SQL Instructions

SELECT

```
SELECT predicate {*|table.*|[table.]field1 [, [table.]field2 [,...]]}
FROM tableexpression
```

- `predicate` - Can be one of the following predicates: `ALL` or `DISTINCT`. A predicate allows you to limit the number of records in the result. If you do not specify a predicate, `SQL` uses the default setting `ALL`.
- `asterisk *` - Indicates that all fields from the specified table or tables are selected.
- `table` - The name of the table that contains the fields from which records are selected. This is mainly used when retrieving data from multiple tables.
- `field1`, `field2` - The names of the fields from which data is retrieved. If you specify more than one field, they are retrieved in the order given.
- `tableexpression` - The name of the table or the names of the tables from which you want to retrieve data.
- If there are multiple options, they are listed between curly braces `{}`. The `|` is placed between the different options. Square brackets `[]` indicate optional elements.

WHERE

```
WHERE attribute operator constant|attribute
```

- `attribute` - The attribute is what you want to set a filter on. Usually, it refers to the name of a field.
- `operator` - The operator can be <, >, <=, >=, <>, =, `LIKE`, `BETWEEN ... AND ...`, `IN(...)`.

  The `LIKE` operator offers several special capabilities. The most important ones are:

  1. `%` matches any sequence of characters. E.g. `'Jo%'` matches both `'John'` and `'Joanna'`.
  2. `_` matches a single character. E.g. `'J_n'` matches both `'Jen'` and `'Jan'` but not `'John'`.
  3. `[]` matches any single character in brackets. E.g. `'J[ao]n'` matches `'Jan'` and `'Jon'` but not `'Jen'` nor `'John'`.
  4. `[^]` excludes all the single characters in brackets. E.g. `'J[^ao]n'` accepts every single character in between but excludes both `'Jan'` and `'Jon'`.

  Of course, these options can also be combined.
- `constant` - The constant can be a number or a sequence of characters. A character string must always begin and end with a quote `'...'`.
- The comparison and sorting of character strings depend on the collation setting of your database server. The default collation in MySQL does **not** distinguish between uppercase and lowercase letters.

ORDER BY

```
ORDER BY attribute asc|desc [, attribute asc|desc [,...]]  ]
```

- `attribute` - As an attribute, you can use a field name. You can also use the column number (first column = 1).
- `asc|desc` - With `asc`, you sort from smallest to largest; with `desc`, from largest to smallest. By default, data is sorted in ascending order.

GROUP BY

```
SELECT [predicate] FROM [table] WHERE [condition] GROUP BY [field]
```

With the help of `GROUP BY` and aggregate functions, you can perform calculations for groups of records instead of for all records together. A group consists of all records with the same values for all fields (or expressions) in the `GROUP BY` field.

HAVING

```
SELECT [predicate] FROM [table] WHERE [condition] GROUP BY [field] HAVING [condition]
```

`HAVING` allows you to filter based on the result of an aggregate function. Only the rows that meet the condition are shown in the final result.

INNER JOIN - 2 tables, 1 field

```
SELECT fields FROM table1 INNER JOIN table 2 ON table1.field1 operator table2.field2
```

- `table1`, `table2` - The names of the tables from which records are combined.
- `field1`, `field2` - The names of the fields that are linked. Both fields must have the same data type and contain the same kind of data. They do not need to have the same name. If the fields in the two tables have the same name, you distinguish them by placing the table name in front: `table.field`.
- `operator` - The desired relational comparison operator: `=`, `<`, `>`, `<=`, `>=`, `<>`.

INNER JOIN - 2 tables, multiple fields

```
SELECT fields FROM table1 INNER JOIN table 2 ON table1.field1 operator table2.field2
AND (table1.field2 operator table2.field2 OR table1.field3 operator table2.field3)
```

You can base a `JOIN` on multiple fields by combining the different comparisons with a logical operator. You need this syntax when you want to join two tables using a composite key (a key that consists of multiple fields).

INNER JOIN - multiple tables

```
SELECT fields FROM table1 INNER JOIN table 2 ON table1.field1 operator table2.field2
INNER JOIN table 3 ON table2.field2 operator table3.field3
```

OUTER JOIN

```
SELECT fields FROM table1 LEFT|RIGHT [OUTER] JOIN table 2
ON table1.field1 operator table2.field2
```

- `table1`, `table2` - The names of the tables from which records are combined.
- `field1`, `field2` - The names of the fields that are linked. Both fields must have the same data type and contain the same kind of data. They do not need to have the same name. If the fields in the two tables have the same name, you distinguish them by placing the table name in front: `table.field`.
- `operator` - The desired relational comparison operator: `=`, `<`, `>`, `<=`, `>=`, `<>`.

With a `LEFT JOIN` operation, you create a `LEFT OUTER JOIN`. In a `LEFT OUTER JOIN`, all records from the first (left) table are included, even if there are no matching values in the second (right) table.

