

When To Use

- To display a collection of structured data.
- To sort, search, paginate, filter data.

How To Use

Specify `dataSource` of Table as an array of data.

```
const dataSource = [
  {
    key: '1',
    name: 'Mike',
    age: 32,
    address: '10 Downing Street',
  },
  {
    key: '2',
    name: 'John',
    age: 42,
    address: '10 Downing Street',
  },
];

const columns = [
  {
    title: 'Name',
    dataIndex: 'name',
    key: 'name',
  },
  {
    title: 'Age',
    dataIndex: 'age',
    key: 'age',
  },
  {
    title: 'Address',
    dataIndex: 'address',
    key: 'address',
  },
];

<Table dataSource={dataSource} columns={columns} />;
```

Promotion

- [Kitchen Sketch Plugin](#) 💎

- [ProTable - Advanced Tables](#)
- [S2 - Analytical Tables](#)

Examples

Basic Usage

```
import React from 'react';
import { Space, Table, Tag } from 'antd';
import type { TableProps } from 'antd';

interface DataType {
  key: string;
  name: string;
  age: number;
  address: string;
  tags: string[];
}

const columns: TableProps<DataType>['columns'] = [
  {
    title: 'Name',
    dataIndex: 'name',
    key: 'name',
    render: (text) => <a>{text}</a>,
  },
  {
    title: 'Age',
    dataIndex: 'age',
    key: 'age',
  },
  {
    title: 'Address',
    dataIndex: 'address',
    key: 'address',
  },
  {
    title: 'Tags',
    key: 'tags',
    dataIndex: 'tags',
    render: (_, { tags }) => (
      <>
        {tags.map((tag) => {
          let color = tag.length > 5 ? 'geekblue' : 'green';
          if (tag === 'loser') {
            color = 'volcano';
          }
          <span style={color}>{tag}</span>
        })}
      </>
    )
  }
];
```

```

        }
        return (
            <Tag color={color} key={tag}>
                {tag.toUpperCase()}
            </Tag>
        );
    })}
</>
),
},
{
    title: 'Action',
    key: 'action',
    render: (_, record) => (
        <Space size="middle">
            <a>Invite {record.name}</a>
            <a>Delete</a>
        </Space>
    ),
},
];

const data: DataType[] = [
    {
        key: '1',
        name: 'John Brown',
        age: 32,
        address: 'New York No. 1 Lake Park',
        tags: ['nice', 'developer'],
    },
    {
        key: '2',
        name: 'Jim Green',
        age: 42,
        address: 'London No. 1 Lake Park',
        tags: ['loser'],
    },
    {
        key: '3',
        name: 'Joe Black',
        age: 32,
        address: 'Sydney No. 1 Lake Park',
        tags: ['cool', 'teacher'],
    },
];

```

```
const App: React.FC = () => <Table<DataType> columns={columns} dataSource=
{data} />;

export default App;
```

JSX style API

```
import React from 'react';
import { Space, Table, Tag } from 'antd';

const { Column, ColumnGroup } = Table;

interface DataType {
  key: React.Key;
  firstName: string;
  lastName: string;
  age: number;
  address: string;
  tags: string[];
}

const data: DataType[] = [
  {
    key: '1',
    firstName: 'John',
    lastName: 'Brown',
    age: 32,
    address: 'New York No. 1 Lake Park',
    tags: ['nice', 'developer'],
  },
  {
    key: '2',
    firstName: 'Jim',
    lastName: 'Green',
    age: 42,
    address: 'London No. 1 Lake Park',
    tags: ['loser'],
  },
  {
    key: '3',
    firstName: 'Joe',
    lastName: 'Black',
    age: 32,
    address: 'Sydney No. 1 Lake Park',
    tags: ['cool', 'teacher'],
  },
]
```

```

];

const App: React.FC = () => (
  <Table<DataType> dataSource={data}>
    <ColumnGroup title="Name">
      <Column title="First Name" dataIndex="firstName" key="firstName" />
      <Column title="Last Name" dataIndex="lastName" key="lastName" />
    </ColumnGroup>
    <Column title="Age" dataIndex="age" key="age" />
    <Column title="Address" dataIndex="address" key="address" />
    <Column
      title="Tags"
      dataIndex="tags"
      key="tags"
      render={(tags: string[]) => (
        <>
          {tags.map((tag) => {
            let color = tag.length > 5 ? 'geekblue' : 'green';
            if (tag === 'loser') {
              color = 'volcano';
            }
            return (
              <Tag color={color} key={tag}>
                {tag.toUpperCase()}
              </Tag>
            );
          })}
        </>
      )}
    />
    <Column
      title="Action"
      key="action"
      render={(_: any, record: DataType) => (
        <Space size="middle">
          <a>Invite {record.lastName}</a>
          <a>Delete</a>
        </Space>
      )}
    />
  </Table>
);

export default App;

```

selection

```
import React, { useState } from 'react';
import { Divider, Radio, Table } from 'antd';
import type { TableColumnsType, TableProps } from 'antd';

interface DataType {
  key: React.Key;
  name: string;
  age: number;
  address: string;
}

const columns: TableColumnsType<DataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
    render: (text: string) => <a>{text}</a>,
  },
  {
    title: 'Age',
    dataIndex: 'age',
  },
  {
    title: 'Address',
    dataIndex: 'address',
  },
];

const data: DataType[] = [
  {
    key: '1',
    name: 'John Brown',
    age: 32,
    address: 'New York No. 1 Lake Park',
  },
  {
    key: '2',
    name: 'Jim Green',
    age: 42,
    address: 'London No. 1 Lake Park',
  },
  {
    key: '3',
    name: 'Joe Black',
    age: 32,
    address: 'Sydney No. 1 Lake Park',
  },
];
```

```

    {
      key: '4',
      name: 'Disabled User',
      age: 99,
      address: 'Sydney No. 1 Lake Park',
    },
  ],
];

// rowSelection object indicates the need for row selection
const rowSelection: TableProps<DataType>['rowSelection'] = {
  onChange: (selectedRowKeys: React.Key[], selectedRows: DataType[]) => {
    console.log(`selectedRowKeys: ${selectedRowKeys}`, 'selectedRows: ',
selectedRows);
  },
  getCheckboxProps: (record: DataType) => ({
    disabled: record.name === 'Disabled User', // Column configuration not
to be checked
    name: record.name,
  }),
};

const App: React.FC = () => {
  const [selectionType, setSelectionType] = useState<'checkbox' | 'radio'>
('checkbox');

  return (
    <div>
      <Radio.Group onChange={(e) => setSelectionType(e.target.value)}
value={selectionType}>
        <Radio value="checkbox">Checkbox</Radio>
        <Radio value="radio">radio</Radio>
      </Radio.Group>
      <Divider />
      <Table<DataType>
        rowSelection={{ type: selectionType, ...rowSelection }}
        columns={columns}
        dataSource={data}
      />
    </div>
  );
};

export default App;

```

Selection and operation

```

import React, { useState } from 'react';
import { Button, Flex, Table } from 'antd';
import type { TableColumnsType, TableProps } from 'antd';

type TableRowSelection<T extends object = object> = TableProps<T>
['rowSelection'];

interface DataType {
  key: React.Key;
  name: string;
  age: number;
  address: string;
}

const columns: TableColumnsType<DataType> = [
  { title: 'Name', dataIndex: 'name' },
  { title: 'Age', dataIndex: 'age' },
  { title: 'Address', dataIndex: 'address' },
];

const dataSource = Array.from<DataType>({ length: 46 }).map<DataType>((_,
i) => ({
  key: i,
  name: `Edward King ${i}`,
  age: 32,
  address: `London, Park Lane no. ${i}`,
}));

const App: React.FC = () => {
  const [selectedRowKeys, setSelectedRowKeys] = useState<React.Key[]>([]);
  const [loading, setLoading] = useState(false);

  const start = () => {
    setLoading(true);
    // ajax request after empty completing
    setTimeout(() => {
      setSelectedRowKeys([]);
      setLoading(false);
    }, 1000);
  };

  const onSelectChange = (newSelectedRowKeys: React.Key[]) => {
    console.log('selectedRowKeys changed: ', newSelectedRowKeys);
    setSelectedRowKeys(newSelectedRowKeys);
  };

```



```

const rowSelection: TableRowSelection<DataType> = {
  selectedRowKeys,
  onChange: onSelectChange,
};

const hasSelected = selectedRowKeys.length > 0;

return (
  <Flex gap="middle" vertical>
    <Flex align="center" gap="middle">
      <Button type="primary" onClick={start} disabled={!hasSelected}
loading={loading}>
        Reload
      </Button>
      {hasSelected ? `Selected ${selectedRowKeys.length} items` : null}
    </Flex>
    <Table<DataType> rowSelection={rowSelection} columns={columns}
dataSource={dataSource} />
  </Flex>
);
};

export default App;

```

Custom selection

```

import React, { useState } from 'react';
import { Table } from 'antd';
import type { TableColumnsType, TableProps } from 'antd';

type TableRowSelection<T extends object = object> = TableProps<T>
['rowSelection'];

interface DataType {
  key: React.Key;
  name: string;
  age: number;
  address: string;
}

const columns: TableColumnsType<DataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
  },

```

```

    {
      title: 'Age',
      dataIndex: 'age',
    },
    {
      title: 'Address',
      dataIndex: 'address',
    },
  ],
];

const dataSource = Array.from({ length: 46 }).map<DataType>((_, i) => ({
  key: i,
  name: `Edward King ${i}`,
  age: 32,
  address: `London, Park Lane no. ${i}`,
}));

const App: React.FC = () => {
  const [selectedRowKeys, setSelectedRowKeys] = useState<React.Key[]>([]);

  const onSelectChange = (newSelectedRowKeys: React.Key[]) => {
    console.log('selectedRowKeys changed: ', newSelectedRowKeys);
    setSelectedRowKeys(newSelectedRowKeys);
  };

  const rowSelection: TableRowSelection<DataType> = {
    selectedRowKeys,
    onChange: onSelectChange,
    selections: [
      Table.SELECTION_ALL,
      Table.SELECTION_INVERT,
      Table.SELECTION_NONE,
      {
        key: 'odd',
        text: 'Select Odd Row',
        onSelect: (changeableRowKeys) => {
          let newSelectedRowKeys = [];
          newSelectedRowKeys = changeableRowKeys.filter((_, index) => {
            if (index % 2 !== 0) {
              return false;
            }
            return true;
          });
          setSelectedRowKeys(newSelectedRowKeys);
        },
      },
    ],
  },
};

```

```

    {
      key: 'even',
      text: 'Select Even Row',
      onSelect: (changeableRowKeys) => {
        let newSelectedRowKeys = [];
        newSelectedRowKeys = changeableRowKeys.filter((_, index) => {
          if (index % 2 !== 0) {
            return true;
          }
          return false;
        });
        setSelectedRowKeys(newSelectedRowKeys);
      },
    },
  ],
};

return <Table<DataType> rowSelection={rowSelection} columns={columns}
dataSource={dataSource} />;
};

export default App;

```

Selection Perf

Debug

```

import React, { useState } from 'react';
import { InputNumber, Table } from 'antd';
import type { TableColumnsType, TableProps } from 'antd';

type TableRowSelection<T extends object = object> = TableProps<T>
['rowSelection'];

const RenderTimes: React.FC = () => {
  const timesRef = React.useRef(0);
  timesRef.current += 1;
  return <span>{timesRef.current}</span>;
};

interface DataType {
  key: React.Key;
  name: string;
  age: number;
  address: string;
}

```

```
const shouldCellUpdate = (record: DataType, prevRecord: DataType) => record
!== prevRecord;
```

```
const columns: TableColumnsType<DataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
    shouldCellUpdate,
  },
  {
    title: 'Age',
    dataIndex: 'age',
    shouldCellUpdate,
  },
  {
    title: 'Address',
    dataIndex: 'address',
    shouldCellUpdate,
    render: (addr) => (
      <>
        {addr}
        <RenderTimes />
      </>
    ),
  },
];
```

```
function genData(length: number) {
  return Array.from({ length }).map<DataType>((_, i) => ({
    key: i,
    name: `Edward King ${i}`,
    age: 32,
    address: `London, Park Lane no. ${i}`,
  }));
}
```

```
const App: React.FC = () => {
  const [data, setData] = useState<DataType[]>(genData(50));
  const [selectedRowKeys, setSelectedRowKeys] = useState<React.Key[]>([]);

  const onSelectChange = (newSelectedRowKeys: React.Key[]) => {
    console.log('selectedRowKeys changed: ', newSelectedRowKeys);
    setSelectedRowKeys(newSelectedRowKeys);
  };
};
```

```

const rowSelection: TableRowSelection<DataType> = {
  selectedRowKeys,
  onChange: onSelectChange,
};

return (
  <>
    <InputNumber
      value={data.length}
      onChange={(cnt) => {
        setData(genData(cnt || 0));
      }}
    />
    <Table<DataType>
      rowSelection={rowSelection}
      columns={columns}
      dataSource={data}
      pagination={false}
    />
  </>
);
};

export default App;

```

Filter and sorter

```

import React from 'react';
import { Table } from 'antd';
import type { TableColumnsType, TableProps } from 'antd';

interface DataType {
  key: React.Key;
  name: string;
  age: number;
  address: string;
}

const columns: TableColumnsType<DataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
    showSorterTooltip: { target: 'full-header' },
    filters: [
      {

```

```

      text: 'Joe',
      value: 'Joe',
    },
    {
      text: 'Jim',
      value: 'Jim',
    },
    {
      text: 'Submenu',
      value: 'Submenu',
      children: [
        {
          text: 'Green',
          value: 'Green',
        },
        {
          text: 'Black',
          value: 'Black',
        },
      ],
    },
  ],
},
],
// specify the condition of filtering result
// here is that finding the name started with `value`
onFilter: (value, record) => record.name.indexOf(value as string) ===
0,
sorter: (a, b) => a.name.length - b.name.length,
sortDirections: ['descend'],
},
{
  title: 'Age',
  dataIndex: 'age',
  defaultSortOrder: 'descend',
  sorter: (a, b) => a.age - b.age,
},
{
  title: 'Address',
  dataIndex: 'address',
  filters: [
    {
      text: 'London',
      value: 'London',
    },
    {
      text: 'New York',
      value: 'New York',
    },
  ],
},

```

```

    },
  ],
  onFilter: (value, record) => record.address.indexOf(value as string)
=== 0,
},
];

const data = [
  {
    key: '1',
    name: 'John Brown',
    age: 32,
    address: 'New York No. 1 Lake Park',
  },
  {
    key: '2',
    name: 'Jim Green',
    age: 42,
    address: 'London No. 1 Lake Park',
  },
  {
    key: '3',
    name: 'Joe Black',
    age: 32,
    address: 'Sydney No. 1 Lake Park',
  },
  {
    key: '4',
    name: 'Jim Red',
    age: 32,
    address: 'London No. 2 Lake Park',
  },
];

const onChange: TableProps<DataType>['onChange'] = (pagination, filters,
sorter, extra) => {
  console.log('params', pagination, filters, sorter, extra);
};

const App: React.FC = () => (
  <Table<DataType>
    columns={columns}
    dataSource={data}
    onChange={onChange}
    showSorterTooltip={{ target: 'sorter-icon' }}
  />

```

```
);
```

```
export default App;
```

Filter in Tree

```
import React from 'react';
import { Table } from 'antd';
import type { TableColumnsType, TableProps } from 'antd';
```

```
interface DataType {
  key: React.Key;
  name: string;
  age: number;
  address: string;
}
```

```
const columns: TableColumnsType<DataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
    filters: [
      {
        text: 'Joe',
        value: 'Joe',
      },
      {
        text: 'Category 1',
        value: 'Category 1',
        children: [
          {
            text: 'Yellow',
            value: 'Yellow',
          },
          {
            text: 'Pink',
            value: 'Pink',
          },
        ],
      },
    ],
  },
  {
    text: 'Category 2',
    value: 'Category 2',
    children: [
      {
        text: 'Green',
```



```

        value: 'Green',
      },
      {
        text: 'Black',
        value: 'Black',
      },
    ],
  },
  filterMode: 'tree',
  filterSearch: true,
  onFilter: (value, record) => record.name.includes(value as string),
  width: '30%',
},
{
  title: 'Age',
  dataIndex: 'age',
  sorter: (a, b) => a.age - b.age,
},
{
  title: 'Address',
  dataIndex: 'address',
  filters: [
    {
      text: 'London',
      value: 'London',
    },
    {
      text: 'New York',
      value: 'New York',
    },
  ],
  onFilter: (value, record) => record.address.startsWith(value as
string),
  filterSearch: true,
  width: '40%',
},
];

const data: DataType[] = [
  {
    key: '1',
    name: 'John Brown',
    age: 32,
    address: 'New York No. 1 Lake Park',
  },

```

```

    {
      key: '2',
      name: 'Jim Green',
      age: 42,
      address: 'London No. 1 Lake Park',
    },
    {
      key: '3',
      name: 'Joe Black',
      age: 32,
      address: 'Sydney No. 1 Lake Park',
    },
    {
      key: '4',
      name: 'Jim Red',
      age: 32,
      address: 'London No. 2 Lake Park',
    },
  ],
];

const onChange: TableProps<DataType>['onChange'] = (pagination, filters,
sorter, extra) => {
  console.log('params', pagination, filters, sorter, extra);
};

const App: React.FC = () => (
  <Table<DataType> columns={columns} dataSource={data} onChange={onChange}
/>
);

export default App;

```

Filter search

```

import React from 'react';
import { Table } from 'antd';
import type { TableColumnsType, TableProps } from 'antd';

interface DataType {
  key: React.Key;
  name: string;
  age: number;
  address: string;
}

const columns: TableColumnsType<DataType> = [

```

```

{
  title: 'Name',
  dataIndex: 'name',
  filters: [
    {
      text: 'Joe',
      value: 'Joe',
    },
    {
      text: 'Category 1',
      value: 'Category 1',
    },
    {
      text: 'Category 2',
      value: 'Category 2',
    },
  ],
  filterMode: 'tree',
  filterSearch: true,
  onFilter: (value, record) => record.name.startsWith(value as string),
  width: '30%',
},
{
  title: 'Age',
  dataIndex: 'age',
  sorter: (a, b) => a.age - b.age,
},
{
  title: 'Address',
  dataIndex: 'address',
  filters: [
    {
      text: 'London',
      value: 'London',
    },
    {
      text: 'New York',
      value: 'New York',
    },
  ],
  onFilter: (value, record) => record.address.startsWith(value as
string),
  filterSearch: true,
  width: '40%',
},
];

