridad al select, incluso más alta que a una cláusula update.
    - Esta opción solamente debería ser utilizada con consultas muy rápidas y que deben ser llevadas a cabo de una sola vez.
    - Un SELECT HIGH_PRIORITY sobre una tabla bloqueada aún puede ser ejecutada.
    - Esta opción no puede ser utilizada en selects que tienen participación en un UNION.

Bases de datos 1

- **Características de la cláusula select**
  - STRAIGHT_JOIN fuerza al optimizador a vincular las tablas en el mismo orden en el que se las listó en la clásula FROM.
    - STRAIGHT_JOIN no se aplica a tablas consideradas como constantes o de sistema.
  - SQL_BIG_RESULT o SQL_SMALL_RESULT pueden ser utilizadas con GROUP BY o DISTINCT para pasar al optimizador información acerca de la cantidad de datos del resultado.
    - Si se utiliza SQL_BIG_RESULT, MySQL utiliza directamente tablas temporarias basadas en disco y utiliza primero el ordenamiento antes que la tabla temporaria.
    - En cambio, con la otra opción MySQL utiliza tablas temporarias veloces en vez de ordenamientos.

Bases de datos 1

- **Características de la cláusula select**
  - SQL_BUFFER_RESULT fuerza que el resultado sea enviado a una tabla temporaria.
    - Esto permite liberar antes los bloqueos de las tablas.
    - Solamente se puede utilizar con los select de primer nivel.
  - SQL_CALC_FOUND_ROWS solicita a MySQL que calcule la cantidad de filas que tendrá el resultado.
  - SQL_CACHE y SQL_NO_CACHE afectan como se guarda en la caché los resultados de la consulta.

Bases de datos 1

- **Optimización de consultas**
  - 4 pasos básicos
    - Evitar la consulta.
    - “Cachear” la consulta.
    - Simplificar la consulta.
    - Optimizar la consulta.

Bases de datos 1

- **Optimización de consultas**
  - EXPLAIN PLAN
  - ANALYZE TABLE
  - SHOW_INDEX
  - Estimación de la performance de una consulta
    - Generalmente se puede basar en la cantidad de accesos al disco.
      - Para tablas pequeñas usualmente bastará un solo acceso ya que el índice está en la caché.
      - Para tablas grandes se puede aplicar la siguiente fórmula
      - log(*row_count*) / log(*index_block_length* / 3 * 2 / (*index_length* + *data_pointer_length*)) + 1

Bases de datos 1

- **Optimización de consultas**
  - Tabla de 500.000 elementos.
  - En MySQL usualmente un bloque de índice ocupa 1024 bytes.
  - Un puntero generalmente utiliza 4 bytes.
  - Entonces, si cada valor del índice pesa 3 bytes por ejemplo se necesitarán
    - log(500,000)/log(1024/3*2/(3+4)) + 1 = 4 accesos
  - En realidad, como él índice pesa aproximadamente 5.2 mb, es probable que esté en memoria por lo que se requerirán menos accesos.

Bases de datos 1

STORED PROCEDURES

Bases de datos 1

- Stored procedures

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body

CREATE
    [DEFINER = { user | CURRENT_USER }]
    FUNCTION sp_name ([func_parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body

proc_parameter:
    [ IN | OUT | INOUT ] param_name type

func_parameter:
    param_name type

type:
    Any valid MySQL data type

characteristic:
    COMMENT 'string'
  | LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }

routine_body:
    Valid SQL routine statement
```

Bases de datos 1

- Ejemplo simple