With a `RIGHT JOIN` operation, you create a `RIGHT OUTER JOIN`. In a `RIGHT OUTER JOIN`, all records from the second (right) table are included, even if there are no matching values in the first (left) table.

! The word `OUTER` is not required in the instruction.

1. If you want to create a query that only includes records with identical data in the linked fields, use an `INNER JOIN` operation.
2. `LEFT JOIN` or `RIGHT JOIN` can be nested within `INNER JOIN`, but `INNER JOIN` cannot be nested within `LEFT JOIN` or `RIGHT JOIN`.
3. You can link multiple `ON` components.
4. You can also combine an `OUTER JOIN` with inner joins. You place the `OUTER JOIN` last in the sequence of joins.

SELF JOIN

```
SELECT fields FROM table as t1 INNER|LEFT|RIGHT JOIN table as t2
ON t1.field operator t2.field
```

UNION

```
query1 UNION [ALL] query2 [UNION [ALL] queryn [..]]
```

INSERT - user provided values

```
INSERT INTO table [(field1[,field2[,...]])]  VALUES (value1[,value2 [,...]])
```

- `table` - The name of the table to which the records are being added.
- `field1`, `field2` - These are the names of the fields to which the data is being added. They must always be enclosed in parentheses `()`.
- `value1`, `value2` - The values that must be added to the fields of the new record. Each value is inserted into the field that corresponds to its position in the list: `value1` is inserted into `field1`, `value2` into `field2`, and so on.

  Each value of a non-numeric field must be enclosed in single quotes `' '`. The different values are separated by commas. The entire set of values is enclosed in parentheses `()`.

INSERT - data from other tables

```
INSERT INTO table [(field1[,field2[,...]])]  SELECT [source.]  field1 [, field2 [,...]]
FROM tableexpression [WHERE condition]
```

- `table` - The name of the table to which the records are being added.
- `source` - The name of the table (or query) from which the records are copied.
- `field1`, `field2` - These fields contain the values that are being added. You retrieve the values from other tables.
- `tableexpression` - The name of the table or tables from which the added records originate. This can be: a single table name; a combination using `INNER JOIN`, `LEFT JOIN` or `RIGHT JOIN`; a saved query.

UPDATE

```
UPDATE tableexpression SET newvalue WHERE criteria
```

- `tableexpression` - The name of the table in which data needs to be modified.
- `newvalue` - An expression that determines which value is inserted into a specific field in the updated records.
- `criteria` - An expression that determines which records are updated. Only the records that meet the expression are updated.

DELETE

```
DELETE [table] FROM tableexpression WHERE criteria
```

- `table` - The optional name of the table from which the records are deleted.
- `tableexpression` - The name of the table from which records must be deleted.
- `criteria` - An expression that determines which records are deleted. Only the records that meet the expression are deleted.

If you use an `UPDATE` or `DELETE` statement without a `WHERE` clause on the primary key, you get `error code 1175`. Execute the statement `SET SQL_SAFE_UPDATES = 0` to allow all types of update instructions. MySQL remembers this until you close the Workbench.

CREATE TABLE - empty table

```
CREATE TABLE table
(field1 type [(size)] [not null] [index1]
[, field2 type [(size)] [not null] [index2] [,...]  ]
[CONSTRAINT multipleindex [,...]])
```

- `table` - This is the name of the table you want to create.
- `field1`, `field2` - These are the names of the fields you want to add to the table. At least one field must be created. These must always be enclosed in parentheses `()`.
- `type` - This is the data type of the field in the new table (refer to Table 1).
- `size` - This is the field size in characters (only for text and binary fields).
- `index1`, `index2`, `multipleindex` - This creates an index.

If `not null` is specified for a field, the content of the field must always be filled in.

CREATE TABLE - based on data from another table

```
CREATE TABLE newtable AS SELECT field1[,field2[,...]]  FROM source
```

- `newtable` - This is the name of the table that needs to be created. If the name of the new table is the same as an existing table, you will get an error.
- `field1`, `field2` - These are the names of the fields that need to be copied to the new table.
- `source` - This is the name of an existing table from which records are selected. This can be: a single table name; a combination using `INNER JOIN`, `LEFT JOIN` or `RIGHT JOIN`; a saved query.

DROP TABLE

```
DROP TABLE [table]
```

- `table` - The name of the table you want to delete.

The `DROP` statement can be used to completely remove an existing table from the database.

ADD FIELD

```
ALTER TABLE table
ADD [COLUMN] field1 type [(size)] [NOT NULL], field2 type [(size)] [NOT NULL],..., fieldn
type [(size)] [NOT NULL]
```

- `table` - Name of the table you want to modify.
- `field1`, `field2` - Name of the new fields.
- `type` - Data type of the new field (refer to Table 1).

- `size` - Field size in characters. Only for text and binary fields. The size is always enclosed in parentheses `()`.
- `not null` - When entering new records, only valid data may be entered in this field.