```

```

const data: DataType[] = [
  {
    key: '1',
    name: 'John Brown',
    age: 32,
    address: 'New York No. 1 Lake Park',
  },
  {
    key: '2',
    name: 'Jim Green',
    age: 42,
    address: 'London No. 1 Lake Park',
  },
  {
    key: '3',
    name: 'Joe Black',
    age: 32,
    address: 'Sydney No. 1 Lake Park',
  },
  {
    key: '4',
    name: 'Jim Red',
    age: 32,
    address: 'London No. 2 Lake Park',
  },
];

const onChange: TableProps<DataType>['onChange'] = (pagination, filters,
sorter, extra) => {
  console.log('params', pagination, filters, sorter, extra);
};

const App: React.FC = () => (
  <Table<DataType> columns={columns} dataSource={data} onChange={onChange}
/>
);

export default App;

```

Multiple sorter

```

import React from 'react';
import { Table } from 'antd';
import type { TableColumnsType, TableProps } from 'antd';

```

```
interface DataType {
  key: React.Key;
  name: string;
  chinese: number;
  math: number;
  english: number;
}

const columns: TableColumnsType<DataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
  },
  {
    title: 'Chinese Score',
    dataIndex: 'chinese',
    sorter: {
      compare: (a, b) => a.chinese - b.chinese,
      multiple: 3,
    },
  },
  {
    title: 'Math Score',
    dataIndex: 'math',
    sorter: {
      compare: (a, b) => a.math - b.math,
      multiple: 2,
    },
  },
  {
    title: 'English Score',
    dataIndex: 'english',
    sorter: {
      compare: (a, b) => a.english - b.english,
      multiple: 1,
    },
  },
];

const data: DataType[] = [
  {
    key: '1',
    name: 'John Brown',
    chinese: 98,
    math: 60,
    english: 70,
  },
];
```

```

    },
    {
      key: '2',
      name: 'Jim Green',
      chinese: 98,
      math: 66,
      english: 89,
    },
    {
      key: '3',
      name: 'Joe Black',
      chinese: 98,
      math: 90,
      english: 70,
    },
    {
      key: '4',
      name: 'Jim Red',
      chinese: 88,
      math: 99,
      english: 89,
    },
  ],

  const onChange: TableProps<DataType>['onChange'] = (pagination, filters,
  sorter, extra) => {
    console.log('params', pagination, filters, sorter, extra);
  };

  const App: React.FC = () => (
    <Table<DataType> columns={columns} dataSource={data} onChange={onChange}
    />
  );

  export default App;

```

Reset filters and sorters

```

import React, { useState } from 'react';
import type { TableColumnsType, TableProps } from 'antd';
import { Button, Space, Table } from 'antd';

type OnChange = NonNullable<TableProps<DataType>['onChange']>;
type Filters = Parameters<OnChange>[1];

type GetSingle<T> = T extends (infer U)[] ? U : never;

```

```
type Sorts = GetSingle<Parameters<OnChange>[2]>;

interface DataType {
  key: string;
  name: string;
  age: number;
  address: string;
}

const data: DataType[] = [
  {
    key: '1',
    name: 'John Brown',
    age: 32,
    address: 'New York No. 1 Lake Park',
  },
  {
    key: '2',
    name: 'Jim Green',
    age: 42,
    address: 'London No. 1 Lake Park',
  },
  {
    key: '3',
    name: 'Joe Black',
    age: 32,
    address: 'Sydney No. 1 Lake Park',
  },
  {
    key: '4',
    name: 'Jim Red',
    age: 32,
    address: 'London No. 2 Lake Park',
  },
];

const App: React.FC = () => {
  const [filteredInfo, setFilteredInfo] = useState<Filters>({});
  const [sortedInfo, setSortedInfo] = useState<Sorts>({});

  const handleChange: OnChange = (pagination, filters, sorter) => {
    console.log('Various parameters', pagination, filters, sorter);
    setFilteredInfo(filters);
    setSortedInfo(sorter as Sorts);
  };
};
```

```
const clearFilters = () => {
  setFilteredInfo({});
};

const clearAll = () => {
  setFilteredInfo({});
  setSortedInfo({});
};

const setAgeSort = () => {
  setSortedInfo({
    order: 'descend',
    columnKey: 'age',
  });
};

const columns: TableColumnsType<DataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
    key: 'name',
    filters: [
      { text: 'Joe', value: 'Joe' },
      { text: 'Jim', value: 'Jim' },
    ],
    filteredValue: filteredInfo.name || null,
    onFilter: (value, record) => record.name.includes(value as string),
    sorter: (a, b) => a.name.length - b.name.length,
    sortOrder: sortedInfo.columnKey === 'name' ? sortedInfo.order : null,
    ellipsis: true,
  },
  {
    title: 'Age',
    dataIndex: 'age',
    key: 'age',
    sorter: (a, b) => a.age - b.age,
    sortOrder: sortedInfo.columnKey === 'age' ? sortedInfo.order : null,
    ellipsis: true,
  },
  {
    title: 'Address',
    dataIndex: 'address',
    key: 'address',
    filters: [
      { text: 'London', value: 'London' },
      { text: 'New York', value: 'New York' },
    ],
  },
]
```



```

    ],
    filteredValue: filteredInfo.address || null,
    onFilter: (value, record) => record.address.includes(value as
string),
    sorter: (a, b) => a.address.length - b.address.length,
    sortOrder: sortedInfo.columnKey === 'address' ? sortedInfo.order :
null,
    ellipsis: true,
  },
];

return (
  <>
    <Space style={{ marginBottom: 16 }}>
      <Button onClick={setAgeSort}>Sort age</Button>
      <Button onClick={clearFilters}>Clear filters</Button>
      <Button onClick={clearAll}>Clear filters and sorters</Button>
    </Space>
    <Table<DataType> columns={columns} dataSource={data} onChange=
{handleChange} />
  </>
);
};

export default App;

```

Customized filter panel

```

import React, { useRef, useState } from 'react';
import { SearchOutlined } from '@ant-design/icons';
import type { InputRef, TableColumnsType, TableColumnType } from 'antd';
import { Button, Input, Space, Table } from 'antd';
import type { FilterDropdownProps } from 'antd/es/table/interface';
import Highlighter from 'react-highlight-words';

interface DataType {
  key: string;
  name: string;
  age: number;
  address: string;
}

type DataIndex = keyof DataType;

const data: DataType[] = [

```

```

{
  key: '1',
  name: 'John Brown',
  age: 32,
  address: 'New York No. 1 Lake Park',
},
{
  key: '2',
  name: 'Joe Black',
  age: 42,
  address: 'London No. 1 Lake Park',
},
{
  key: '3',
  name: 'Jim Green',
  age: 32,
  address: 'Sydney No. 1 Lake Park',
},
{
  key: '4',
  name: 'Jim Red',
  age: 32,
  address: 'London No. 2 Lake Park',
},
];

const App: React.FC = () => {
  const [searchText, setSearchText] = useState('');
  const [searchedColumn, setSearchedColumn] = useState('');
  const searchInput = useRef<InputRef>(null);

  const handleSearch = (
    selectedKeys: string[],
    confirm: FilterDropdownProps['confirm'],
    dataIndex: DataIndex,
  ) => {
    confirm();
    setSearchText(selectedKeys[0]);
    setSearchedColumn(dataIndex);
  };

  const handleReset = (clearFilters: () => void) => {
    clearFilters();
    setSearchText('');
  };

```

```

const getColumnSearchProps = (dataIndex: DataIndex):
TableColumnType<DataType> => ({
  filterDropdown: ({ setSelectedKeys, selectedKeys, confirm,
clearFilters, close }) => (
    <div style={{ padding: 8 }} onKeyDown={(e) => e.stopPropagation()}>
      <Input
        ref={searchInput}
        placeholder={`Search ${dataIndex}`}
        value={selectedKeys[0]}
        onChange={(e) => setSelectedKeys(e.target.value ?
[e.target.value] : [])}
        onPressEnter={() => handleSearch(selectedKeys as string[],
confirm, dataIndex)}
        style={{ marginBottom: 8, display: 'block' }}
      />
      <Space>
        <Button
          type="primary"
          onClick={() => handleSearch(selectedKeys as string[], confirm,
dataIndex)}
          icon={<SearchOutlined />}
          size="small"
          style={{ width: 90 }}
        >
          Search
        </Button>
        <Button
          onClick={() => clearFilters && handleReset(clearFilters)}
          size="small"
          style={{ width: 90 }}
        >
          Reset
        </Button>
        <Button
          type="link"
          size="small"
          onClick={() => {
            confirm({ closeDropdown: false });
            setSearchText((selectedKeys as string[])[0]);
            setSearchedColumn(dataIndex);
          }}
        >
          Filter
        </Button>
        <Button
          type="link"

```

```

        size="small"
        onClick={() => {
            close();
        }}
    >
        close
    </Button>
</Space>
</div>
),
filterIcon: (filtered: boolean) => (
    <SearchOutlined style={{ color: filtered ? '#1677ff' : undefined }}
/>
),
onFilter: (value, record) =>
    record[dataIndex]
        .toString()
        .toLowerCase()
        .includes((value as string).toLowerCase()),
filterDropdownProps: {
    onOpenChange(open) {
        if (open) {
            setTimeout(() => searchInput.current?.select(), 100);
        }
    },
},
render: (text) =>
    searchedColumn === dataIndex ? (
        <Highlighter
            highlightStyle={{ backgroundColor: '#ffc069', padding: 0 }}
            searchWords={[searchText]}
            autoEscape
            textToHighlight={text ? text.toString() : ''}
        />
    ) : (
        text
    ),
});

const columns: TableColumnsType<DataType> = [
    {
        title: 'Name',
        dataIndex: 'name',
        key: 'name',
        width: '30%',
        ...getColumnSearchProps('name'),
    },

```

```

    },
    {
      title: 'Age',
      dataIndex: 'age',
      key: 'age',
      width: '20%',
      ...getColumnSearchProps('age'),
    },
    {
      title: 'Address',
      dataIndex: 'address',
      key: 'address',
      ...getColumnSearchProps('address'),
      sorter: (a, b) => a.address.length - b.address.length,
      sortDirections: ['descend', 'ascend'],
    },
  ],
];

return <Table<DataType> columns={columns} dataSource={data} />;
};

export default App;

```

Ajax

```

/* eslint-disable compat/compat */
import React, { useEffect, useState } from 'react';
import type { GetProp, TableProps } from 'antd';
import { Table } from 'antd';
import type { AnyObject } from 'antd/es/_util/type';
import type { SorterResult } from 'antd/es/table/interface';

type ColumnsType<T extends object = object> = TableProps<T>['columns'];
type TablePaginationConfig = Exclude<GetProp<TableProps, 'pagination'>,
boolean>;

interface DataType {
  name: {
    first: string;
    last: string;
  };
  gender: string;
  email: string;
  login: {
    uuid: string;
  };
};

```

```

}

interface TableParams {
  pagination?: TablePaginationConfig;
  sortField?: SorterResult<any>['field'];
  sortOrder?: SorterResult<any>['order'];
  filters?: Parameters<GetProp<TableProps, 'onChange'>>[1];
}

const columns: ColumnsType<DataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
    sorter: true,
    render: (name) => `${name.first} ${name.last}`,
    width: '20%',
  },
  {
    title: 'Gender',
    dataIndex: 'gender',
    filters: [
      { text: 'Male', value: 'male' },
      { text: 'Female', value: 'female' },
    ],
    width: '20%',
  },
  {
    title: 'Email',
    dataIndex: 'email',
  },
];

const toURLSearchParams = <T extends AnyObject>(record: T) => {
  const params = new URLSearchParams();
  for (const [key, value] of Object.entries(record)) {
    params.append(key, value);
  }
  return params;
};

const getRandomuserParams = (params: TableParams) => ({
  results: params.pagination?.pageSize,
  page: params.pagination?.current,
  ...params,
});

```

```

const App: React.FC = () => {
  const [data, setData] = useState<DataType[]>();
  const [loading, setLoading] = useState(false);
  const [tableParams, setTableParams] = useState<TableParams>({
    pagination: {
      current: 1,
      pageSize: 10,
    },
  });

  const params = toURLSearchParams(getRandomuserParams(tableParams));

  const fetchData = () => {
    setLoading(true);
    fetch(`https://randomuser.me/api?${params.toString()}`)
      .then((res) => res.json())
      .then(({ results }) => {
        setData(results);
        setLoading(false);
        setTableParams({
          ...tableParams,
          pagination: {
            ...tableParams.pagination,
            total: 200,
            // 200 is mock data, you should read it from server
            // total: data.totalCount,
          },
        });
      });
  };

  useEffect(fetchData, [
    tableParams.pagination?.current,
    tableParams.pagination?.pageSize,
    tableParams?.sortOrder,
    tableParams?.sortField,
    JSON.stringify(tableParams.filters),
  ]);

  const handleTableChange: TableProps<DataType>['onChange'] = (pagination,
filters, sorter) => {
    setTableParams({
      pagination,
      filters,
      sortOrder: Array.isArray(sorter) ? undefined : sorter.order,
      sortField: Array.isArray(sorter) ? undefined : sorter.field,
    });
  };

```

```

    });

    // `dataSource` is useless since `pageSize` changed
    if (pagination.pageSize !== tableParams.pagination?.pageSize) {
      setData([]);
    }
  };

  return (
    <Table<DataType>
      columns={columns}
      rowKey={(record) => record.login.uuid}
      dataSource={data}
      pagination={tableParams.pagination}
      loading={loading}
      onChange={handleTableChange}
    />
  );
};

export default App;

```

size

```

import React from 'react';
import { Divider, Table } from 'antd';
import type { TableColumnsType } from 'antd';

interface DataType {
  key: React.Key;
  name: string;
  age: number;
  address: string;
}

const columns: TableColumnsType<DataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
  },
  {
    title: 'Age',
    dataIndex: 'age',
  },
  {
    title: 'Address',

```



```

        dataIndex: 'address',
      },
    ];

const data: DataType[] = [
  {
    key: '1',
    name: 'John Brown',
    age: 32,
    address: 'New York No. 1 Lake Park',
  },
  {
    key: '2',
    name: 'Jim Green',
    age: 42,
    address: 'London No. 1 Lake Park',
  },
  {
    key: '3',
    name: 'Joe Black',
    age: 32,
    address: 'Sydney No. 1 Lake Park',
  },
];

const App: React.FC = () => (
  <>
    <Divider>Middle size table</Divider>
    <Table<DataType> columns={columns} dataSource={data} size="middle" />
    <Divider>Small size table</Divider>
    <Table<DataType> columns={columns} dataSource={data} size="small" />
  </>
);

export default App;

```

size

Debug

```

import React from 'react';
import { Table } from 'antd';
import type { TableColumnsType } from 'antd';

interface DataType {
  key: React.Key;

```

```

    name: string;
    age: number;
    address: string;
  }

const columns: TableColumnsType<DataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
  },
  {
    title: 'Age',
    dataIndex: 'age',
  },
  {
    title: 'Address',
    dataIndex: 'address',
  },
];

const dataSource = Array.from({ length: 200 }).map<DataType>((_, key) => ({
  key,
  name: 'Sample Name',
  age: 30 + (key % 5),
  address: `Sample Address ${key}`,
}));

const App: React.FC = () => (
  <div style={{ width: 300 }}>
    <Table<DataType>
      columns={columns}
      dataSource={dataSource}
      size="small"
      pagination={{ defaultCurrent: 13 }}
    />
  </div>
);

export default App;

```

border, title and footer

```

import React from 'react';
import { Table } from 'antd';
import type { TableProps } from 'antd';

```

```
interface DataType {
  key: string;
  name: string;
  money: string;
  address: string;
}

const columns: TableProps<DataType>['columns'] = [
  {
    title: 'Name',
    dataIndex: 'name',
    render: (text) => <a>{text}</a>,
  },
  {
    title: 'Cash Assets',
    className: 'column-money',
    dataIndex: 'money',
    align: 'right',
  },
  {
    title: 'Address',
    dataIndex: 'address',
  },
];

const data: DataType[] = [
  {
    key: '1',
    name: 'John Brown',
    money: '¥300,000.00',
    address: 'New York No. 1 Lake Park',
  },
  {
    key: '2',
    name: 'Jim Green',
    money: '¥1,256,000.00',
    address: 'London No. 1 Lake Park',
  },
  {
    key: '3',
    name: 'Joe Black',
    money: '¥120,000.00',
    address: 'Sydney No. 1 Lake Park',
  },
];
```

```

const App: React.FC = () => (
  <Table<DataType>
    columns={columns}
    dataSource={data}
    bordered
    title={() => 'Header'}
    footer={() => 'Footer'}
  />
);

export default App;

```

Expandable Row

```

import React from 'react';
import { Table } from 'antd';
import type { TableColumnsType } from 'antd';

interface DataType {
  key: React.Key;
  name: string;
  age: number;
  address: string;
  description: string;
}

const columns: TableColumnsType<DataType> = [
  { title: 'Name', dataIndex: 'name', key: 'name' },
  { title: 'Age', dataIndex: 'age', key: 'age' },
  { title: 'Address', dataIndex: 'address', key: 'address' },
  {
    title: 'Action',
    dataIndex: '',
    key: 'x',
    render: () => <a>Delete</a>,
  },
];

const data: DataType[] = [
  {
    key: 1,
    name: 'John Brown',
    age: 32,
    address: 'New York No. 1 Lake Park',
    description: 'My name is John Brown, I am 32 years old, living in New

```

```

York No. 1 Lake Park.',
  },
  {
    key: 2,
    name: 'Jim Green',
    age: 42,
    address: 'London No. 1 Lake Park',
    description: 'My name is Jim Green, I am 42 years old, living in London
No. 1 Lake Park.',
  },
  {
    key: 3,
    name: 'Not Expandable',
    age: 29,
    address: 'Jiangsu No. 1 Lake Park',
    description: 'This not expandable',
  },
  {
    key: 4,
    name: 'Joe Black',
    age: 32,
    address: 'Sydney No. 1 Lake Park',
    description: 'My name is Joe Black, I am 32 years old, living in Sydney
No. 1 Lake Park.',
  },
];

const App: React.FC = () => (
  <Table<DataType>
    columns={columns}
    expandable={{
      expandedRowRender: (record) => <p style={{ margin: 0 }}>
{record.description}</p>,
      rowExpandable: (record) => record.name !== 'Not Expandable',
    }}
    dataSource={data}
  />
);

export default App;

```

Order Specific Column

```

import React from 'react';
import { Table } from 'antd';
import type { TableColumnsType } from 'antd';

