Aufgabe 1

- Vereinigung ($\cup$)
  - hashbasiert

```
1 Function: union(R1, R2)
2 build hash table H;
3 foreach record r ∈ R1 do
4     if mismatch on H via h(r) then
5         append r to Rres;
6 foreach record r ∈ R2 do
7     if mismatch on H via h(r) then
8         append r to Rres;
9 return Rres;
```

  - sortierungsbasiert

```
1  Function: union(R1, R2)
2  foreach record r ∈ R1 do
3      append r to Rres;
4  foreach record r ∈ R2 do
5      append r to Rres;
6  foreach record ri ∈ Rres do
7      if ri+1 exists then
8          if ri+1 equals ri then
9              Rres ← Rres \ ri+1
10 return Rres;
```

- Vereinigung ($\cap$)
  - hashbasiert

```
1  Function: union(R1, R2)
2  build hash table H;
3  foreach record r ∈ R1 do
4      if mismatch on H via h(r) then
5          append r to Rres;
6  foreach record r ∈ R2 do
7      if match on H via h(r) then
8          append r to Rres;
9  return Rres;
```

  - sortierungsbasiert

```
1  Function: union(R1, R2)
2  foreach record r ∈ R1 do
3      append r to R1,2;
4  foreach record ri ∈ R1,2 do
5      append r to R1,2;
6  foreach record ri ∈ R1,2 do
7      if ri+1 exists then
8          if ri+1 equals ri then
               // res|Rres| is the least entry of Rres
9              if res|Rres| ≠ ri+1 then
10                 Rres ← R1,2 \ ri
11 return Rres;
```

Aufgabe 2

- block nested loop join

```
1 Function: block_nljoin(R, S, p)
2 foreach b_R-sized block in R do
3     foreach b_S-sized block in S do
          /* performed in the buffer                                    */
          /* find matches in current R- and S-blocks and append them to
             the result                                                 */
4         foreach record r_{b_R} ∈ b_R do
5             foreach record r_{b_S} ∈ b_S do
6                 if ⟨r_{b_R}, r_{b_S}⟩ matches p then
7                     append ⟨r_{b_R}, r_{b_S}⟩ to the result and mark as matched
8             if is not marked as copied then
9                 append ⟨r_{b_R}, −⟩ to the result
```

Tritt das Prädikat auf mindestens ein Tupel zusammengefügt aus beiden Relationen zu, wird dieses der Zieltabelle angehängt. Sollte sich kein solches Tupel für ein Tupel der Relation R finden, wird r_{b_R} ohne Partner der Zieltabelle angefügt, da es unmarkiert blieb.

- index nested loop join

```
1 Function: index_nljoin(R, S, p)
2 foreach record r ∈ R do
      /* scan S-index using (key value in) r and concatenate r with all
         matching tuples s                                              */
3     if ⟨r, IndexScan(r, S)⟩ matches p then
4         foreach s ∈ IndexScan(r, S) do
5             append ⟨r, s⟩ to result
6         else append ⟨r, −⟩ to result;
```

Sofern der IndexScan mindestens ein Element liefert, finden sich für r Partner. Andernfalls gibt es kein Tupel $\langle r, s\rangle$, $\langle r, -\rangle$ wird dem Ergebnis angefügt.

Aufgabe 3

Aufgabe 4

1. (a)

```
1 Function: open()
2 R.open();
3 S.open();
```

```
1 Function: close()
2 R.close();
3 S.close();
```

```
1 Function: next()
2 if (r ← R.next()) ≠ ⟨EOF⟩ then
3     return r;
4     else if (s ← S.next()) ≠ ⟨EOF⟩ then
5         return s;
6     else return ⟨EOF⟩;
7     ;
```

(b)

```
1  Function: open()
2  R.open();
3  S.open();

1  Function: close()
2  R.close();
3  S.close();

1  Function: next()
2  if ((r ← R.next()) ≠ ⟨EOF⟩) & r mismatches H then
3      add r via h(r) to H;
4      return r;
5      else if ((s ← S.next()) ≠ ⟨EOF⟩) & s mismatches H then
6          add s via h(r) to H;
7          return s;
8      else if ((r ← R.next()) ≠ ⟨EOF⟩) & (s ← S.next()) ≠ ⟨EOF⟩)) then
9          return next();
10     ;
11     else return ⟨EOF⟩;
12     ;
```

(c)

```
1 Function: open()
2 R.open();
3 S.open();
```

```
1 Function: close()
2 R.close();
3 S.close();
```

```
1 Function: next()
2 while (r ← R.next()) ≠ ⟨EOF⟩ do
3 |   sum ← sum + r
4 while (s ← S.next()) ≠ ⟨EOF⟩ do
5 |   sum ← sum + s
6 return ⟨EOF⟩;
```

2. `UNION ALL` und `UNION` sind nicht blockende Operatoren, mit `next()` wird nur ein einziges Tupel geliefert.
   `SUM` hingegen muss alle Tupel durchlaufen, da sonst unvollständig, er ist ein blockender Operator.