```
mysql> delimiter //

mysql> CREATE PROCEDURE simpleproc (OUT param1 INT)
    -> BEGIN
    ->   SELECT COUNT(*) INTO param1 FROM t;
    -> END//
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> CALL simpleproc(@a);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @a;
+------+
| @a   |
+------+
| 3    |
+------+
1 row in set (0.00 sec)
```

Bases de datos 1

- **Stored procedures**
  - Razones por las cuales se podría justificar el uso de Stored procedures:
    - Los stored procedures se pueden optimizar y el beneficio puede ser obtenido por todos los clientes.
    - Se cuenta con múltiples aplicaciones que deben consumir la misma lógica de negocios, y cada aplicación está desarrollada con diferente plataforma/lenguaje.
    - El tráfico en la red debe ser minimizado.
    - Existen arquitecturas en las cuales el cliente tiene muy poca capacidad de procesamiento de la información, por lo que ya debería recibir toda la información procesada.
      - Lógica en javascript vs lógica en SP.
    - Existen algunas consultas en SQL que son muy díficiles de realizar sin la ayuda de variables auxiliares e iteradores
      - Actualización de un índice que representa la posición.

Bases de datos 1

- **Stored procedures en MySQL**
  - Requieren la presencia de una tabla de sistema denominada “proc”.
  - MySQL adopta la sintaxis SQL:2003, la cual también es adoptada por DB2.
  - Todavía se sigue implementando el soporte de funciones y procedimientos, no es “completo” todavía.
  - Un procedimiento se invoca mediante “call”.
  - En cambio una función se invoca directamente dentro de una sentencia (retorna una valor escalar).
  - Los procedimientos y las funciones siempre pertenecen a una base de datos, no al servidor.
  - Cuando se elimina una base de datos (drop) también se eliminan los procedimientos y funciones.

Bases de datos 1

- **Stored procedures**
  - Las funciones no pueden ser recursivas.
  - Los procedimientos pueden utilizar recursividad, pero por defecto está desactivado su soporte (max_sp_recursion_depth)
  - Privilegios asociados
    - CREATE ROUTINE
    - ALTER ROUTINE
    - EXECUTE
  - Metadata sobre los SP
    - Tabla ROUTINES en la base de datos INFORMATION_SCHEMA.

Bases de datos 1

- **Stored procedures**
  - LAST_INSERT_ID()
    - Retorna el primer valor generado automáticamente por la última sentencia INSERT sobre una tabla con alguna columna con un valor AUTO_INCREMENT.
    - Si la última sentencia produce un error, el valor es “indefinido”.
  - Stored procedures
    - Si un stored procedure realiza algún cambio que afecta el valor de la función, luego de finalizar el stored procedures las siguientes sentencias verán el cambio realizado.
  - Funciones
    - Si una función realiza un cambio que afecta el valor de esta función, las siguientes sentencias no ven el cambio ya que se lo vuelve al valor original antes de la función.

Bases de datos 1

CURSORES

Bases de datos 1

- **Cursores**
  - Esencialmente son sentencias SQL a las cuales se les ha asignado un nombre.
  - Permiten recuperar los valores de las sentencias y realizar operaciones con ellos.
  - Operaciones sobre cursores
    - DECLARE
    - OPEN
    - FETCH
    - CLOSE
  - En MySQL tienen las siguientes propiedades
    - Read only: no se los puede actualizar.
    - Nonscrollable: se los puede recorrer en una sola dirección y no se puyeden saltear filas.

Bases de datos 1

- Cursores

```
CREATE PROCEDURE curdemo()
BEGIN
  DECLARE done INT DEFAULT 0;
  DECLARE a CHAR(16);
  DECLARE b,c INT;
  DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;
  DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

  OPEN cur1;
  OPEN cur2;

  read_loop: LOOP
    FETCH cur1 INTO a, b;
    FETCH cur2 INTO c;
    IF done THEN
      LEAVE read_loop;
    END IF;
    IF b < c THEN
      INSERT INTO test.t3 VALUES (a,b);
    ELSE
      INSERT INTO test.t3 VALUES (a,c);
    END IF;
  END LOOP;

  CLOSE cur1;
  CLOSE cur2;
END;
```

Bases de datos 1

VIEWS

Bases de datos 1

- **Vistas**
  - Habitualmente el modelo lógico suele ser muy complejo para el usuario.
  - Existen consideraciones de seguridad para que un usuario no acceda al todo el modelo.
  - En ocasiones es mejor darle al usuario una “versión” personalizada del modelo que se ajuste mejor a sus necesidades de consulta.
  - No se debe permitir al usuario realizar operaciones sobre los datos (insert/update/delete).

```
CREATE
    [OR REPLACE]
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    [DEFINER = { user | CURRENT_USER }]
    [SQL SECURITY { DEFINER | INVOKER }]
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

Bases de datos 1

- **Views en MySQL**
  - Una vez creada, la definición de una vista es “congelada”. Esto significa que cambios posteriores a las tablas de la vista no afectarán la vista.
    - Por ejemplo, si luego de crear una vista con SELECT *, si luego se agregan nuevas columnas, éstas no aparecerán en la vista.
  - Las vistas pertenecen a una base de datos, por lo que si se elimina la base, se elimina la vista.
  - Una vista puede ser creada en base a tablas, otras vistas, joins, UNION y subqueries.
  - Los nombres de las columnas deben ser únicos.
  - Para cada columna de la vista se pueden utilizar todos los operadores y funciones disponibles.

Bases de datos 1

- **Restricciones aplicables a las vistas**
  - No se pueden utilizar subqueries en la cláusula FROM.
  - El SELECT no puede referirse a variables del usuario o del sistema.
  - Si se crea dentro de un programa (sp) no se pueden utilizar los parámetros del programa.
  - Cuando se está definiendo la vista, todas las tablas y/o otras vistas a las que se menciona deben existir.
  - No se pueden utilizar tablas temporales ni crear vistas temporales.
  - No se pueden asociar triggers con las vistas.
  - Si se utilizan alias, se verifica que su nombre sea menor a 64 caracteres (el límite real es de 256).
  - Se pueden utilizar ORDER BY, pero se lo ignora si el select viene acompañado de uno propio.