```

```
interface DataType {
  key: React.Key;
  name: string;
  age: number;
  address: string;
  description: string;
}

const columns: TableColumnsType<DataType> = [
  { title: 'Name', dataIndex: 'name', key: 'name' },
  Table.EXPAND_COLUMN,
  { title: 'Age', dataIndex: 'age', key: 'age' },
  Table.SELECTION_COLUMN,
  { title: 'Address', dataIndex: 'address', key: 'address' },
];

const data: DataType[] = [
  {
    key: 1,
    name: 'John Brown',
    age: 32,
    address: 'New York No. 1 Lake Park',
    description: 'My name is John Brown, I am 32 years old, living in New York No. 1 Lake Park.',
  },
  {
    key: 2,
    name: 'Jim Green',
    age: 42,
    address: 'London No. 1 Lake Park',
    description: 'My name is Jim Green, I am 42 years old, living in London No. 1 Lake Park.',
  },
  {
    key: 3,
    name: 'Not Expandable',
    age: 29,
    address: 'Jiangsu No. 1 Lake Park',
    description: 'This not expandable',
  },
  {
    key: 4,
    name: 'Joe Black',
    age: 32,
    address: 'Sydney No. 1 Lake Park',
  },
];
```

```

        description: 'My name is Joe Black, I am 32 years old, living in Sydney
No. 1 Lake Park.',
    },
];

const App: React.FC = () => (
  <Table<DataType>
    columns={columns}
    rowSelection={{}}
    expandable={{
      expandedRowRender: (record) => <p style={{ margin: 0 }}>
{record.description}</p>,
    }}
    dataSource={data}
  />
);

export default App;

```

colSpan and rowspan

```

import React from 'react';
import { Table } from 'antd';
import type { TableProps } from 'antd';

interface DataType {
  key: string;
  name: string;
  age: number;
  tel: string;
  phone: number;
  address: string;
}

// In the fifth row, other columns are merged into first column
// by setting it's colSpan to be 0
const sharedOnCell = (_, DataType, index?: number) => {
  if (index === 1) {
    return { colSpan: 0 };
  }

  return {};
};

const columns: TableProps<DataType>['columns'] = [
  {

```

```

    title: 'RowHead',
    dataIndex: 'key',
    rowScope: 'row',
  },
  {
    title: 'Name',
    dataIndex: 'name',
    render: (text) => <a>{text}</a>,
    onCell: (_, index) => ({
      colSpan: index === 1 ? 5 : 1,
    }),
  },
  {
    title: 'Age',
    dataIndex: 'age',
    onCell: sharedOnCell,
  },
  {
    title: 'Home phone',
    colSpan: 2,
    dataIndex: 'tel',
    onCell: (_, index) => {
      if (index === 3) {
        return { rowSpan: 2 };
      }
      // These two are merged into above cell
      if (index === 4) {
        return { rowSpan: 0 };
      }
      if (index === 1) {
        return { colSpan: 0 };
      }

      return {};
    },
  },
  {
    title: 'Phone',
    colSpan: 0,
    dataIndex: 'phone',
    onCell: sharedOnCell,
  },
  {
    title: 'Address',
    dataIndex: 'address',
    onCell: sharedOnCell,
  },

```



```
    },  
  ],  
];  
  
const data: DataType[] = [  
  {  
    key: '1',  
    name: 'John Brown',  
    age: 32,  
    tel: '0571-22098909',  
    phone: 18889898989,  
    address: 'New York No. 1 Lake Park',  
  },  
  {  
    key: '2',  
    name: 'Jim Green',  
    tel: '0571-22098333',  
    phone: 18889898888,  
    age: 42,  
    address: 'London No. 1 Lake Park',  
  },  
  {  
    key: '3',  
    name: 'Joe Black',  
    age: 32,  
    tel: '0575-22098909',  
    phone: 18900010002,  
    address: 'Sydney No. 1 Lake Park',  
  },  
  {  
    key: '4',  
    name: 'Jim Red',  
    age: 18,  
    tel: '0575-22098909',  
    phone: 18900010002,  
    address: 'London No. 2 Lake Park',  
  },  
  {  
    key: '5',  
    name: 'Jake White',  
    age: 18,  
    tel: '0575-22098909',  
    phone: 18900010002,  
    address: 'Dublin No. 2 Lake Park',  
  },  
];
```

```
const App: React.FC = () => <Table<DataType> columns={columns} dataSource=
{data} bordered />;

export default App;
```

Tree data

```
import React, { useState } from 'react';
import { Space, Switch, Table } from 'antd';
import type { TableColumnsType, TableProps } from 'antd';

type TableRowSelection<T extends object = object> = TableProps<T>
['rowSelection'];

interface DataType {
  key: React.ReactNode;
  name: string;
  age: number;
  address: string;
  children?: DataType[];
}

const columns: TableColumnsType<DataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
    key: 'name',
  },
  {
    title: 'Age',
    dataIndex: 'age',
    key: 'age',
    width: '12%',
  },
  {
    title: 'Address',
    dataIndex: 'address',
    width: '30%',
    key: 'address',
  },
];

const data: DataType[] = [
  {
    key: 1,
    name: 'John Brown sr.',
```

```
age: 60,
address: 'New York No. 1 Lake Park',
children: [
  {
    key: 11,
    name: 'John Brown',
    age: 42,
    address: 'New York No. 2 Lake Park',
  },
  {
    key: 12,
    name: 'John Brown jr.',
    age: 30,
    address: 'New York No. 3 Lake Park',
    children: [
      {
        key: 121,
        name: 'Jimmy Brown',
        age: 16,
        address: 'New York No. 3 Lake Park',
      },
    ],
  },
],
},
{
  key: 13,
  name: 'Jim Green sr.',
  age: 72,
  address: 'London No. 1 Lake Park',
  children: [
    {
      key: 131,
      name: 'Jim Green',
      age: 42,
      address: 'London No. 2 Lake Park',
      children: [
        {
          key: 1311,
          name: 'Jim Green jr.',
          age: 25,
          address: 'London No. 3 Lake Park',
        },
      ],
    },
    {
      key: 1312,
      name: 'Jimmy Green sr.',
      age: 18,
      address: 'London No. 4 Lake Park',
    },
  ],
},
],
}
```

```

        },
      ],
    },
  ],
},
],
},
{
  key: 2,
  name: 'Joe Black',
  age: 32,
  address: 'Sydney No. 1 Lake Park',
},
];

// rowSelection objects indicates the need for row selection
const rowSelection: TableRowSelection<DataType> = {
  onChange: (selectedRowKeys, selectedRows) => {
    console.log(`selectedRowKeys: ${selectedRowKeys}`, 'selectedRows: ',
selectedRows);
  },
  onSelect: (record, selected, selectedRows) => {
    console.log(record, selected, selectedRows);
  },
  onSelectAll: (selected, selectedRows, changeRows) => {
    console.log(selected, selectedRows, changeRows);
  },
};

const App: React.FC = () => {
  const [checkStrictly, setCheckStrictly] = useState(false);

  return (
    <>
      <Space align="center" style={{ marginBottom: 16 }}>
        CheckStrictly: <Switch checked={checkStrictly} onChange=
{setCheckStrictly} />
      </Space>
      <Table<DataType>
        columns={columns}
        rowSelection={{ ...rowSelection, checkStrictly }}
        dataSource={data}
      />
    </>
  );
};

```

```
export default App;
```

Tree data ellipsis debug demo

Debug

```
import React, { useState } from 'react';
import { Space, Switch, Table } from 'antd';
import type { TableColumnType } from 'antd';

interface DataType {
  key: React.ReactNode;
  name: string;
  age: number;
  address: string;
  children?: DataType[];
}

const data: DataType[] = [
  {
    key: 1,
    name: 'John Brown sr. John Brown sr. John Brown sr. John Brown sr. John Brown sr. John Brown sr.',
    age: 60,
    address: 'New York No. 1 Lake Park',
    children: [
      {
        key: 11,
        name: 'John Brown sr. John Brown sr. John Brown sr. John Brown sr. John Brown sr. John Brown sr.',
        age: 42,
        address: 'New York No. 2 Lake Park',
      },
      {
        key: 12,
        name: 'John Brown sr. John Brown sr. John Brown sr. John Brown sr. John Brown sr. John Brown sr.',
        age: 30,
        address: 'New York No. 3 Lake Park',
        children: [
          {
            key: 121,
            name: 'John Brown sr. John Brown sr. John Brown sr. John Brown sr. John Brown sr. John Brown sr.',

```

```

    age: 16,
    address: 'New York No. 3 Lake Park',
  },
],
},
{
  key: 13,
  name: 'Jim Green sr. Jim Green sr. Jim Green sr. Jim Green sr.',
  age: 72,
  address: 'London No. 1 Lake Park',
  children: [
    {
      key: 131,
      name: 'Jim Green. Jim Green. Jim Green. Jim Green. Jim Green.
Jim Green.',
      age: 42,
      address: 'London No. 2 Lake Park',
      children: [
        {
          key: 1311,
          name: 'Jim Green jr. Jim Green jr. Jim Green jr. Jim Green
jr.',
          age: 25,
          address: 'London No. 3 Lake Park',
        },
        {
          key: 1312,
          name: 'Jimmy Green sr. Jimmy Green sr. Jimmy Green sr.',
          age: 18,
          address: 'London No. 4 Lake Park',
        },
      ],
    },
  ],
},
],
},
],
},
],
},
{
  key: 2,
  name: 'Joe Black',
  age: 32,
  address: 'Sydney No. 1 Lake Park',
},
];

const App: React.FC = () => {

```

```

const [fixed, setFixed] = useState(true);

const columns: TableColumnsType<DataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
    key: 'name',
    width: '30%',
    ellipsis: true,
    fixed,
  },
  {
    title: 'Age',
    dataIndex: 'age',
    key: 'age',
    width: '12%',
  },
  {
    title: 'Address',
    dataIndex: 'address',
    key: 'address',
  },
];

return (
  <>
    <Space align="center" style={{ marginBottom: 16 }}>
      Fixed first column: <Switch checked={fixed} onChange={setFixed} />
    </Space>
    <Table<DataType>
      columns={columns}
      rowSelection={{ columnWidth: 100 }}
      expandable={{ defaultExpandAllRows: true }}
      dataSource={data}
    />
  </>
);
};

export default App;

```

Fixed Header

```

import React from 'react';
import { Table } from 'antd';

```

```

import type { TableColumnsType } from 'antd';
import { createStyles } from 'antd-style';

const useStyles = createStyles(({ css, token }) => {
  const { antCls } = token;
  return {
    customTable: css`
      ${antCls}-table {
        ${antCls}-table-container {
          ${antCls}-table-body,
          ${antCls}-table-content {
            scrollbar-width: thin;
            scrollbar-color: #eaeaea transparent;
            scrollbar-gutter: stable;
          }
        }
      }
    `,
  };
});

interface DataType {
  key: React.Key;
  name: string;
  age: number;
  address: string;
}

const columns: TableColumnsType<DataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
    width: 150,
  },
  {
    title: 'Age',
    dataIndex: 'age',
    width: 150,
  },
  {
    title: 'Address',
    dataIndex: 'address',
  },
];

const dataSource = Array.from({ length: 100 }).map<DataType>((_, i) => ({

```



```

    key: i,
    name: `Edward King ${i}`,
    age: 32,
    address: `London, Park Lane no. ${i}`,
  }));

const App: React.FC = () => {
  const { styles } = useStyles();
  return (
    <Table<DataType>
      className={styles.customTable}
      columns={columns}
      dataSource={dataSource}
      pagination={{ pageSize: 50 }}
      scroll={{ y: 55 * 5 }}
    />
  );
};

export default App;

```

Fixed Columns

```

import React from 'react';
import { Table } from 'antd';
import type { TableColumnsType } from 'antd';
import { createStyles } from 'antd-style';

const useStyles = createStyles(({ css, token }) => {
  const { antCls } = token;
  return {
    customTable: css`
      ${antCls}-table {
        ${antCls}-table-container {
          ${antCls}-table-body,
          ${antCls}-table-content {
            scrollbar-width: thin;
            scrollbar-color: #eaeaea transparent;
            scrollbar-gutter: stable;
          }
        }
      }
    `,
  };
});

```

```
interface DataType {
  key: React.Key;
  name: string;
  age: number;
  address: string;
}

const columns: TableColumnsType<DataType> = [
  {
    title: 'Full Name',
    width: 100,
    dataIndex: 'name',
    key: 'name',
    fixed: 'left',
  },
  {
    title: 'Age',
    width: 100,
    dataIndex: 'age',
    key: 'age',
    fixed: 'left',
    sorter: true,
  },
  { title: 'Column 1', dataIndex: 'address', key: '1' },
  { title: 'Column 2', dataIndex: 'address', key: '2' },
  { title: 'Column 3', dataIndex: 'address', key: '3' },
  { title: 'Column 4', dataIndex: 'address', key: '4' },
  { title: 'Column 5', dataIndex: 'address', key: '5' },
  { title: 'Column 6', dataIndex: 'address', key: '6' },
  { title: 'Column 7', dataIndex: 'address', key: '7' },
  { title: 'Column 8', dataIndex: 'address', key: '8' },
  { title: 'Column 9', dataIndex: 'address', key: '9' },
  { title: 'Column 10', dataIndex: 'address', key: '10' },
  { title: 'Column 11', dataIndex: 'address', key: '11' },
  { title: 'Column 12', dataIndex: 'address', key: '12' },
  { title: 'Column 13', dataIndex: 'address', key: '13' },
  { title: 'Column 14', dataIndex: 'address', key: '14' },
  { title: 'Column 15', dataIndex: 'address', key: '15' },
  { title: 'Column 16', dataIndex: 'address', key: '16' },
  { title: 'Column 17', dataIndex: 'address', key: '17' },
  { title: 'Column 18', dataIndex: 'address', key: '18' },
  { title: 'Column 19', dataIndex: 'address', key: '19' },
  { title: 'Column 20', dataIndex: 'address', key: '20' },
  {
    title: 'Action',
    key: 'operation',
  },
]
```

```

    fixed: 'right',
    width: 100,
    render: () => <a>action</a>,
  },
];

const dataSource: DataType[] = [
  { key: '1', name: 'Olivia', age: 32, address: 'New York Park' },
  { key: '2', name: 'Ethan', age: 40, address: 'London Park' },
];

const App: React.FC = () => {
  const { styles } = useStyles();
  return (
    <Table<DataType>
      className={styles.customTable}
      pagination={false}
      columns={columns}
      dataSource={dataSource}
      scroll={{ x: 'max-content' }}
    />
  );
};

export default App;

```

Stack Fixed Columns

v5.14.0

```

import React from 'react';
import { Table } from 'antd';
import type { TableColumnsType } from 'antd';
import { createStyles } from 'antd-style';

const useStyles = createStyles(({ css, token }) => {
  const { antCls } = token;
  return {
    customTable: css`
      ${antCls}-table {
        ${antCls}-table-container {
          ${antCls}-table-body,
          ${antCls}-table-content {
            scrollbar-width: thin;
            scrollbar-color: #eaeaea transparent;
            scrollbar-gutter: stable;
          }
        }
      }
    `
  };
});

```

```

    }
  }
},
};
});

```

```

interface DataType {
  key: React.Key;
  name: string;
  age: number;
  address: string;
}

```

```

const columns: TableColumnsType<DataType> = [
  {
    title: 'Full Name',
    width: 100,
    dataIndex: 'name',
    fixed: 'left',
  },
  {
    title: 'Age',
    width: 100,
    dataIndex: 'age',
  },
  { title: 'Column 1', dataIndex: 'address', key: '1', fixed: 'left' },
  { title: 'Column 2', dataIndex: 'address', key: '2' },
  { title: 'Column 3', dataIndex: 'address', key: '3' },
  { title: 'Column 4', dataIndex: 'address', key: '4' },
  { title: 'Column 5', dataIndex: 'address', key: '5' },
  { title: 'Column 6', dataIndex: 'address', key: '6' },
  { title: 'Column 7', dataIndex: 'address', key: '7' },
  { title: 'Column 8', dataIndex: 'address', key: '8' },
  { title: 'Column 9', dataIndex: 'address', key: '9' },
  { title: 'Column 10', dataIndex: 'address', key: '10' },
  { title: 'Column 11', dataIndex: 'address', key: '11' },
  { title: 'Column 12', dataIndex: 'address', key: '12' },
  { title: 'Column 13', dataIndex: 'address', key: '13' },
  { title: 'Column 14', dataIndex: 'address', key: '14' },
  { title: 'Column 15', dataIndex: 'address', key: '15' },
  { title: 'Column 16', dataIndex: 'address', key: '16' },
  { title: 'Column 17', dataIndex: 'address', key: '17' },
  { title: 'Column 18', dataIndex: 'address', key: '18' },
  { title: 'Column 19', dataIndex: 'address', key: '19' },
  { title: 'Column 20', dataIndex: 'address', key: '20' },

```

```

    {
      title: 'Action 1',
      fixed: 'right',
      width: 90,
      render: () => <a>action</a>,
    },
    {
      title: 'Action 2',
      width: 90,
      render: () => <a>action</a>,
    },
    {
      title: 'Action 3',
      fixed: 'right',
      width: 90,
      render: () => <a>action</a>,
    },
  ],
];

const dataSource: DataType[] = [
  { key: '1', name: 'Olivia', age: 32, address: 'New York Park' },
  { key: '2', name: 'Ethan', age: 40, address: 'London Park' },
];

const App: React.FC = () => {
  const { styles } = useStyles();
  return (
    <Table<DataType>
      bordered
      className={styles.customTable}
      columns={columns}
      dataSource={dataSource}
      scroll={{ x: 'max-content' }}
      pagination={false}
    />
  );
};

export default App;

```

Fixed Columns and Header

```

import React from 'react';
import { Table } from 'antd';
import type { TableColumnsType } from 'antd';
import { createStyles } from 'antd-style';

```

```

const useStyles = createStyles(({ css, token }) => {
  const { antCls } = token;
  return {
    customTable: css`
      ${antCls}-table {
        ${antCls}-table-container {
          ${antCls}-table-body,
          ${antCls}-table-content {
            scrollbar-width: thin;
            scrollbar-color: #eaeaea transparent;
            scrollbar-gutter: stable;
          }
        }
      }
    `,
  };
});

```

```

interface DataType {
  key: React.Key;
  name: string;
  age: number;
  address: string;
}

```

```

const columns: TableColumnsType<DataType> = [
  {
    title: 'Full Name',
    width: 100,
    dataIndex: 'name',
    key: 'name',
    fixed: 'left',
  },
  {
    title: 'Age',
    width: 100,
    dataIndex: 'age',
    key: 'age',
    fixed: 'left',
  },
  {
    title: 'Column 1',
    dataIndex: 'address',
    key: '1',
    width: 150,
  },
];