DROP FIELD

```
ALTER TABLE table DROP [COLUMN] field1, DROP [COLUMN] field2,...  fieldn
```

CONSTRAINT

The `CONSTRAINT` component is always used within a `CREATE TABLE` or `ALTER TABLE` statement. It allows you to:

- create primary keys
- create foreign keys
- define relationships
- enforce referential integrity

There are 2 types of `CONSTRAINT` components:

1. You can set a restrictive condition on one field. You place this in the field definition of the `ALTER TABLE` or directly after the data type in `CREATE TABLE`.
2. You can set a restrictive condition on multiple fields.

With a `CONSTRAINT`, you can set various restrictions on one or more fields: Unique, Primary key, Foreign key.

UNIQUE

This allows you to designate a field as a unique index. This can be applied to any field in your table. Unique prevents two records in the table from having the same value for a particular field.

If the unique index is applied to multiple fields, it means that the combination of the values in those fields must be unique.

PRIMARY KEY

With the reserved words `primary key`, you can designate one field or a group of fields in a table as the primary key.

- All values in the primary key must be unique and not Null.
- You can define only one primary key per table.
- The field with the primary key is often also an auto-numbering field. You can achieve this by adding the auto_increment option to the definition of your field. Note: This is only possible if you also make this field a primary key!

FOREIGN KEY

A foreign key is a field (or a combination of fields) that refers to the primary key of another table. This foreign key serves to guarantee the referential integrity of the data. In other words: only values defined in the primary key can be used as values in the foreign key.

Basic structure syntax

```
CREATE TABLE table (fielddescription1 [, fielddescription2 [,...]]
[, tableconstraintdescription[,...]  ]  )
```

Field description syntax

```
fieldname type [(size)] [NOT NULL] [INDEX [indexname]][AUTO_INCREMENT]
[ [CONSTRAINT constraintname]
UNIQUE|PRIMARY KEY|FOREIGN KEY REFERENCES tablename (primaryKeyName) ] )
```

Table constraint description syntax

```
CONSTRAINT constraintname
{ UNIQUE (fieldname1 [, fieldname2 [,...]])
|PRIMARY KEY (fieldname1 [, fieldname2 [,...]])
|FOREIGN KEY (fieldname1 [, fieldname2 [,...]])
REFERENCES tablename (primaryKeyName1 [,primaryKeyName2 [,...]])  }
```

- `constraintname` - The name of the restrictive condition.
- `fieldname` - The name of the field to which the constraint applies.
- `indexname` - Name of the index.
- `tablename` - The name of the table with which you want to establish a relationship. This is called the referenced table.
- `primaryKeyName` - The name of the field in the referenced table. This field is the primary key or part of the primary key in the referenced table.

CREATE INDEX

```
CREATE [UNIQUE] INDEX name ON table (field [asc | desc] [, field [asc | desc] [,...]])
```

- `name` - The name of the index.
- `table` - The name of the (existing) table for which you are creating an index.
- `field` - The name of the field or fields you want to include in the index. For each field, you can specify whether it should be sorted ascending (`ASC`) or descending (`DESC`). By default, this will be ascending.

! You can prevent duplicate values in the indexed field using the reserved word `UNIQUE`.

You can also create an index when creating the table or via `ALTER TABLE`.

```
ALTER TABLE table ADD INDEX indexname(columnname, ...)
```

DROP INDEX

```
DROP INDEX name ON table
```

CREATE VIEW

```
CREATE VIEW viewname AS SELECT-instruction
```

`ORDER BY` is not allowed. All other clauses are permitted, including the use of aggregate functions and expressions.

DROP VIEW

```
DROP VIEW viewname
```

DATA TYPES

Name	**Storage method**	**Characteristics**
Boolean	1 byte	Can only contain the value 0 or 1.
TinyInt	1 byte	Can contain a number between -128 and 127.
Datetime	8 bytes	Can contain a date and time value between 100 and 9999.
Float	4 bytes	Can contain a single-precision floating point number.
Decimal (M,D)	4 bytes per 9 characters	Contains a decimal number.
→ M = maximum number of digits (between 1 and 65).		
→ D = number of digits after the decimal point (between 0 and 30 and D $\leq$ M).		
→ It is always better to choose the `decimal` type over the `float` one to avoid rounding errors.		
Real - Double precision	8 bytes	Can contain a double-precision floating point number. This provides better accuracy than the Float type.
SmallInt	2 bytes	Can contain a number between -32768 and 32767.
Integer	4 bytes	Can contain a number between -2147483648 and 2147473647.
Char/Varchar	1 byte per character	Can contain a string
→ E.g. `name varchar(100)` means that `name` can contain a maximum of 100 characters.		
→ E.g. `code char(6)` means that `code` always contains 6 characters.		
Binary/Varbinary	1 byte per character	Can contain any type of information.
→ The use of Binary and Varbinary is similar to Char and Varchar.		

Table 1: Overview of the various data types