Bases de datos 1

TRIGGERS

Bases de datos 1

- **Triggers**
  - A partir de MySQL 5.0.2 se incorporó el soporte básico para disparadores (triggers).
  - Un disparador es un objeto con nombre dentro de una base de datos el cual se asocia con una tabla y se activa cuando ocurre en ésta un evento en particular.

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
    -> FOR EACH ROW SET @sum = @sum + NEW.amount;
```

Bases de datos 1

- **Triggers**

```
CREATE TRIGGER nombre_disp momento_disp evento_disp
    ON nombre_tabla FOR EACH ROW sentencia_disp
```

  - Nombre_tabla: referencia a una tabla, no puede ser temporal ni una vista.
  - Momento_disp:
    - BEFORE
    - AFTER
  - Evento_disp:
    - INSERT
    - UPDATE
    - DELETE
  - Sentencia_disp: conjunto de sentencias que se desea ejecutar.

```
DROP TRIGGER [nombre_esquema.]nombre_disp
```

Bases de datos 1

CONSTRAINTS

Bases de datos 1

- **Restricciones en MySQL**
  - Normalmente, un error ocurre cuando trata de ejecutar un INSERT o UPDATE en un registro que viole la clave primaria, clave única o clave foránea.
  - Si se usa un motor transaccional como InnoDB, MySQL automáticamente deshace el comando.
  - Si se usa un motor no transaccional, MySQL para de procesar el comando en el registro en el que ocurre el error y deja sin procesar el resto de registros.

Bases de datos 1

- **Antes de la versión 5.0.2 se permitía insertar datos inválidos tratando de convertirlos en datos válidos**
  - Si inserta un valor "incorrecto" en una columna, como NULL en una columna NOT NULL o una valor numérico demasiado grande en una columna numérica, MySQL cambia el valor al "mejor valor posible" para la columna en lugar de producir un error:
    - Si trata de almacenar un valor fuera de rango en una columna numérica, MySQL Server en su lugar almacena cero, el menor valor posible, o el mayor valor posible en la columna.
    - Para cadenas de caracteres, MySQL almacena una cadena vacía o tanto de la cadena de carácteras como quepa en la columna.

Bases de datos 1

- **Restricciones**
  - Si trata de almacenar una cadena de caracteres que no empiece con un número en una columna numérica, MySQL Server almacena 0.
  - MySQL le permite almacenar ciertos valores incorrectos en columnas DATE y DATETIME (tales como '2000-02-31' o '2000-02-00').
  - Si intenta almacenar NULL en una columna que no admita valores NULL ocurre un error para los comandos INSERT de un solo registro.
  - Si un comando INSERT no especifica un valor para una columna, MySQL inserta su valor por defecto si la columna especifica un valor mediante la cláusula DEFAULT. Si la definición no tiene tal cláusula DEFAULT clause, MySQL inserta el valor por defecto implícito para el tipo de datos de la columna.

Bases de datos 1

- **Restricciones**

```
[CONSTRAINT símbolo] FOREIGN KEY [id] (nombre_índice, ...)
    REFERENCES nombre_de_tabla (nombre_índice, ...)
    [ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
    [ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
```

- Ambas tablas deben ser InnoDB y no deben ser tablas temporales.
- En la tabla que hace referencia, debe haber un índice donde las columnas de clave extranjera estén listadas en *primer* lugar, en el mismo orden.
- En la tabla referenciada, debe haber un índice donde las columnas referenciadas se listen en *primer* lugar, en el mismo orden. En MySQL/InnoDB 5.0, tal índice se creará automáticamente en la tabla referenciada si no existe aún.

Bases de datos 1

- **Restricciones**
  - Cuando el usuario intenta borrar o actualizar una fila de una tabla padre
    - CASCADE: Borra o actualiza el registro en la tabla padre y automáticamente borra o actualiza los registros coincidentes en la tabla hija.
    - SET NULL: Borra o actualiza el registro en la tabla padre y establece en NULL la o las columnas de clave foránea en la tabla hija.
    - NO ACTION: En el estándar ANSI SQL-92, NO ACTION significa *ninguna acción* en el sentido de que unintento de borrar o actualizar un valor de clave primaria no sera permitido si en la tabla referenciada hay una valor de clave foránea relacionado.
    - RESTRICT: Rechaza la operación de eliminación o actualización en la tabla padre.
    - SET DEFAULT: Esta acción es reconocida por el procesador de sentencias (parser), pero InnoDB rechaza definiciones de tablas que contengan ON DELETE SET DEFAULT u ON UPDATE SET DEFAULT.