```

```
},
{
  title: 'Column 2',
  dataIndex: 'address',
  key: '2',
  width: 150,
},
{
  title: 'Column 3',
  dataIndex: 'address',
  key: '3',
  width: 150,
},
{
  title: 'Column 4',
  dataIndex: 'address',
  key: '4',
  width: 150,
},
{
  title: 'Column 5',
  dataIndex: 'address',
  key: '5',
  width: 150,
},
{
  title: 'Column 6',
  dataIndex: 'address',
  key: '6',
  width: 150,
},
{
  title: 'Column 7',
  dataIndex: 'address',
  key: '7',
  width: 150,
},
{ title: 'Column 8', dataIndex: 'address', key: '8' },
{ title: 'Column 9', dataIndex: 'address', key: '9' },
{ title: 'Column 10', dataIndex: 'address', key: '10' },
{ title: 'Column 11', dataIndex: 'address', key: '11' },
{ title: 'Column 12', dataIndex: 'address', key: '12' },
{ title: 'Column 13', dataIndex: 'address', key: '13' },
{ title: 'Column 14', dataIndex: 'address', key: '14' },
{ title: 'Column 15', dataIndex: 'address', key: '15' },
{ title: 'Column 16', dataIndex: 'address', key: '16' },
```

```

    { title: 'Column 17', dataIndex: 'address', key: '17' },
    { title: 'Column 18', dataIndex: 'address', key: '18' },
    { title: 'Column 19', dataIndex: 'address', key: '19' },
    { title: 'Column 20', dataIndex: 'address', key: '20' },
    {
      title: 'Action',
      key: 'operation',
      fixed: 'right',
      width: 100,
      render: () => <a>action</a>,
    },
  ],
];

const dataSource = Array.from({ length: 100 }).map<DataType>((_, i) => ({
  key: i,
  name: `Edward King ${i}`,
  age: 32,
  address: `London, Park Lane no. ${i}`,
}));

const App: React.FC = () => {
  const { styles } = useStyles();
  return (
    <Table<DataType>
      className={styles.customTable}
      columns={columns}
      dataSource={dataSource}
      scroll={{ x: 'max-content', y: 55 * 5 }}
    />
  );
};

export default App;

```

Hidden Columns

v5.13.0

```

import React, { useState } from 'react';
import { Checkbox, Divider, Table } from 'antd';
import type { CheckboxOptionType, TableColumnsType } from 'antd';

interface DataType {
  key: React.Key;
  name: string;
  age: number;
}

```



```

    address: string;
}

const columns: TableColumnsType<DataType> = [
  { title: 'Column 1', dataIndex: 'address', key: '1' },
  { title: 'Column 2', dataIndex: 'address', key: '2' },
  { title: 'Column 3', dataIndex: 'address', key: '3' },
  { title: 'Column 4', dataIndex: 'address', key: '4' },
  { title: 'Column 5', dataIndex: 'address', key: '5' },
  { title: 'Column 6', dataIndex: 'address', key: '6' },
  { title: 'Column 7', dataIndex: 'address', key: '7' },
  { title: 'Column 8', dataIndex: 'address', key: '8' },
];

const data: DataType[] = [
  {
    key: '1',
    name: 'John Brown',
    age: 32,
    address: 'New York Park',
  },
  {
    key: '2',
    name: 'Jim Green',
    age: 40,
    address: 'London Park',
  },
];

const defaultCheckedList = columns.map((item) => item.key);

const App: React.FC = () => {
  const [checkedList, setCheckedList] = useState(defaultCheckedList);

  const options = columns.map(({ key, title }) => ({
    label: title,
    value: key,
  }));

  const newColumns = columns.map((item) => ({
    ...item,
    hidden: !checkedList.includes(item.key as string),
  }));

  return (
    <>

```

```

    <Divider>Columns displayed</Divider>
    <Checkbox.Group
      value={checkedList}
      options={options as CheckboxOptionType[]}
      onChange={(value) => {
        setCheckedList(value as string[]);
      }}
    />
    <Table<DataType> columns={newColumns} dataSource={data} style={{
marginTop: 24 }} />
  </>
);
};

export default App;

```

Grouping table head

```

import React from 'react';
import { Table } from 'antd';
import type { TableColumnsType } from 'antd';
import { createStyles } from 'antd-style';

const useStyles = createStyles(({ css, token }) => {
  const { antCls } = token;
  return {
    customTable: css`
      ${antCls}-table {
        ${antCls}-table-container {
          ${antCls}-table-body,
          ${antCls}-table-content {
            scrollbar-width: thin;
            scrollbar-color: #eaeaea transparent;
            scrollbar-gutter: stable;
          }
        }
      }
    `,
  };
});

interface DataType {
  key: React.Key;
  name: string;
  age: number;
}

```

```

street: string;
building: string;
number: number;
companyAddress: string;
companyName: string;
gender: string;
}

const columns: TableColumnsType<DataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
    key: 'name',
    width: 100,
    fixed: 'left',
    filters: [
      {
        text: 'Joe',
        value: 'Joe',
      },
      {
        text: 'John',
        value: 'John',
      },
    ],
    onFilter: (value, record) => record.name.indexOf(value as string) ===
0,
  },
  {
    title: 'Other',
    children: [
      {
        title: 'Age',
        dataIndex: 'age',
        key: 'age',
        width: 150,
        sorter: (a, b) => a.age - b.age,
      },
      {
        title: 'Address',
        children: [
          {
            title: 'Street',
            dataIndex: 'street',
            key: 'street',
            width: 150,

```

```

    },
    {
      title: 'Block',
      children: [
        {
          title: 'Building',
          dataIndex: 'building',
          key: 'building',
          width: 100,
        },
        {
          title: 'Door No.',
          dataIndex: 'number',
          key: 'number',
          width: 100,
        },
      ],
    },
  ],
},
],
},
],
},
],
},
{
  title: 'Company',
  children: [
    {
      title: 'Company Address',
      dataIndex: 'companyAddress',
      key: 'companyAddress',
      width: 200,
    },
    {
      title: 'Company Name',
      dataIndex: 'companyName',
      key: 'companyName',
    },
  ],
},
],
},
{
  title: 'Gender',
  dataIndex: 'gender',
  key: 'gender',
  width: 80,
  fixed: 'right',
},
],
];

```

```

const dataSource = Array.from({ length: 100 }).map<DataType>((_, i) => ({
  key: i,
  name: 'John Brown',
  age: i + 1,
  street: 'Lake Park',
  building: 'C',
  number: 2035,
  companyAddress: 'Lake Street 42',
  companyName: 'SoftLake Co',
  gender: 'M',
}));

const App: React.FC = () => {
  const { styles } = useStyles();
  return (
    <Table<DataType>
      className={styles.customTable}
      columns={columns}
      dataSource={dataSource}
      bordered
      size="middle"
      scroll={{ x: 'calc(700px + 50%)', y: 47 * 5 }}
    />
  );
};

export default App;

```

Editable Cells

```

import React, { useContext, useEffect, useRef, useState } from 'react';
import type { GetRef, InputRef, TableProps } from 'antd';
import { Button, Form, Input, Popconfirm, Table } from 'antd';

type FormInstance<T> = GetRef<typeof Form<T>>;

const EditableContext = React.createContext<FormInstance<any> | null>
(null);

interface Item {
  key: string;
  name: string;
  age: string;
  address: string;
}

```

```

interface EditableRowProps {
  index: number;
}

const EditableRow: React.FC<EditableRowProps> = ({ index, ...props }) => {
  const [form] = Form.useForm();
  return (
    <Form form={form} component={false}>
      <EditableContext.Provider value={form}>
        <tr {...props} />
      </EditableContext.Provider>
    </Form>
  );
};

interface EditableCellProps {
  title: React.ReactNode;
  editable: boolean;
  dataIndex: keyof Item;
  record: Item;
  handleSave: (record: Item) => void;
}

const EditableCell: React.FC<React.PropsWithChildren<EditableCellProps>> =
({
  title,
  editable,
  children,
  dataIndex,
  record,
  handleSave,
  ...restProps
}) => {
  const [editing, setEditing] = useState(false);
  const inputRef = useRef<InputRef>(null);
  const form = useContext(EditableContext)!;

  useEffect(() => {
    if (editing) {
      inputRef.current?.focus();
    }
  }, [editing]);

  const toggleEdit = () => {
    setEditing(!editing);
  }

```

```

    form.setFieldsValue({ [dataIndex]: record[dataIndex] });
};

const save = async () => {
  try {
    const values = await form.validateFields();

    toggleEdit();
    handleSave({ ...record, ...values });
  } catch (errInfo) {
    console.log('Save failed:', errInfo);
  }
};

let childNode = children;

if (editable) {
  childNode = editing ? (
    <Form.Item
      style={{ margin: 0 }}
      name={dataIndex}
      rules={[{ required: true, message: `${title} is required.` }]}
    >
      <Input ref={inputRef} onPressEnter={save} onBlur={save} />
    </Form.Item>
  ) : (
    <div
      className="editable-cell-value-wrap"
      style={{ paddingInlineEnd: 24 }}
      onClick={toggleEdit}
    >
      {children}
    </div>
  );
}

return <td {...restProps}>{childNode}</td>;
};

interface DataType {
  key: React.Key;
  name: string;
  age: string;
  address: string;
}

```

```

type ColumnTypes = Exclude<TableProps<DataType>['columns'], undefined>;

const App: React.FC = () => {
  const [dataSource, setDataSource] = useState<DataType[]>([
    {
      key: '0',
      name: 'Edward King 0',
      age: '32',
      address: 'London, Park Lane no. 0',
    },
    {
      key: '1',
      name: 'Edward King 1',
      age: '32',
      address: 'London, Park Lane no. 1',
    },
  ]);

  const [count, setCount] = useState(2);

  const handleDelete = (key: React.Key) => {
    const newData = dataSource.filter((item) => item.key !== key);
    setDataSource(newData);
  };

  const defaultColumns: (ColumnTypes[number] & { editable?: boolean;
  dataIndex: string })[] = [
    {
      title: 'name',
      dataIndex: 'name',
      width: '30%',
      editable: true,
    },
    {
      title: 'age',
      dataIndex: 'age',
    },
    {
      title: 'address',
      dataIndex: 'address',
    },
    {
      title: 'operation',
      dataIndex: 'operation',
      render: (_, record) =>
        dataSource.length >= 1 ? (

```



```

        <Popconfirm title="Sure to delete?" onConfirm={() =>
handleDelete(record.key)}>
          <a>Delete</a>
        </Popconfirm>
      ) : null,
    },
  ],

  const handleAdd = () => {
    const newData: DataType = {
      key: count,
      name: `Edward King ${count}`,
      age: '32',
      address: `London, Park Lane no. ${count}`,
    };
    setDataSource([...dataSource, newData]);
    setCount(count + 1);
  };

  const handleSave = (row: DataType) => {
    const newData = [...dataSource];
    const index = newData.findIndex((item) => row.key === item.key);
    const item = newData[index];
    newData.splice(index, 1, {
      ...item,
      ...row,
    });
    setDataSource(newData);
  };

  const components = {
    body: {
      row: EditableRow,
      cell: EditableCell,
    },
  };

  const columns = defaultColumns.map((col) => {
    if (!col.editable) {
      return col;
    }
    return {
      ...col,
      onCell: (record: DataType) => ({
        record,
        editable: col.editable,

```

```

    dataIndex: col.dataIndex,
    title: col.title,
    handleSave,
  )),
};
});

return (
  <div>
    <Button onClick={handleAdd} type="primary" style={{ marginBottom: 16
  }}>
      Add a row
    </Button>
    <Table<DataType>
      components={components}
      rowClassName={() => 'editable-row'}
      bordered
      dataSource={dataSource}
      columns={columns as ColumnTypes}
    </>
  </div>
);
};

export default App;
```

Editable Rows

```
import React, { useState } from 'react';
import type { TableProps } from 'antd';
import { Form, Input, InputNumber, Popconfirm, Table, Typography } from
'antd';

interface DataType {
  key: string;
  name: string;
  age: number;
  address: string;
}

const originData = Array.from({ length: 100 }).map<DataType>((_, i) => ({
  key: i.toString(),
  name: `Edward ${i}`,
  age: 32,
  address: `London Park no. ${i}`,
```

```

    }));

interface EditableCellProps extends React.HTMLAttributes<HTMLElement> {
  editing: boolean;
  dataIndex: string;
  title: any;
  inputType: 'number' | 'text';
  record: DataType;
  index: number;
}

const EditableCell: React.FC<React.PropsWithChildren<EditableCellProps>> =
({
  editing,
  dataIndex,
  title,
  inputType,
  record,
  index,
  children,
  ...restProps
}) => {
  const inputNode = inputType === 'number' ? <InputNumber /> : <Input />;

  return (
    <td {...restProps}>
      {editing ? (
        <Form.Item
          name={dataIndex}
          style={{ margin: 0 }}
          rules={[
            {
              required: true,
              message: `Please Input ${title}!`,
            },
          ]}
        >
          {inputNode}
        </Form.Item>
      ) : (
        children
      )}
    </td>
  );
};

```

```

const App: React.FC = () => {
  const [form] = Form.useForm();
  const [data, setData] = useState<DataType[]>(originData);
  const [editingKey, setEditingKey] = useState('');

  const isEditing = (record: DataType) => record.key === editingKey;

  const edit = (record: Partial<DataType> & { key: React.Key }) => {
    form.setFieldsValue({ name: '', age: '', address: '', ...record });
    setEditingKey(record.key);
  };

  const cancel = () => {
    setEditingKey('');
  };

  const save = async (key: React.Key) => {
    try {
      const row = (await form.validateFields()) as DataType;

      const newData = [...data];
      const index = newData.findIndex((item) => key === item.key);
      if (index > -1) {
        const item = newData[index];
        newData.splice(index, 1, {
          ...item,
          ...row,
        });
        setData(newData);
        setEditingKey('');
      } else {
        newData.push(row);
        setData(newData);
        setEditingKey('');
      }
    } catch (errInfo) {
      console.log('Validate Failed:', errInfo);
    }
  };

  const columns = [
    {
      title: 'name',
      dataIndex: 'name',
      width: '25%',
      editable: true,
    },
  ],

```

```

    },
    {
      title: 'age',
      dataIndex: 'age',
      width: '15%',
      editable: true,
    },
    {
      title: 'address',
      dataIndex: 'address',
      width: '40%',
      editable: true,
    },
    {
      title: 'operation',
      dataIndex: 'operation',
      render: (_, any, record: DataType) => {
        const editable = isEditing(record);
        return editable ? (
          <span>
            <Typography.Link onClick={() => save(record.key)} style={{
marginInlineEnd: 8 }}>
              Save
            </Typography.Link>
            <Popconfirm title="Sure to cancel?" onConfirm={cancel}>
              <a>Cancel</a>
            </Popconfirm>
          </span>
        ) : (
          <Typography.Link disabled={editingKey !== ''} onClick={() =>
edit(record)}>
            Edit
          </Typography.Link>
        );
      },
    },
  ],
];

const mergedColumns: TableProps<DataType>['columns'] = columns.map((col)
=> {
  if (!col.editable) {
    return col;
  }
  return {
    ...col,
    onCell: (record: DataType) => ({

```

```

        record,
        inputType: col.dataIndex === 'age' ? 'number' : 'text',
        dataIndex: col.dataIndex,
        title: col.title,
        editing: isEditing(record),
      )),
    };
  });

  return (
    <Form form={form} component={false}>
      <Table<DataType>
        components={{
          body: { cell: EditableCell },
        }}
        bordered
        dataSource={data}
        columns={mergedColumns}
        rowClassName="editable-row"
        pagination={{ onChange: cancel }}
      />
    </Form>
  );
};

export default App;

```

Nested tables

```

import React from 'react';
import { DownOutlined } from '@ant-design/icons';
import type { TableColumnsType } from 'antd';
import { Badge, Dropdown, Space, Table } from 'antd';

interface ExpandedDataType {
  key: React.Key;
  date: string;
  name: string;
  upgradeNum: string;
}

interface DataType {
  key: React.Key;
  name: string;
  platform: string;
}

```

```

    version: string;
    upgradeNum: number;
    creator: string;
    createdAt: string;
  }

  const items = [
    { key: '1', label: 'Action 1' },
    { key: '2', label: 'Action 2' },
  ];

  const expandDataSource = Array.from({ length: 3 }).map<ExpandedDataType>
  ((_, i) => ({
    key: i.toString(),
    date: '2014-12-24 23:12:00',
    name: 'This is production name',
    upgradeNum: 'Upgraded: 56',
  }));

  const dataSource = Array.from({ length: 3 }).map<DataType>((_, i) => ({
    key: i.toString(),
    name: 'Screen',
    platform: 'iOS',
    version: '10.3.4.5654',
    upgradeNum: 500,
    creator: 'Jack',
    createdAt: '2014-12-24 23:12:00',
  }));

  const expandColumns: TableColumnsType<ExpandedDataType> = [
    { title: 'Date', dataIndex: 'date', key: 'date' },
    { title: 'Name', dataIndex: 'name', key: 'name' },
    {
      title: 'Status',
      key: 'state',
      render: () => <Badge status="success" text="Finished" />,
    },
    { title: 'Upgrade Status', dataIndex: 'upgradeNum', key: 'upgradeNum' },
    {
      title: 'Action',
      key: 'operation',
      render: () => (
        <Space size="middle">
          <a>Pause</a>
          <a>Stop</a>
          <Dropdown menu={{ items }}>

```

```

        <a>
          More <DownOutlined />
        </a>
      </Dropdown>
    </Space>
  ),
},
];

const columns: TableColumnsType<DataType> = [
  { title: 'Name', dataIndex: 'name', key: 'name' },
  { title: 'Platform', dataIndex: 'platform', key: 'platform' },
  { title: 'Version', dataIndex: 'version', key: 'version' },
  { title: 'Upgraded', dataIndex: 'upgradeNum', key: 'upgradeNum' },
  { title: 'Creator', dataIndex: 'creator', key: 'creator' },
  { title: 'Date', dataIndex: 'createdAt', key: 'createdAt' },
  { title: 'Action', key: 'operation', render: () => <a>Publish</a> },
];

const expandedRowRender = () => (
  <Table<ExpandedDataType>
    columns={expandColumns}
    dataSource={expandDataSource}
    pagination={false}
  />
);

const App: React.FC = () => (
  <>
    <Table<DataType>
      columns={columns}
      expandable={{ expandedRowRender, defaultExpandedRowKeys: ['0'] }}
      dataSource={dataSource}
    />
    <Table<DataType>
      columns={columns}
      expandable={{ expandedRowRender, defaultExpandedRowKeys: ['0'] }}
      dataSource={dataSource}
      size="middle"
    />
    <Table<DataType>
      columns={columns}
      expandable={{ expandedRowRender, defaultExpandedRowKeys: ['0'] }}
      dataSource={dataSource}
      size="small"
    />
  </>
);

```



```

    </>
  );

  export default App;

```

Drag sorting

```

import React, { useState } from 'react';
import type { DragEndEvent } from '@dnd-kit/core';
import { DndContext, PointerSensor, useSensor, useSensors } from '@dnd-kit/core';
import { restrictToVerticalAxis } from '@dnd-kit/modifiers';
import {
  arrayMove,
  SortableContext,
  useSortable,
  verticalListSortingStrategy,
} from '@dnd-kit/sortable';
import { CSS } from '@dnd-kit/utilities';
import { Table } from 'antd';
import type { TableColumnsType } from 'antd';

interface DataType {
  key: string;
  name: string;
  age: number;
  address: string;
}

const columns: TableColumnsType<DataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
  },
  {
    title: 'Age',
    dataIndex: 'age',
  },
  {
    title: 'Address',
    dataIndex: 'address',
  },
];

interface RowProps extends React.HTMLAttributes<HTMLTableRowElement> {
  'data-row-key': string;

```

```

}

const Row: React.FC<Readonly<RowProps>> = (props) => {
  const { attributes, listeners, setNodeRef, transform, transition,
isDragging } = useSortable({
    id: props['data-row-key'],
  });

  const style: React.CSSProperties = {
    ...props.style,
    transform: CSS.Translate.toString(transform),
    transition,
    cursor: 'move',
    ...(isDragging ? { position: 'relative', zIndex: 9999 } : {}),
  };

  return <tr {...props} ref={setNodeRef} style={style} {...attributes}
{...listeners} />;
};

const App: React.FC = () => {
  const [dataSource, setDataSource] = useState([
    {
      key: '1',
      name: 'John Brown',
      age: 32,
      address:
        'Long text Long text Long text Long text Long text Long text Long
text Long text Long text Long text Long text Long text Long text Long text
Long text Long text Long text Long text Long text Long text Long text Long
text Long text Long text Long text Long text Long text Long text Long text
Long text Long text Long text',
    },
    {
      key: '2',
      name: 'Jim Green',
      age: 42,
      address: 'London No. 1 Lake Park',
    },
    {
      key: '3',
      name: 'Joe Black',
      age: 32,
      address: 'Sidney No. 1 Lake Park',
    },
  ]);
};

```

```

const sensors = useSensors(
  useSensor(PointerSensor, {
    activationConstraint: {
      // https://docs.dndkit.com/api-
documentation/sensors/pointer#activation-constraints
      distance: 1,
    },
  }),
);

const onDragEnd = ({ active, over }: DragEndEvent) => {
  if (active.id !== over?.id) {
    setDataSource((prev) => {
      const activeIndex = prev.findIndex((i) => i.key === active.id);
      const overIndex = prev.findIndex((i) => i.key === over?.id);
      return arrayMove(prev, activeIndex, overIndex);
    });
  }
};

return (
  <DndContext sensors={sensors} modifiers={[restrictToVerticalAxis]}
onDragEnd={onDragEnd}>
    <SortableContext
      // rowKey array
      items={dataSource.map((i) => i.key)}
      strategy={verticalListSortingStrategy}
    >
      <Table<DataType>
        components={{
          body: { row: Row },
        }}
        rowKey="key"
        columns={columns}
        dataSource={dataSource}
      />
    </SortableContext>
  </DndContext>
);
};

export default App;

```

Drag Column sorting

```

import React, { createContext, useContext, useState } from 'react';
import type { DragEndEvent, DragOverEvent, UniqueIdentifier } from '@dnd-kit/core';
import {
  closestCenter,
  DndContext,
  DragOverlay,
  PointerSensor,
  useSensor,
  useSensors,
} from '@dnd-kit/core';
import { restrictToHorizontalAxis } from '@dnd-kit/modifiers';
import {
  arrayMove,
  horizontalListSortingStrategy,
  SortableContext,
  useSortable,
} from '@dnd-kit/sortable';
import { Table } from 'antd';
import type { TableColumnsType } from 'antd';

interface DataType {
  key: string;
  name: string;
  gender: string;
  age: number;
  email: string;
  address: string;
}

interface HeaderCellProps extends React.HTMLAttributes<HTMLTableCellElement> {
  id: string;
}

interface BodyCellProps extends React.HTMLAttributes<HTMLTableCellElement> {
  id: string;
}

interface DragIndexState {
  active: UniqueIdentifier;
  over: UniqueIdentifier | undefined;
  direction?: 'left' | 'right';
}

```

```

const DragIndexContext = createContext<DragIndexState>({ active: -1, over:
-1 });

const dragActiveStyle = (dragState: DragIndexState, id: string) => {
  const { active, over, direction } = dragState;
  // drag active style
  let style: React.CSSProperties = {};
  if (active && active === id) {
    style = { backgroundColor: 'gray', opacity: 0.5 };
  }
  // dragover dashed style
  else if (over && id === over && active !== over) {
    style =
      direction === 'right'
        ? { borderRight: '1px dashed gray' }
        : { borderLeft: '1px dashed gray' };
  }
  return style;
};

const TableBodyCell: React.FC<BodyCellProps> = (props) => {
  const dragState = useContext<DragIndexState>(DragIndexContext);
  return <td {...props} style={{ ...props.style,
...dragActiveStyle(dragState, props.id) }} />;
};

const TableHeaderCell: React.FC<HeaderCellProps> = (props) => {
  const dragState = useContext(DragIndexContext);
  const { attributes, listeners, setNodeRef, isDragging } = useSortable({
id: props.id });
  const style: React.CSSProperties = {
    ...props.style,
    cursor: 'move',
    ...(isDragging ? { position: 'relative', zIndex: 9999, userSelect:
'none' } : {}),
    ...dragActiveStyle(dragState, props.id),
  };
  return <th {...props} ref={setNodeRef} style={style} {...attributes}
{...listeners} />;
};

const dataSource: DataType[] = [
  {
    key: '1',
    name: 'John Brown',
    gender: 'male',
  }
];

```

```

    age: 32,
    email: 'John Brown@example.com',
    address: 'London No. 1 Lake Park',
  },
  {
    key: '2',
    name: 'Jim Green',
    gender: 'female',
    age: 42,
    email: 'jimGreen@example.com',
    address: 'London No. 1 Lake Park',
  },
  {
    key: '3',
    name: 'Joe Black',
    gender: 'female',
    age: 32,
    email: 'JoeBlack@example.com',
    address: 'Sidney No. 1 Lake Park',
  },
  {
    key: '4',
    name: 'George Hcc',
    gender: 'male',
    age: 20,
    email: 'george@example.com',
    address: 'Sidney No. 1 Lake Park',
  },
];

```

```

const baseColumns: TableColumnType<DataType> = [
  { title: 'Name', dataIndex: 'name' },
  { title: 'Gender', dataIndex: 'gender' },
  { title: 'Age', dataIndex: 'age' },
  { title: 'Email', dataIndex: 'email' },
  { title: 'Address', dataIndex: 'address' },
];

```

```

const App: React.FC = () => {
  const [dragIndex, setDragIndex] = useState<DragIndexState>({ active: -1,
over: -1 });

  const [columns, setColumns] = useState(() =>
    baseColumns.map((column, i) => ({
      ...column,
      key: ` ${i}`,

```

```

        onHeaderCell: () => ({ id: `${i}` }),
        onCell: () => ({ id: `${i}` }),
    })),
);

const sensors = useSensors(
    useSensor(PointerSensor, {
        activationConstraint: {
            // https://docs.dndkit.com/api-
documentation/sensors/pointer#activation-constraints
            distance: 1,
        },
    }),
);

const onDragEnd = ({ active, over }: DragEndEvent) => {
    if (active.id !== over?.id) {
        setColumns((prevState) => {
            const activeIndex = prevState.findIndex((i) => i.key ===
active?.id);
            const overIndex = prevState.findIndex((i) => i.key === over?.id);
            return arrayMove(prevState, activeIndex, overIndex);
        });
    }
    setDragIndex({ active: -1, over: -1 });
};

const onDragOver = ({ active, over }: DragOverEvent) => {
    const activeIndex = columns.findIndex((i) => i.key === active.id);
    const overIndex = columns.findIndex((i) => i.key === over?.id);
    setDragIndex({
        active: active.id,
        over: over?.id,
        direction: overIndex > activeIndex ? 'right' : 'left',
    });
};

return (
    <DndContext
        sensors={sensors}
        modifiers={[restrictToHorizontalAxis]}
        onDragEnd={onDragEnd}
        onDragOver={onDragOver}
        collisionDetection={closestCenter}
    >
        <SortableContext items={columns.map((i) => i.key)} strategy=

```

```

{horizontalListSortingStrategy}>
  <DragIndexContext.Provider value={dragIndex}>
    <Table<DataType>
      rowKey="key"
      columns={columns}
      dataSource={dataSource}
      components={{
        header: { cell: TableHeaderCell },
        body: { cell: TableBodyCell },
      }}
    />
  </DragIndexContext.Provider>
</SortableContext>
<DragOverlay>
  <th style={{ backgroundColor: 'gray', padding: 16 }}>
    {columns[columns.findIndex((i) => i.key ===
dragIndex.active)]?.title as React.ReactNode}
  </th>
</DragOverlay>
</DndContext>
);
};

export default App;

```

Drag sorting with handler

```

import React, { useContext, useMemo } from 'react';
import { HolderOutlined } from '@ant-design/icons';
import type { DragEndEvent } from '@dnd-kit/core';
import { DndContext } from '@dnd-kit/core';
import type { SyntheticListenerMap } from '@dnd-kit/core/dist/hooks/utilities';
import { restrictToVerticalAxis } from '@dnd-kit/modifiers';
import {
  arrayMove,
  SortableContext,
  useSortable,
  verticalListSortingStrategy,
} from '@dnd-kit/sortable';
import { CSS } from '@dnd-kit/utilities';
import { Button, Table } from 'antd';
import type { TableColumnsType } from 'antd';

interface DataType {

```



```

    key: string;
    name: string;
    age: number;
    address: string;
  }

  interface RowContextProps {
    setActivatorNodeRef?: (element: HTMLElement | null) => void;
    listeners?: SyntheticListenerMap;
  }

  const RowContext = React.createContext<RowContextProps>({});

  const DragHandle: React.FC = () => {
    const { setActivatorNodeRef, listeners } = useContext(RowContext);
    return (
      <Button
        type="text"
        size="small"
        icon={<HolderOutlined />}
        style={{ cursor: 'move' }}
        ref={setActivatorNodeRef}
        {...listeners}
      />
    );
  };

  const columns: TableColumnsType<DataType> = [
    { key: 'sort', align: 'center', width: 80, render: () => <DragHandle /> },
    { title: 'Name', dataIndex: 'name' },
    { title: 'Age', dataIndex: 'age' },
    { title: 'Address', dataIndex: 'address' },
  ];

  const initialData: DataType[] = [
    { key: '1', name: 'John Brown', age: 32, address: 'Long text Long' },
    { key: '2', name: 'Jim Green', age: 42, address: 'London No. 1 Lake Park' },
    { key: '3', name: 'Joe Black', age: 32, address: 'Sidney No. 1 Lake Park' },
  ];

  interface RowProps extends React.HTMLAttributes<HTMLTableRowElement> {
    'data-row-key': string;
  }

```

```

const Row: React.FC<RowProps> = (props) => {
  const {
    attributes,
    listeners,
    setNodeRef,
    setActivatorNodeRef,
    transform,
    transition,
    isDragging,
  } = useSortable({ id: props['data-row-key'] });

  const style: React.CSSProperties = {
    ...props.style,
    transform: CSS.Translate.toString(transform),
    transition,
    ...(isDragging ? { position: 'relative', zIndex: 999 } : {}),
  };

  const contextValue = useMemo<RowContextProps>(
    () => ({ setActivatorNodeRef, listeners }),
    [setActivatorNodeRef, listeners],
  );

  return (
    <RowContext.Provider value={contextValue}>
      <tr {...props} ref={setNodeRef} style={style} {...attributes} />
    </RowContext.Provider>
  );
};

const App: React.FC = () => {
  const [dataSource, setDataSource] = React.useState<DataType[]>(
    initialData);

  const onDragEnd = ({ active, over }: DragEndEvent) => {
    if (active.id !== over?.id) {
      setDataSource((prevState) => {
        const activeIndex = prevState.findIndex((record) => record.key ===
active?.id);
        const overIndex = prevState.findIndex((record) => record.key ===
over?.id);
        return arrayMove(prevState, activeIndex, overIndex);
      });
    }
  };
};

```

```

    return (
      <DndContext modifiers={[restrictToVerticalAxis]} onDragEnd={onDragEnd}>
        <SortableContext items={dataSource.map((i) => i.key)} strategy=
{verticalListSortingStrategy}>
          <Table<DataType>
            rowKey="key"
            components={{ body: { row: Row } }}
            columns={columns}
            dataSource={dataSource}
          />
        </SortableContext>
      </DndContext>
    );
  };

export default App;

```

Resizable column

Debug

```

import React, { useState } from 'react';
import { Table } from 'antd';
import type { TableColumnsType } from 'antd';
import type { ResizeCallbackData } from 'react-resizable';
import { Resizable } from 'react-resizable';

interface DataType {
  key: React.Key;
  date: string;
  amount: number;
  type: string;
  note: string;
}

interface TitlePropsType {
  width: number;
  onResize: (e: React.SyntheticEvent<Element>, data: ResizeCallbackData) =>
void;
}

const ResizableTitle: React.FC<Readonly<React.HTMLAttributes<any> &
TitlePropsType>> = (props) => {
  const { onResize, width, ...restProps } = props;

```

```

    if (!width) {
      return <th {...restProps} />;
    }

    return (
      <Resizable
        width={width}
        height={0}
        handle={<span className="react-resizable-handle" onClick={(e) =>
e.stopPropagation()} />}
        onResize={onResize}
        draggableOpts={{ enableUserSelectHack: false }}
      >
        <th {...restProps} />
      </Resizable>
    );
  };
};

const data: DataType[] = [
  {
    key: 0,
    date: '2018-02-11',
    amount: 120,
    type: 'income',
    note: 'transfer',
  },
  {
    key: 1,
    date: '2018-03-11',
    amount: 243,
    type: 'income',
    note: 'transfer',
  },
  {
    key: 2,
    date: '2018-04-11',
    amount: 98,
    type: 'income',
    note: 'transfer',
  },
];

const App: React.FC = () => {
  const [columns, setColumns] = useState<TableColumnsType<DataType>>([
    {

```

```

        title: 'Date',
        dataIndex: 'date',
        width: 200,
    },
    {
        title: 'Amount',
        dataIndex: 'amount',
        width: 100,
        sorter: (a, b) => a.amount - b.amount,
    },
    {
        title: 'Type',
        dataIndex: 'type',
        width: 100,
    },
    {
        title: 'Note',
        dataIndex: 'note',
        width: 100,
    },
    {
        title: 'Action',
        key: 'action',
        render: () => <a>Delete</a>,
    },
  ],
);

const handleResize =
  (index: number) =>
  (_, { size }: ResizeCallbackData) => {
    const newColumns = [...columns];
    newColumns[index] = {
      ...newColumns[index],
      width: size.width,
    };
    setColumns(newColumns);
  };

const mergedColumns = columns.map<TableColumnsType<DataType>[number]>
((col, index) => ({
  ...col,
  onHeaderCell: (column: TableColumnsType<DataType>[number]) => ({
    width: column.width,
    onResize: handleResize(index) as React.ReactEventHandler<any>,
  }),
})));

```

```

return (
  <Table<DataType>
    bordered
    components={{ header: { cell: ResizableTitle } }}
    columns={mergedColumns}
    dataSource={data}
  />
);
};

export default App;

```

ellipsis column

```

import React from 'react';
import { Table } from 'antd';
import type { TableColumnsType } from 'antd';

interface DataType {
  key: React.Key;
  name: string;
  age: number;
  address: string;
}

const columns: TableColumnsType<DataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
    key: 'name',
    render: (text) => <a>{text}</a>,
    width: 150,
  },
  {
    title: 'Age',
    dataIndex: 'age',
    key: 'age',
    width: 80,
  },
  {
    title: 'Address',
    dataIndex: 'address',
    key: 'address 1',
    ellipsis: true,
  },
];

```

```

    {
      title: 'Long Column Long Column Long Column',
      dataIndex: 'address',
      key: 'address 2',
      ellipsis: true,
    },
    {
      title: 'Long Column Long Column',
      dataIndex: 'address',
      key: 'address 3',
      ellipsis: true,
    },
    {
      title: 'Long Column',
      dataIndex: 'address',
      key: 'address 4',
      ellipsis: true,
    },
  ],
];

const data = [
  {
    key: '1',
    name: 'John Brown',
    age: 32,
    address: 'New York No. 1 Lake Park, New York No. 1 Lake Park',
    tags: ['nice', 'developer'],
  },
  {
    key: '2',
    name: 'Jim Green',
    age: 42,
    address: 'London No. 2 Lake Park, London No. 2 Lake Park',
    tags: ['loser'],
  },
  {
    key: '3',
    name: 'Joe Black',
    age: 32,
    address: 'Sydney No. 1 Lake Park, Sydney No. 1 Lake Park',
    tags: ['cool', 'teacher'],
  },
];

const App: React.FC = () => <Table<DataType> columns={columns} dataSource=
{data} />;

```

```
export default App;
```

ellipsis column custom tooltip

```
import React from 'react';
import { Table, Tooltip } from 'antd';
import type { TableColumnsType } from 'antd';

interface DataType {
  key: React.Key;
  name: string;
  age: number;
  address: string;
}

const columns: TableColumnsType<DataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
    key: 'name',
    render: (text) => <a>{text}</a>,
    width: 150,
  },
  {
    title: 'Age',
    dataIndex: 'age',
    key: 'age',
    width: 80,
  },
  {
    title: 'Address',
    dataIndex: 'address',
    key: 'address 1',
    ellipsis: {
      showTitle: false,
    },
    render: (address) => (
      <Tooltip placement="topLeft" title={address}>
        {address}
      </Tooltip>
    ),
  },
  {
    title: 'Long Column Long Column Long Column',
    dataIndex: 'address',
```



```

    key: 'address 2',
    ellipsis: {
      showTitle: false,
    },
    render: (address) => (
      <Tooltip placement="topLeft" title={address}>
        {address}
      </Tooltip>
    ),
  },
  {
    title: 'Long Column Long Column',
    dataIndex: 'address',
    key: 'address 3',
    ellipsis: {
      showTitle: false,
    },
    render: (address) => (
      <Tooltip placement="topLeft" title={address}>
        {address}
      </Tooltip>
    ),
  },
  {
    title: 'Long Column',
    dataIndex: 'address',
    key: 'address 4',
    ellipsis: {
      showTitle: false,
    },
    render: (address) => (
      <Tooltip placement="topLeft" title={address}>
        {address}
      </Tooltip>
    ),
  },
];

const data: DataType[] = [
  {
    key: '1',
    name: 'John Brown',
    age: 32,
    address: 'New York No. 1 Lake Park, New York No. 1 Lake Park',
  },
  {

```

```

    key: '2',
    name: 'Jim Green',
    age: 42,
    address: 'London No. 2 Lake Park, London No. 2 Lake Park',
  },
  {
    key: '3',
    name: 'Joe Black',
    age: 32,
    address: 'Sydney No. 1 Lake Park, Sydney No. 1 Lake Park',
  },
];

const App: React.FC = () => <Table<DataType> columns={columns} dataSource=
{data} />;

export default App;

```

Custom empty

```

import React, { useState } from 'react';
import type { GetProp } from 'antd';
import { Button, ConfigProvider, Empty, Table } from 'antd';

interface DataType {
  key: number;
  name: string;
  age: number;
  address: string;
}

const genFakeData = (count = 5) =>
  Array.from({ length: count }).map<DataType>((_, index) => ({
    key: index,
    name: `Edward King ${index}`,
    age: 32 + index,
    address: `London, Park Lane no. ${index}`,
  }));

const renderEmpty: GetProp<typeof ConfigProvider, 'renderEmpty'> =
(componentName) => {
  if (componentName === 'Table.filter' /** 🐞 5.20.0+ */) {
    return <Empty image={Empty.PRESENTED_IMAGE_SIMPLE} description="No
Filter(Custom)" />;
  }
};

```

```

function App() {
  const [dataSource, setDataSource] = useState<DataType[]>(genFakeData);

  const handleToggle = () => {
    setDataSource(dataSource.length ? [] :
genFakeData(Math.floor(Math.random() * 10)));
  };

  const columns: GetProp<typeof Table<DataType>, 'columns'> = [
    {
      title: 'Name',
      dataIndex: 'name',
      key: 'name',
    },
    {
      title: 'Age',
      dataIndex: 'age',
      key: 'age',
      filters: dataSource.length
        ? [
            { text: '>=35', value: 'gte35' },
            { text: '<18', value: 'lt18' },
          ]
        : [],
    },
    {
      title: 'Address',
      dataIndex: 'address',
      key: 'address',
    },
  ],
];

  const toggleButton = (
    <Button type="primary" onClick={handleToggle}>
      Toggle Data
    </Button>
  );

  return (
    <ConfigProvider renderEmpty={renderEmpty}>
      {dataSource.length ? toggleButton : null}
      <div style={{ marginBlock: 8 }} />
      <Table<DataType>
        bordered
        dataSource={dataSource}

```

```

        columns={columns}
        locale={{ emptyText: <Empty description="No Data">{toggleButton}
</Empty> }}
      />
    </ConfigProvider>
  );
}

export default App;

```

Summary

```

import React from 'react';
import { Flex, Table, Typography } from 'antd';
import type { TableColumnsType } from 'antd';
import { createStyles } from 'antd-style';

const useStyles = createStyles(({ css, token }) => {
  const { antCls } = token;
  return {
    customTable: css`
      ${antCls}-table {
        ${antCls}-table-container {
          ${antCls}-table-body,
          ${antCls}-table-content {
            scrollbar-width: thin;
            scrollbar-color: #eaeaea transparent;
            scrollbar-gutter: stable;
          }
        }
      }
    `,
  };
});

const { Text } = Typography;

interface DataType {
  key: string;
  name: string;
  borrow: number;
  repayment: number;
}

interface FixedDataType {

```

```
    key: React.Key;
    name: string;
    description: string;
  }

const columns: TableColumnsType<DataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
  },
  {
    title: 'Borrow',
    dataIndex: 'borrow',
  },
  {
    title: 'Repayment',
    dataIndex: 'repayment',
  },
];

const dataSource: DataType[] = [
  {
    key: '1',
    name: 'John Brown',
    borrow: 10,
    repayment: 33,
  },
  {
    key: '2',
    name: 'Jim Green',
    borrow: 100,
    repayment: 0,
  },
  {
    key: '3',
    name: 'Joe Black',
    borrow: 10,
    repayment: 10,
  },
  {
    key: '4',
    name: 'Jim Red',
    borrow: 75,
    repayment: 45,
  },
];
```

```

const fixedColumns: TableColumnsType<FixedDataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
    fixed: true,
    width: 100,
  },
  {
    title: 'Description',
    dataIndex: 'description',
  },
];

const fixedDataSource = Array.from({ length: 20 }).map<FixedDataType>((_,
i) => ({
  key: i,
  name: ['Light', 'Bamboo', 'Little'][i % 3],
  description: 'Everything that has a beginning, has an end.',
}));

const App: React.FC = () => {
  const { styles } = useStyles();
  return (
    <Flex vertical gap="small">
      <Table<DataType>
        bordered
        className={styles.customTable}
        columns={columns}
        dataSource={dataSource}
        pagination={false}
        summary={(pageData) => {
          let totalBorrow = 0;
          let totalRepayment = 0;
          pageData.forEach(({ borrow, repayment }) => {
            totalBorrow += borrow;
            totalRepayment += repayment;
          });
          return (
            <>
              <Table.Summary.Row>
                <Table.Summary.Cell index={0}>Total</Table.Summary.Cell>
                <Table.Summary.Cell index={1}>
                  <Text type="danger">{totalBorrow}</Text>
                </Table.Summary.Cell>
                <Table.Summary.Cell index={2}>

```

```

        <Text>{totalRepayment}</Text>
      </Table.Summary.Cell>
    </Table.Summary.Row>
    <Table.Summary.Row>
      <Table.Summary.Cell index={0}>Balance</Table.Summary.Cell>
      <Table.Summary.Cell index={1} colSpan={2}>
        <Text type="danger">{totalBorrow - totalRepayment}</Text>
      </Table.Summary.Cell>
    </Table.Summary.Row>
  </>
);
}}
/>
<Table<FixedDataType>
  className={styles.customTable}
  columns={fixedColumns}
  dataSource={fixedDataSource}
  pagination={false}
  scroll={{ x: 2000, y: 500 }}
  bordered
  summary={() => (
    <Table.Summary fixed>
      <Table.Summary.Row>
        <Table.Summary.Cell index={0}>Summary</Table.Summary.Cell>
        <Table.Summary.Cell index={1}>This is a summary
content</Table.Summary.Cell>
      </Table.Summary.Row>
    </Table.Summary>
  )}
/>
</Flex>
);
};

export default App;

```

Virtual list

v5.9.0

```

import React from 'react';
import { Button, Segmented, Space, Switch, Table, Typography } from 'antd';
import type { TableProps } from 'antd';

interface RecordType {

```

```

    id: number;
    firstName: string;
    lastName: string;
    age: number;
    address1: string;
    address2: string;
    address3: string;
  }

  const fixedColumns: TableProps<RecordType>['columns'] = [
    {
      title: 'ID',
      dataIndex: 'id',
      width: 100,
      fixed: 'left',
    },
    {
      title: 'FistName',
      dataIndex: 'firstName',
      width: 120,
      fixed: 'left',
    },
    {
      title: 'LastName',
      dataIndex: 'lastName',
      width: 120,
      fixed: 'left',
    },
    {
      title: 'Group',
      width: 120,
      render: (_, record) => `Group ${Math.floor(record.id / 4)}\`,
      onCell: (record) => ({
        rowspan: record.id % 4 === 0 ? 4 : 0,
      }),
    },
    {
      title: 'Age',
      dataIndex: 'age',
      width: 100,
      onCell: (record) => ({
        colspan: record.id % 4 === 0 ? 2 : 1,
      }),
    },
    {
      title: 'Address 1',

```



```

        dataIndex: 'address1',
        onCell: (record) => ({
            colspan: record.id % 4 === 0 ? 0 : 1,
        }),
    },
    {
        title: 'Address 2',
        dataIndex: 'address2',
    },
    {
        title: 'Address 3',
        dataIndex: 'address3',
    },
    {
        title: 'Action',
        width: 150,
        fixed: 'right',
        render: () => (
            <Space>
                <Typography.Link>Action1</Typography.Link>
                <Typography.Link>Action2</Typography.Link>
            </Space>
        ),
    },
];

const columns: TableProps<RecordType>['columns'] = [
    {
        title: 'ID',
        dataIndex: 'id',
        width: 100,
    },
    {
        title: 'FistName',
        dataIndex: 'firstName',
        width: 120,
    },
    {
        title: 'LastName',
        dataIndex: 'lastName',
    },
];

const getData = (length: number) =>
    Array.from({ length }).map<RecordType>((_, index) => ({
        id: index,
    }));

```

```

    firstName: `First_${index.toString(16)}`,
    lastName: `Last_${index.toString(16)}`,
    age: 25 + (index % 10),
    address1: `New York No. ${index} Lake Park`,
    address2: `London No. ${index} Lake Park`,
    address3: `Sydney No. ${index} Lake Park`,
  }));

const App: React.FC = () => {
  const [fixed, setFixed] = React.useState(true);
  const [bordered, setBordered] = React.useState(true);
  const [expanded, setExpanded] = React.useState(false);
  const [empty, setEmpty] = React.useState(false);
  const [count, setCount] = React.useState(10000);

  const tblRef: Parameters<typeof Table>[0]['ref'] = React.useRef(null);

  const data = React.useMemo<RecordType[]>(() => getData(count), [count]);

  const mergedColumns = React.useMemo<typeof fixedColumns>(() => {
    if (!fixed) {
      return columns;
    }

    if (!expanded) {
      return fixedColumns;
    }

    return fixedColumns.map((col) => ({ ...col, onCell: undefined }));
  }, [expanded, fixed]);

  const expandableProps = React.useMemo<TableProps<RecordType>
['expandable']>(() => {
    if (!expanded) {
      return undefined;
    }

    return {
      columnWidth: 48,
      expandedRowRender: (record) => <p style={{ margin: 0 }}>👉 Expanded
{record.address1}</p>,
      rowExpandable: (record) => record.id % 2 === 0,
    };
  }, [expanded]);

  return (

```

```

<div style={{ padding: 64 }}>
  <Space direction="vertical" style={{ width: '100%' }}>
    <Space>
      <Switch
        checked={bordered}
        onChange={() => setBordered(!bordered)}
        checkedChildren="Bordered"
        uncheckedChildren="Bordered"
      />
      <Switch
        checked={fixed}
        onChange={() => setFixed(!fixed)}
        checkedChildren="Fixed"
        uncheckedChildren="Fixed"
      />
      <Switch
        checked={expanded}
        onChange={() => setExpanded(!expanded)}
        checkedChildren="Expandable"
        uncheckedChildren="Expandable"
      />
      <Switch
        checked={empty}
        onChange={() => setEmpty(!empty)}
        checkedChildren="Empty"
        uncheckedChildren="Empty"
      />
      <Segmented
        value={count}
        onChange={setCount}
        options={[
          { label: 'None', value: 0 },
          { label: 'Less', value: 4 },
          { label: 'Lot', value: 10000 },
        ]}
      />

      {data.length >= 999 && (
        <Button onClick={() => tblRef.current?.scrollTo({ index: 999
      ))}>

        Scroll To index 999
      </Button>
    )}
  </Space>

  <Table<RecordType>

```

```

        bordered={bordered}
        virtual
        columns={mergedColumns}
        scroll={{ x: 2000, y: 400 }}
        rowKey="id"
        dataSource={empty ? [] : data}
        pagination={false}
        ref={tblRef}
        rowSelection={expanded ? undefined : { type: 'radio',
columnWidth: 48 }}
        expandable={expandableProps}
      />
    </Space>
  </div>
);
};

export default App;

```

Responsive

```

import React from 'react';
import { Table } from 'antd';
import type { TableColumnsType } from 'antd';

interface DataType {
  key: React.Key;
  name: string;
  age: number;
  address: string;
}

const columns: TableColumnsType<DataType> = [
  {
    title: 'Name (all screens)',
    dataIndex: 'name',
    key: 'name',
    render: (text) => <a>{text}</a>,
  },
  {
    title: 'Age (medium screen or bigger)',
    dataIndex: 'age',
    key: 'age',
    responsive: ['md'],
  },
];

```

```

    {
      title: 'Address (large screen or bigger)',
      dataIndex: 'address',
      key: 'address',
      responsive: ['lg'],
    },
  ],
];

const data: DataType[] = [
  {
    key: '1',
    name: 'John Brown',
    age: 32,
    address: 'New York No. 1 Lake Park',
  },
];

const App: React.FC = () => <Table<DataType> columns={columns} dataSource=
{data} />;

export default App;

```

Nested Bordered Table Debug

Debug

```

import React, { useState } from 'react';
import { DownOutlined } from '@ant-design/icons';
import type { TableColumnsType, TableProps } from 'antd';
import { Badge, Dropdown, Form, Space, Switch, Table } from 'antd';

interface DataType {
  key: React.Key;
  name: string;
  platform: string;
  version: string;
  upgradeNum: number;
  creator: string;
  createdAt: string;
}

interface ExpandedDataType {
  key: React.Key;
  date: string;
  name: string;
  upgradeNum: string;
}

```

```

}

const items = [
  { key: '1', label: 'Action 1' },
  { key: '2', label: 'Action 2' },
];

const expandedColumns: TableProps<ExpandedDataType>['columns'] = [
  { title: 'Date', dataIndex: 'date', key: 'date' },
  { title: 'Name', dataIndex: 'name', key: 'name' },
  {
    title: 'Status',
    key: 'state',
    render: () => (
      <span>
        <Badge status="success" />
        Finished
      </span>
    ),
  },
  { title: 'Upgrade Status', dataIndex: 'upgradeNum', key: 'upgradeNum' },
  {
    title: 'Action',
    dataIndex: 'operation',
    key: 'operation',
    render: () => (
      <Space size="middle">
        <a>Pause</a>
        <a>Stop</a>
        <Dropdown menu={{ items }}>
          <a>
            More <DownOutlined />
          </a>
        </Dropdown>
      </Space>
    ),
  },
];

const expandedDataSource = Array.from({ length: 3 }).map<ExpandedDataType>(
  (_, i) => ({
    key: i,
    date: '2014-12-24 23:12:00',
    name: 'This is production name',
    upgradeNum: 'Upgraded: 56',
  }));

```

```

const createExpandedRowRender = (bordered: boolean) => () => (
  <Table<ExpandedDataType>
    columns={expandedColumns}
    dataSource={expandedDataSource}
    pagination={false}
    bordered={bordered}
  />
);

const columns: TableColumnsType<DataType> = [
  { title: 'Name', dataIndex: 'name', key: 'name' },
  { title: 'Platform', dataIndex: 'platform', key: 'platform' },
  { title: 'Version', dataIndex: 'version', key: 'version' },
  { title: 'Upgraded', dataIndex: 'upgradeNum', key: 'upgradeNum' },
  { title: 'Creator', dataIndex: 'creator', key: 'creator' },
  { title: 'Date', dataIndex: 'createdAt', key: 'createdAt' },
  { title: 'Action', key: 'operation', render: () => <a>Publish</a> },
];

const dataSource = Array.from({ length: 3 }).map<DataType>((_, i) => ({
  key: i,
  name: 'Screem',
  platform: 'iOS',
  version: '10.3.4.5654',
  upgradeNum: 500,
  creator: 'Jack',
  createdAt: '2014-12-24 23:12:00',
})));

const App: React.FC = () => {
  const [rootTableBordered, setRootTableBordered] = useState(true);
  const [childTableBordered, setChildTableBordered] = useState(true);
  return (
    <>
      <Form layout="inline" className="table-demo-control-bar" style={{
marginBottom: 16 }}>
        <Form.Item label="Root Table Bordered">
          <Switch checked={rootTableBordered} onChange={(v) =>
setRootTableBordered(v)} />
        </Form.Item>
        <Form.Item label="Child Table Bordered">
          <Switch checked={childTableBordered} onChange={(v) =>
setChildTableBordered(v)} />
        </Form.Item>
      </Form>
    </>
  );
}

```

```

    <Table<DataType>
      title={() => 'cool'}
      footer={() => 'cool'}
      columns={columns}
      expandable={{ expandedRowRender:
createExpandedRowRender(childTableBordered) }}
      dataSource={dataSource}
      bordered={rootTableBordered}
    />
  </>
);
};

export default App;

```

Pagination Settings

```

import React, { useState } from 'react';
import { Radio, Space, Table, Tag } from 'antd';
import type { TableProps } from 'antd';

type ColumnsType<T extends object> = TableProps<T>['columns'];
type TablePagination<T extends object> = NonNullable<Exclude<TableProps<T>
['pagination'], boolean>>;
type TablePaginationPosition<T extends object> = NonNullable<
  TablePagination<T>['position']
>[number];

interface DataType {
  key: string;
  name: string;
  age: number;
  address: string;
  tags: string[];
}

const topOptions = [
  { label: 'topLeft', value: 'topLeft' },
  { label: 'topCenter', value: 'topCenter' },
  { label: 'topRight', value: 'topRight' },
  { label: 'none', value: 'none' },
];

const bottomOptions = [
  { label: 'bottomLeft', value: 'bottomLeft' },

```



```

    { label: 'bottomCenter', value: 'bottomCenter' },
    { label: 'bottomRight', value: 'bottomRight' },
    { label: 'none', value: 'none' },
  ];

const columns: ColumnsType<DataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
    key: 'name',
    render: (text) => <a>{text}</a>,
  },
  {
    title: 'Age',
    dataIndex: 'age',
    key: 'age',
  },
  {
    title: 'Address',
    dataIndex: 'address',
    key: 'address',
  },
  {
    title: 'Tags',
    key: 'tags',
    dataIndex: 'tags',
    render: (tags: string[]) => (
      <span>
        {tags.map((tag) => {
          let color = tag.length > 5 ? 'geekblue' : 'green';
          if (tag === 'loser') {
            color = 'volcano';
          }
          return (
            <Tag color={color} key={tag}>
              {tag.toUpperCase()}
            </Tag>
          );
        })}
      </span>
    ),
  },
  {
    title: 'Action',
    key: 'action',
    render: (_, record) => (

```

```

        <Space size="middle">
          <a>Invite {record.name}</a>
          <a>Delete</a>
        </Space>
      ),
    },
  ],
];

```

```

const data: DataType[] = [
  {
    key: '1',
    name: 'John Brown',
    age: 32,
    address: 'New York No. 1 Lake Park',
    tags: ['nice', 'developer'],
  },
  {
    key: '2',
    name: 'Jim Green',
    age: 42,
    address: 'London No. 1 Lake Park',
    tags: ['loser'],
  },
  {
    key: '3',
    name: 'Joe Black',
    age: 32,
    address: 'Sydney No. 1 Lake Park',
    tags: ['cool', 'teacher'],
  },
];

```

```

const App: React.FC = () => {
  const [top, setTop] = useState<TablePaginationPosition<DataType>>
('topLeft');
  const [bottom, setBottom] = useState<TablePaginationPosition<DataType>>
('bottomRight');
  return (
    <div>
      <div>
        <Radio.Group
          style={{ marginBottom: 10 }}
          options={topOptions}
          value={top}
          onChange={(e) => {
            setTop(e.target.value);

```

```

        }}
      />
    </div>
    <Radio.Group
      style={{ marginBottom: 10 }}
      options={bottomOptions}
      value={bottom}
      onChange={(e) => {
        setBottom(e.target.value);
      }}
    />
    <Table<DataType>
      columns={columns}
      pagination={{ position: [top, bottom] }}
      dataSource={data}
    />
  </div>
);
};

export default App;

```

Custom selection group

Debug

```

import React from 'react';
import { Table } from 'antd';
import type { TableColumnsType, TableProps } from 'antd';

type TableRowSelection<T extends object = object> = TableProps<T>
['rowSelection'];

interface DataType {
  key: React.Key;
  name: string;
}

const columns: TableColumnsType<DataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
  },
];

```

```

const dataSource = Array.from({ length: 46 }).map<DataType>((_, i) => ({
  key: i,
  name: i % 2 === 0 ? `Edward King ${i}` : 'Another Row',
}));

const rowSelection: TableRowSelection<DataType> = {
  renderCell: (checked, _record, index, node) => ({
    props: { rowSpan: index % 2 === 0 ? 2 : 0 },
    children: (
      <>
        {String(checked)}: {node}
      </>
    ),
  }),
};

const App: React.FC = () => (
  <Table<DataType> rowSelection={rowSelection} columns={columns}
  dataSource={dataSource} />
);

export default App;

```

Fixed header and scroll bar with the page

```

import React, { useState } from 'react';
import { Switch, Table } from 'antd';
import type { TableColumnsType } from 'antd';

interface DataType {
  key: React.Key;
  name: string;
  age: number;
  address: string;
}

const columns: TableColumnsType<DataType> = [
  {
    title: 'Full Name',
    width: 100,
    dataIndex: 'name',
    key: 'name',
    fixed: 'left',
  },
  {
    title: 'Age',

```

```
    width: 100,
    dataIndex: 'age',
    key: 'age',
    fixed: 'left',
  },
  {
    title: 'Column 1',
    dataIndex: 'address',
    key: '1',
    width: 150,
  },
  {
    title: 'Column 2',
    dataIndex: 'address',
    key: '2',
    width: 150,
  },
  {
    title: 'Column 3',
    dataIndex: 'address',
    key: '3',
    width: 150,
  },
  {
    title: 'Column 4',
    dataIndex: 'address',
    key: '4',
    width: 150,
  },
  {
    title: 'Column 5',
    dataIndex: 'address',
    key: '5',
    width: 150,
  },
  {
    title: 'Column 6',
    dataIndex: 'address',
    key: '6',
    width: 150,
  },
  {
    title: 'Column 7',
    dataIndex: 'address',
    key: '7',
    width: 150,
```

```

    },
    { title: 'Column 8', dataIndex: 'address', key: '8' },
    {
      title: 'Action',
      key: 'operation',
      fixed: 'right',
      width: 100,
      render: () => <a>action</a>,
    },
  ],
];

const dataSource = Array.from({ length: 100 }).map<DataType>((_, i) => ({
  key: i,
  name: `Edward ${i}`,
  age: 32,
  address: `London Park no. ${i}`,
}));

const App: React.FC = () => {
  const [fixedTop, setFixedTop] = useState(false);
  return (
    <Table<DataType>
      columns={columns}
      dataSource={dataSource}
      scroll={{ x: 1500 }}
      summary={() => (
        <Table.Summary fixed={fixedTop ? 'top' : 'bottom'}>
          <Table.Summary.Row>
            <Table.Summary.Cell index={0} colSpan={2}>
              <Switch
                checkedChildren="Fixed Top"
                uncheckedChildren="Fixed Top"
                checked={fixedTop}
                onChange={() => {
                  setFixedTop(!fixedTop);
                }}
              />
            </Table.Summary.Cell>
            <Table.Summary.Cell index={2} colSpan={8}>
              Scroll Context
            </Table.Summary.Cell>
            <Table.Summary.Cell index={10}>Fix Right</Table.Summary.Cell>
          </Table.Summary.Row>
        </Table.Summary>
      )}
    // antd site header height
  )

```

```

        sticky={{ offsetHeader: 64 }}
      />
    );
  };

export default App;

```

Dynamic Settings

```

import React, { useState } from 'react';
import { DownOutlined } from '@ant-design/icons';
import type { GetProp, RadioChangeEvent, TableProps } from 'antd';
import { Form, Radio, Space, Switch, Table } from 'antd';

type SizeType = TableProps['size'];
type ColumnsType<T extends object> = GetProp<TableProps<T>, 'columns'>;
type TablePagination<T extends object> = NonNullable<Exclude<TableProps<T>
['pagination'], boolean>>;
type TablePaginationPosition = NonNullable<TablePagination<any>
['position']>[number];
type ExpandableConfig<T extends object> = TableProps<T>['expandable'];
type TableRowSelection<T extends object> = TableProps<T>['rowSelection'];

interface DataType {
  key: number;
  name: string;
  age: number;
  address: string;
  description: string;
}

const columns: ColumnsType<DataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
  },
  {
    title: 'Age',
    dataIndex: 'age',
    sorter: (a, b) => a.age - b.age,
  },
  {
    title: 'Address',
    dataIndex: 'address',
    filters: [

```

```

        text: 'London',
        value: 'London',
      },
      {
        text: 'New York',
        value: 'New York',
      },
    ],
    onFilter: (value, record) => record.address.indexOf(value as string)
=== 0,
  },
  {
    title: 'Action',
    key: 'action',
    sorter: true,
    render: () => (
      <Space size="middle">
        <a>Delete</a>
        <a>
          <Space>
            More actions
            <DownOutlined />
          </Space>
        </a>
      </Space>
    ),
  },
];

const data = Array.from({ length: 10 }).map<DataType>((_, i) => ({
  key: i,
  name: 'John Brown',
  age: Number(`${i}2`),
  address: `New York No. ${i} Lake Park`,
  description: `My name is John Brown, I am ${i}2 years old, living in New
York No. ${i} Lake Park.`,
}));

const defaultExpandable: ExpandableConfig<DataType> = {
  expandedRowRender: (record: DataType) => <p>{record.description}</p>,
};

const defaultTitle = () => 'Here is title';
const defaultFooter = () => 'Here is footer';

const App: React.FC = () => {

```



```

const [bordered, setBordered] = useState(false);
const [loading, setLoading] = useState(false);
const [size, setSize] = useState<SizeType>('large');
const [expandable, setExpandable] = useState<ExpandableConfig<DataType>>
(defaultExpandable);
const [showTitle, setShowTitle] = useState(false);
const [showHeader, setShowHeader] = useState(true);
const [showFooter, setShowFooter] = useState(true);
const [rowSelection, setRowSelection] =
useState<TableRowSelection<DataType> | undefined>({});
const [hasData, setHasData] = useState(true);
const [tableLayout, setTableLayout] = useState<string>('unset');
const [top, setTop] = useState<TablePaginationPosition>('none');
const [bottom, setBottom] = useState<TablePaginationPosition>
('bottomRight');
const [ellipsis, setEllipsis] = useState(false);
const [yScroll, setYScroll] = useState(false);
const [xScroll, setXScroll] = useState<string>('unset');

const handleBorderChange = (enable: boolean) => {
  setBordered(enable);
};

const handleLoadingChange = (enable: boolean) => {
  setLoading(enable);
};

const handleSizeChange = (e: RadioChangeEvent) => {
  setSize(e.target.value);
};

const handleTableLayoutChange = (e: RadioChangeEvent) => {
  setTableLayout(e.target.value);
};

const handleExpandChange = (enable: boolean) => {
  setExpandable(enable ? defaultExpandable : undefined);
};

const handleEllipsisChange = (enable: boolean) => {
  setEllipsis(enable);
};

const handleTitleChange = (enable: boolean) => {
  setShowTitle(enable);
};

```

```

const handleHeaderChange = (enable: boolean) => {
  setShowHeader(enable);
};

const handleFooterChange = (enable: boolean) => {
  setShowFooter(enable);
};

const handleRowSelectionChange = (enable: boolean) => {
  setRowSelection(enable ? {} : undefined);
};

const handleYScrollChange = (enable: boolean) => {
  setYScroll(enable);
};

const handleXScrollChange = (e: RadioChangeEvent) => {
  setXScroll(e.target.value);
};

const handleDataChange = (newHasData: boolean) => {
  setHasData(newHasData);
};

const scroll: { x?: number | string; y?: number | string } = {};
if (yScroll) {
  scroll.y = 240;
}
if (xScroll !== 'unset') {
  scroll.x = '100vw';
}

const tableColumns = columns.map((item) => ({ ...item, ellipsis }));
if (xScroll === 'fixed') {
  tableColumns[0].fixed = true;
  tableColumns[tableColumns.length - 1].fixed = 'right';
}

const tableProps: TableProps<DataType> = {
  bordered,
  loading,
  size,
  expandable,
  title: showTitle ? defaultTitle : undefined,
  showHeader,

```

```

    footer: showFooter ? defaultFooter : undefined,
    rowSelection,
    scroll,
    tableLayout: tableLayout === 'unset' ? undefined : (tableLayout as
TableProps['tableLayout']),
  };

  return (
    <>
      <Form layout="inline" className="table-demo-control-bar" style={{
marginBottom: 16 }}>
        <Form.Item label="Bordered">
          <Switch checked={bordered} onChange={handleBorderChange} />
        </Form.Item>
        <Form.Item label="loading">
          <Switch checked={loading} onChange={handleLoadingChange} />
        </Form.Item>
        <Form.Item label="Title">
          <Switch checked={showTitle} onChange={handleTitleChange} />
        </Form.Item>
        <Form.Item label="Column Header">
          <Switch checked={showHeader} onChange={handleHeaderChange} />
        </Form.Item>
        <Form.Item label="Footer">
          <Switch checked={showFooter} onChange={handleFooterChange} />
        </Form.Item>
        <Form.Item label="Expandable">
          <Switch checked={!expandable} onChange={handleExpandChange} />
        </Form.Item>
        <Form.Item label="Checkbox">
          <Switch checked={!rowSelection} onChange=
{handleRowSelectionChange} />
        </Form.Item>
        <Form.Item label="Fixed Header">
          <Switch checked={!yScroll} onChange={handleYScrollChange} />
        </Form.Item>
        <Form.Item label="Has Data">
          <Switch checked={!hasData} onChange={handleDataChange} />
        </Form.Item>
        <Form.Item label="Ellipsis">
          <Switch checked={!ellipsis} onChange={handleEllipsisChange} />
        </Form.Item>
        <Form.Item label="Size">
          <Radio.Group value={size} onChange={handleSizeChange}>
            <Radio.Button value="large">Large</Radio.Button>
            <Radio.Button value="middle">Middle</Radio.Button>
          </Radio.Group>
        </Form.Item>
      </Form>
    </>
  );

```

```

        <Radio.Button value="small">Small</Radio.Button>
      </Radio.Group>
    </Form.Item>
    <Form.Item label="Table Scroll">
      <Radio.Group value={xScroll} onChange={handleXScrollChange}>
        <Radio.Button value="unset">Unset</Radio.Button>
        <Radio.Button value="scroll">Scroll</Radio.Button>
        <Radio.Button value="fixed">Fixed Columns</Radio.Button>
      </Radio.Group>
    </Form.Item>
    <Form.Item label="Table Layout">
      <Radio.Group value={tableLayout} onChange=
{handleTableLayoutChange}>
        <Radio.Button value="unset">Unset</Radio.Button>
        <Radio.Button value="fixed">Fixed</Radio.Button>
      </Radio.Group>
    </Form.Item>
    <Form.Item label="Pagination Top">
      <Radio.Group value={top} onChange={(e) =>
setTop(e.target.value)}>
        <Radio.Button value="topLeft">TopLeft</Radio.Button>
        <Radio.Button value="topCenter">TopCenter</Radio.Button>
        <Radio.Button value="topRight">TopRight</Radio.Button>
        <Radio.Button value="none">None</Radio.Button>
      </Radio.Group>
    </Form.Item>
    <Form.Item label="Pagination Bottom">
      <Radio.Group value={bottom} onChange={(e) =>
setBottom(e.target.value)}>
        <Radio.Button value="bottomLeft">BottomLeft</Radio.Button>
        <Radio.Button value="bottomCenter">BottomCenter</Radio.Button>
        <Radio.Button value="bottomRight">BottomRight</Radio.Button>
        <Radio.Button value="none">None</Radio.Button>
      </Radio.Group>
    </Form.Item>
  </Form>
  <Table<DataType>
    {...tableProps}
    pagination={{ position: [top, bottom] }}
    columns={tableColumns}
    dataSource={hasData ? data : []}
    scroll={scroll}
  />
</>
);
};

```

```
export default App;
```

selections with icon

Debug

```
import React from 'react';
import { Table } from 'antd';
import type { TableColumnsType } from 'antd';

interface DataType {
  key: React.Key;
  name: string;
  age: number;
  address: string;
}

const columns: TableColumnsType<DataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
  },
  {
    title: 'Age',
    dataIndex: 'age',
  },
  {
    title: 'Address',
    dataIndex: 'address',
  },
];

const data: DataType[] = [
  {
    key: '1',
    name: 'John Brown',
    age: 32,
    address: 'New York No. 1 Lake Park',
  },
  {
    key: '2',
    name: 'Jim Green',
    age: 42,
    address: 'London No. 1 Lake Park',
  },
];
```

```

    },
    {
      key: '3',
      name: 'Joe Black',
      age: 32,
      address: 'Sydney No. 1 Lake Park',
    },
  ],
];

const App: React.FC = () => (
  <Table<DataType>
    bordered
    rowSelection={{ type: 'checkbox', selections: true }}
    columns={columns}
    dataSource={data}
  />
);

export default App;

```

Component Token

Debug

```

import React, { useState } from 'react';
import { DownOutlined } from '@ant-design/icons';
import type { ConfigProviderProps, GetProp, RadioChangeEvent, TableProps }
from 'antd';
import { ConfigProvider, Form, Radio, Space, Switch, Table } from 'antd';

type SizeType = ConfigProviderProps['componentSize'];
type ColumnsType<T extends object> = GetProp<TableProps<T>, 'columns'>;
type TablePagination = Exclude<GetProp<TableProps, 'pagination'>, boolean>;
type TablePaginationPosition = NonNullable<TablePagination['position']>
[number];
type ExpandableConfig<T extends object> = GetProp<TableProps<T>,
'expandable'>;
type TableRowSelection<T extends object> = GetProp<TableProps<T>,
'rowSelection'>;

interface DataType {
  key: number;
  name: string;
  age: number;
  address: string;
  description: string;
}

```

```

}

const columns: ColumnsType<DataType> = [
  {
    title: 'Name',
    dataIndex: 'name',
  },
  {
    title: 'Age',
    dataIndex: 'age',
    sorter: (a, b) => a.age - b.age,
  },
  {
    title: 'Address',
    dataIndex: 'address',
    filters: [
      {
        text: 'London',
        value: 'London',
      },
      {
        text: 'New York',
        value: 'New York',
      },
    ],
    onFilter: (value, record) => record.address.indexOf(value as string)
    === 0,
  },
  {
    title: 'Action',
    key: 'action',
    sorter: true,
    render: () => (
      <Space size="middle">
        <a>Delete</a>
        <a>
          <Space>
            More actions
            <DownOutlined />
          </Space>
        </a>
      </Space>
    ),
  },
];

```

```

const dataSource = Array.from({ length: 10 }).map<DataType>((_, i) => ({
  key: i,
  name: 'John Brown',
  age: Number(`${i}2`),
  address: `New York No. ${i} Lake Park`,
  description: `My name is John Brown, I am ${i}2 years old, living in New
York No. ${i} Lake Park.` ,
}));

const defaultExpandable: ExpandableConfig<DataType> = {
  expandedRowRender: (record: DataType) => <p>{record.description}</p>,
};

const defaultTitle = () => 'Here is title';
const defaultFooter = () => 'Here is footer';

const App: React.FC = () => {
  const [bordered, setBordered] = useState(false);
  const [loading, setLoading] = useState(false);
  const [size, setSize] = useState<SizeType>('large');
  const [expandable, setExpandable] = useState<ExpandableConfig<DataType> |
undefined>(
    defaultExpandable,
  );
  const [showTitle, setShowTitle] = useState(false);
  const [showHeader, setShowHeader] = useState(true);
  const [showFooter, setShowFooter] = useState(true);
  const [rowSelection, setRowSelection] =
useState<TableRowSelection<DataType> | undefined>({});
  const [hasData, setHasData] = useState(true);
  const [tableLayout, setTableLayout] = useState<string>('unset');
  const [top, setTop] = useState<TablePaginationPosition>('none');
  const [bottom, setBottom] = useState<TablePaginationPosition>
('bottomRight');
  const [ellipsis, setEllipsis] = useState(false);
  const [yScroll, setYScroll] = useState(false);
  const [xScroll, setXScroll] = useState<string>('unset');

  const handleBorderChange = (enable: boolean) => {
    setBordered(enable);
  };

  const handleLoadingChange = (enable: boolean) => {
    setLoading(enable);
  };

```



```
const handleSizeChange = (e: RadioChangeEvent) => {
  setSize(e.target.value);
};

const handleTableLayoutChange = (e: RadioChangeEvent) => {
  setTableLayout(e.target.value);
};

const handleExpandChange = (enable: boolean) => {
  setExpandable(enable ? defaultExpandable : undefined);
};

const handleEllipsisChange = (enable: boolean) => {
  setEllipsis(enable);
};

const handleTitleChange = (enable: boolean) => {
  setShowTitle(enable);
};

const handleHeaderChange = (enable: boolean) => {
  setShowHeader(enable);
};

const handleFooterChange = (enable: boolean) => {
  setShowFooter(enable);
};

const handleRowSelectionChange = (enable: boolean) => {
  setRowSelection(enable ? {} : undefined);
};

const handleYScrollChange = (enable: boolean) => {
  setYScroll(enable);
};

const handleXScrollChange = (e: RadioChangeEvent) => {
  setXScroll(e.target.value);
};

const handleDataChange = (newHasData: boolean) => {
  setHasData(newHasData);
};

const scroll: { x?: number | string; y?: number | string } = {};
if (yScroll) {
```

```

    scroll.y = 240;
  }
  if (xScroll !== 'unset') {
    scroll.x = '100vw';
  }

  const tableColumns = columns.map((item) => ({ ...item, ellipsis }));
  if (xScroll === 'fixed') {
    tableColumns[0].fixed = true;
    tableColumns[tableColumns.length - 1].fixed = 'right';
  }

  const tableProps: TableProps<DataType> = {
    bordered,
    loading,
    size,
    expandable,
    title: showTitle ? defaultTitle : undefined,
    showHeader,
    footer: showFooter ? defaultFooter : undefined,
    rowSelection,
    scroll,
    tableLayout: tableLayout === 'unset' ? undefined : (tableLayout as
TableProps['tableLayout']),
  };

  return (
    <>
      <Form layout="inline" className="table-demo-control-bar" style={{
marginBottom: 16 }}>
        <Form.Item label="Bordered">
          <Switch checked={bordered} onChange={handleBorderChange} />
        </Form.Item>
        <Form.Item label="loading">
          <Switch checked={loading} onChange={handleLoadingChange} />
        </Form.Item>
        <Form.Item label="Title">
          <Switch checked={showTitle} onChange={handleTitleChange} />
        </Form.Item>
        <Form.Item label="Column Header">
          <Switch checked={showHeader} onChange={handleHeaderChange} />
        </Form.Item>
        <Form.Item label="Footer">
          <Switch checked={showFooter} onChange={handleFooterChange} />
        </Form.Item>
        <Form.Item label="Expandable">

```

```

        <Switch checked={!expandable} onChange={handleExpandChange} />
    </Form.Item>
    <Form.Item label="Checkbox">
        <Switch checked={!rowSelection} onChange=
{handleRowSelectionChange} />
    </Form.Item>
    <Form.Item label="Fixed Header">
        <Switch checked={!yScroll} onChange={handleYScrollChange} />
    </Form.Item>
    <Form.Item label="Has Data">
        <Switch checked={!hasData} onChange={handleDataChange} />
    </Form.Item>
    <Form.Item label="Ellipsis">
        <Switch checked={!ellipsis} onChange={handleEllipsisChange} />
    </Form.Item>
    <Form.Item label="Size">
        <Radio.Group value={size} onChange={handleSizeChange}>
            <Radio.Button value="large">Large</Radio.Button>
            <Radio.Button value="middle">Middle</Radio.Button>
            <Radio.Button value="small">Small</Radio.Button>
        </Radio.Group>
    </Form.Item>
    <Form.Item label="Table Scroll">
        <Radio.Group value={xScroll} onChange={handleXScrollChange}>
            <Radio.Button value="unset">Unset</Radio.Button>
            <Radio.Button value="scroll">Scroll</Radio.Button>
            <Radio.Button value="fixed">Fixed Columns</Radio.Button>
        </Radio.Group>
    </Form.Item>
    <Form.Item label="Table Layout">
        <Radio.Group value={tableLayout} onChange=
{handleTableLayoutChange}>
            <Radio.Button value="unset">Unset</Radio.Button>
            <Radio.Button value="fixed">Fixed</Radio.Button>
        </Radio.Group>
    </Form.Item>
    <Form.Item label="Pagination Top">
        <Radio.Group value={top} onChange={(e) =>
setTop(e.target.value)}>
            <Radio.Button value="topLeft">TopLeft</Radio.Button>
            <Radio.Button value="topCenter">TopCenter</Radio.Button>
            <Radio.Button value="topRight">TopRight</Radio.Button>
            <Radio.Button value="none">None</Radio.Button>
        </Radio.Group>
    </Form.Item>
    <Form.Item label="Pagination Bottom">

```

```

        <Radio.Group value={bottom} onChange={(e) =>
setBottom(e.target.value)}>
            <Radio.Button value="bottomLeft">BottomLeft</Radio.Button>
            <Radio.Button value="bottomCenter">BottomCenter</Radio.Button>
            <Radio.Button value="bottomRight">BottomRight</Radio.Button>
            <Radio.Button value="none">None</Radio.Button>
        </Radio.Group>
    </Form.Item>
</Form>
<ConfigProvider
    theme={{
        components: {
            Table: {
                colorBgContainer: '#e6f4ff',
                headerBg: '#1677ff',
                headerColor: '#fff',
                headerSortActiveBg: '#0958d9',
                headerSortHoverBg: '#69b1ff',
                bodySortBg: '#1677ff10',
                rowHoverBg: '#1677ff10',
                rowSelectedBg: '#bae0ff',
                rowSelectedHoverBg: '#91caff',
                rowExpandedBg: '#1677ff10',
                cellPaddingBlock: 20,
                cellPaddingInline: 20,
                cellPaddingBlockMD: 16,
                cellPaddingInlineMD: 16,
                cellPaddingBlockSM: 12,
                cellPaddingInlineSM: 12,
                borderColor: '#e6f4ff',
                headerBorderRadius: 0,
                footerBg: '#1677ff',
                footerColor: '#fff',
                cellFontSize: 16,
                cellFontSizeMD: 16,
                cellFontSizeSM: 14,
                headerSplitColor: '#fff',
                headerFilterHoverBg: 'rgba(0, 0, 0, 0.12)',
                filterDropDownMenuBg: '#fff',
                filterDropDownBg: '#fff',
                expandIconBg: '#e6f4ff',
            },
        },
    }}
>
    <Table<DataType>

```

```

    {...tableProps}
    pagination={{ position: [top, bottom] }}
    columns={tableColumns}
    dataSource={hasData ? dataSource : []}
    scroll={scroll}
  />
</ConfigProvider>
</>
);
};

export default App;

```

API

Common props ref: [Common props](#)

Table

Property	Description	Type	Default	
bordered	Whether to show all table borders	boolean	false	
columns	Columns of table	ColumnsType[]	-	
components	Override default table elements	TableComponents	-	
dataSource	Data record array to be displayed	object[]	-	
expandable	Config expandable content	expandable	-	
footer	Table footer renderer	function(currentPageData)	-	
getPopupContainer	The render container of dropdowns in table	(triggerNode) => HTMLElement	() => TableHtmlElement	

loading	Loading status of table	boolean Spin Props	false	
locale	The i18n text including filter, sort, empty text, etc	object	Default Value	
pagination	Config of pagination. You can ref table pagination config or full pagination document, hide it by setting it to false	object false	-	
rowClassName	Row's className	function(record, index): string	-	
rowKey	Row's unique key, could be a string or function that returns a string	string function(record): string	key	
rowSelection	Row selection config	object	-	
rowHoverable	Row hover	boolean	true	5.16
scroll	Whether the table can be scrollable, config	object	-	
showHeader	Whether to show table header	boolean	true	

showSorterTooltip	The header show next sorter direction tooltip. It will be set as the property of Tooltip if its type is object	boolean Tooltip.props & {target?: 'full-header' 'sorter-icon' }	{ target: 'full-header' }	5.16
size	Size of table	large middle small	large	
sortDirections	Supported sort way, could be ascend, descend	Array	[ascend, descend]	
sticky	Set sticky header and scroll bar	boolean {offsetHeader?: number, offsetScroll?: number, getContainer?: () => HTMLDivElement}	-	4.6. (get 4.7.0)
summary	Summary content	(currentData) => ReactNode	-	
tableLayout	The table-layout attribute of table element	- auto fixed	- fixed when header/columns are fixed, or using column.ellipsis	
title	Table title render	function(currentPageData)	-	
virtual	Support virtual list	boolean	-	5.9.

onChange	Callback executed when pagination, filters or sorter is changed	function(pagination, filters, sorter, extra: { currentDataSource: [], action: paginate sort filter })	-	
onHeaderRow	Set props on per header row	function(columns, index)	-	
onRow	Set props on per row	function(record, index)	-	
onScroll	Triggered when the table body is scrolled. Note that only vertical scrolling will trigger the event when virtual	function(event)	-	5.16

Table ref

Property	Description	Type	Version
nativeElement	The wrap element	HTMLDivElement	5.11.0
scrollTo	Trigger to scroll to target position. key match with record rowKey	(config: { index?: number, key?: React.Key, top?: number }) => void	5.11.0

onRow usage

Same as `onRow` `onHeaderRow` `onCell` `onHeaderCell`

```

<Table
  onRow={(record, rowIndex) => {
    return {
      onClick: (event) => {}, // click row
      onDoubleClick: (event) => {}, // double click row
      onContextMenu: (event) => {}, // right button click row
      onMouseEnter: (event) => {}, // mouse enter row
      onMouseLeave: (event) => {}, // mouse leave row
    }
  }}

```



```

    };
  }}
  onHeaderRow={ (columns, index) => {
    return {
      onClick: () => {}, // click header row
    };
  }}
/>

```

Column

One of the Table `columns` prop for describing the table's columns, Column has the same API.

Property	Description	Type
align	The specify which way that column is aligned	left right center
className	The className of this column	string
colSpan	Span of this column's title	number
dataIndex	Display field of the data record, support nest path by string array	string string[]
defaultFilteredValue	Default filtered values	string[]
filterResetToDefaultFilteredValue	click the reset button, whether to restore the default filter	boolean
defaultSortOrder	Default order of sorted values	ascend descend
ellipsis	The ellipsis cell content, not working with sorter and filters for now. tableLayout would be fixed when ellipsis is true or { showTitle?: boolean }	boolean { showTitle?: boolean }
filterDropdown	Customized filter overlay	ReactNode (props: FilterDropdownProps) => ReactNode
filtered	Whether the dataSource is filtered	boolean
filteredValue	Controlled filtered value, filter icon will highlight	string[]

filterIcon	Customized filter icon	ReactNode (filtered: boolean) => ReactNode
filterOnClose	Whether to trigger filter when the filter menu closes	boolean
filterMultiple	Whether multiple filters can be selected	boolean
filterMode	To specify the filter interface	'menu' 'tree'
filterSearch	Whether to be searchable for filter menu	boolean function(input, record):boolean
filters	Filter menu config	object[]
filterDropdownProps	Customized dropdown props, filterDropdownOpen and onFilterDropdownOpenChange were available before <5.22.0	DropdownProps
fixed	(IE not support) Set column to be fixed: true(same as left) 'left' 'right'	boolean string
key	Unique key of this column, you can ignore this prop if you've set a unique dataIndex	string
render	Renderer of the table cell. value is the value of current cell; record is the value object of current row; index is the row number. The return value should be a ReactNode	function(value, record, index) {}
responsive	The list of breakpoints at which to display this column. Always visible if not set	Breakpoint []
rowScope	Set scope attribute for all cells in this column	row rowgroup
shouldCellUpdate	Control cell render logic	(record, prevRecord) => boolean
showSorterTooltip	If header show next sorter direction tooltip, override	boolean Tooltip props & {target?:

	showSorterTooltip in table	'full-header' 'sorter-icon' }
sortDirections	Supported sort way, override sortDirections in Table, could be ascend, descend	Array
sorter	Sort function for local sort, see Array.sort 's compareFunction. If it is server-side sorting, set to true, but if you want to support multi-column sorting, you can set it to { multiple: number }	function boolean { compare: function, multiple: number }
sortOrder	Order of sorted values: ascend descend null	ascend descend null
sortIcon	Customized sort icon	(props: { sortOrder }) => ReactNode
title	Title of this column	ReactNode ({ sortOrder, sortColumn, filters }) => ReactNode
width	Width of this column (width not working?)	string number
minWidth	Min width of this column, only works when tableLayout="auto"	number
hidden	Hidden this column	boolean
onCell	Set props on per cell	function(record, rowIndex)
onFilter	Function that determines if the row is displayed when filtered	function(value, record) => boolean
onHeaderCell	Set props on per header cell	function(column)

ColumnGroup

Property	Description	Type	Default
title	Title of the column group	ReactNode	-

pagination

Properties for pagination.

Property	Description	Type	Default
position	Specify the position of Pagination, could be topLeft topCenter topRight bottomLeft bottomCenter bottomRight	Array	[bottomRight]

More about pagination, please check [Pagination](#) .

expandable

Properties for expandable.

Property	Description	Type	Default	Version
childrenColumnName	The column contains children to display	string	children	
columnTitle	Set the title of the expand column	ReactNode	-	4.23.0
columnWidth	Set the width of the expand column	string number	-	
defaultExpandAllRows	Expand all rows initially	boolean	false	
defaultExpandedRowKeys	Initial expanded row keys	string[]	-	
expandedRowClassName	Expanded row's className	string (record, index, indent) => string	-	string: 5.22.0
expandedRowKeys	Current expanded row keys	string[]	-	
expandedRowRender	Expanded container render for each row	function(record, index, indent, expanded): ReactNode	-	

expandIcon	Customize row expand icon. Ref example	function(props): ReactNode	-	
expandRowByClick	Whether to expand row by clicking anywhere in the whole row	boolean	false	
fixed	Whether the expansion icon is fixed. Optional true left right	boolean string	false	4.16.0
indentSize	Indent size in pixels of tree data	number	15	
rowExpandable	Enable row can be expandable	(record) => boolean	-	
showExpandColumn	Show expand column	boolean	true	4.18.0
onExpand	Callback executed when the row expand icon is clicked	function(expanded, record)	-	
onExpandedRowsChange	Callback executed when the expanded rows change	function(expandedRows)	-	

rowSelection

Properties for row selection.

Property	Description	Type	Default	
checkStrictly	Check table row precisely; parent row and children rows are not associated	boolean	true	4.0
columnTitle	Set the title of the selection column	ReactNode (originalNode: ReactNode) => ReactNode	-	
columnWidth	Set the width of the selection column	string number	32px	
fixed	Fixed selection column on the left	boolean	-	
getCheckboxProps	Get Checkbox or Radio props	function(record)	-	
hideSelectAll	Hide the selectAll checkbox and custom selection	boolean	false	4.0
preserveSelectedRowKeys	Keep selection key even when it removed from dataSource	boolean	-	4.0
renderCell	Renderer of the table cell. Same as render in column	function(checked, record, index, originNode) {}	-	4.0
selectedRowKeys	Controlled selected row keys	string[] number[]	[]	
selections	Custom selection config , only	object[] boolean	-	

	displays default selections when set to true			
type	checkbox or radio	checkbox radio	checkbox	
onCell	Set props on per cell. Same as onCell in column	function(record, rowIndex)	-	5.1
onChange	Callback executed when selected rows change	function(selectedRowKeys, selectedRows, info: { type })	-	in 4.1
onSelect	Callback executed when select/deselect one row	function(record, selected, selectedRows, nativeEvent)	-	
onSelectAll	Callback executed when select/deselect all rows	function(selected, selectedRows, changeRows)	-	
onSelectInvert	Callback executed when row selection is inverted	function(selectedRowKeys)	-	
onSelectNone	Callback executed when row selection is cleared	function()	-	
onSelectMultiple	Callback executed when row selection is changed by pressing shift	function(selected, selectedRows, changeRows)	-	

scroll

Property	Description	Type	Default
----------	-------------	------	---------

scrollToFirstRowOnChange	Whether to scroll to the top of the table when paging, sorting, filtering changes	boolean	-
x	Set horizontal scrolling, can also be used to specify the width of the scroll area, could be number, percent value, true and 'max-content'	string number true	-
y	Set vertical scrolling, can also be used to specify the height of the scroll area, could be string or number	string number	-

selection

Property	Description	Type	Default
key	Unique key of this selection	string	-
text	Display text of this selection	ReactNode	-
onSelect	Callback executed when this selection is clicked	function(changeableRowKeys)	-

Using in TypeScript

```
import React from 'react';
import { Table } from 'antd';
import type { TableColumnsType } from 'antd';

interface User {
  key: number;
  name: string;
}

const columns: TableColumnsType<User> = [
  {
    key: 'name',
    title: 'Name',
    dataIndex: 'name',
  },
];

const data: User[] = [
  {
    key: 0,
    name: 'Jack',
  },
];
```



```

    },
  ];

  const Demo: React.FC = () => (
    <>
      <Table<User> columns={columns} dataSource={data} />
      { /* JSX style usage */ }
      <Table<User> dataSource={data}>
        <Table.Column<User> key="name" title="Name" dataIndex="name" />
      </Table>
    </>
  );

  export default Demo;

```

Here is the [CodeSandbox for TypeScript](#).

Design Token

Note

According to the [React documentation](#), every child in an array should be assigned a unique key. The values inside the Table's `dataSource` and `columns` should follow this rule. By default, `dataSource[i].key` will be treated as the key value for `dataSource`.

Warning: Each child in an array or iterator should have a unique "key" prop. Check the render method of `Table`. See <https://fb.me/react-warning-keys> for more information.

If `dataSource[i].key` is not provided, then you should specify the primary key of `dataSource` value via `rowKey`, as shown below. If not, warnings like the one above will show in browser console.

```

// primary key is uid
return <Table rowKey="uid" />;
// or
return <Table rowKey={(record) => record.uid} />;

```

FAQ

How to hide pagination when single page or no data?

You can set `hideOnSinglePage` with `pagination` prop.

Table will return to first page when filter data.

Table total page count usually reduce after filter data, we by default return to first page in case of current page is out of filtered results.

You may need to keep current page after filtering when fetch data from remote service, please check [this demo](#) as workaround.

Also you can use the action from extra param to determine when return to first page.

Why Table pagination show size changer?

In order to improve user experience, Pagination show size changer by default when `total > 50` since `4.1.0`. You can set `showSizeChanger=false` to disable this feature.

Why Table fully render when state change?

Table can not tell what state used in `columns.render`, so it always need fully render to avoid sync issue. You can use `column.shouldCellUpdate` to control render.

How to handle fixed column display over the mask layout?

Fixed column use `z-index` to make it over other columns. You will find sometime fixed columns also over your mask layout. You can set `z-index` on your mask layout to resolve.

How to custom render Table Checkbox (For example, adding Tooltip) ?

Since `4.1.0`, You can use [rowSelection.renderCell](#) to custom render Table Checkbox. If you want to add Tooltip, please refer to this [demo](#).

Why does components.body.wrapper or components.body.row report an error when virtual is enabled?

Because virtual table needs to get its ref to do some calculations, so you need to use `React.forwardRef` wrapper and pass the ref to the dom