5.1 Physical design

MySQL storage engines

Logical design specifies tables, columns, and keys. The logical design process is described elsewhere in this material. ***Physical design*** specifies indexes, table structures, and partitions. Physical design affects query performance but never affects query results.

A ***storage engine*** or ***storage manager*** translates instructions generated by a query processor into low-level commands that access data on storage media. Storage engines support different index and table structures, so physical design is dependent on a specific storage engine.

MySQL can be configured with several different storage engines, including:

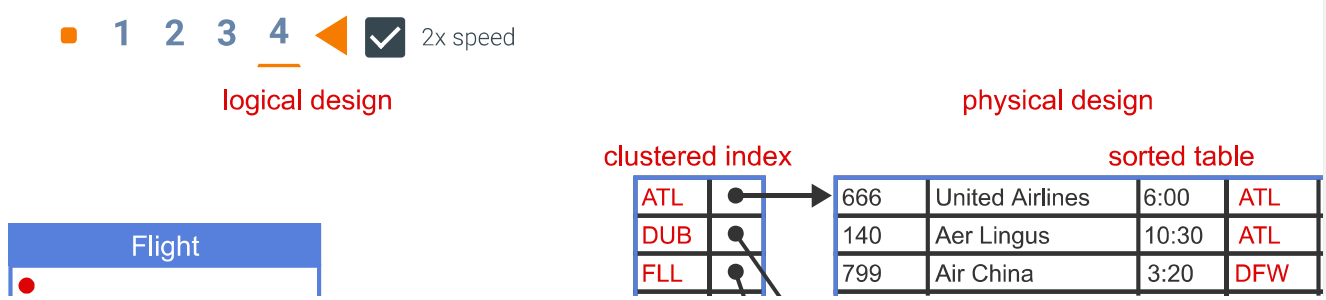
- InnoDB is the default storage engine installed with the MySQL download. InnoDB has full support for transaction management, foreign keys, referential integrity, and locking.
- MyISAM has limited transaction management and locking capabilities. MyISAM is commonly used for analytic applications with limited data updates.
- MEMORY stores all data in main memory. MEMORY is used for fast access with databases small enough to fit in main memory.

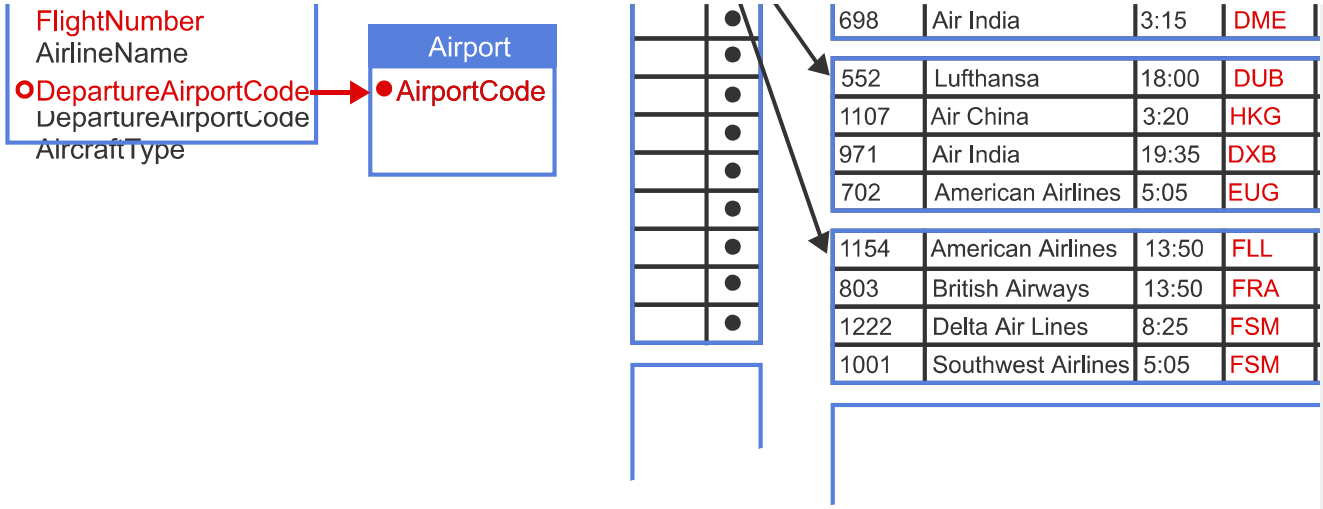
Different databases and storage engines support different table structures and index types. Ex:

- Table structure. Oracle Database supports heap, sorted, hash, and cluster tables. MySQL with InnoDB supports only heap and sorted tables.
- Index type. MySQL with InnoDB or MyISAM supports only B+tree indexes. MySQL with MEMORY supports both B+tree and hash indexes.

This section describes the physical design process and statements for MySQL with InnoDB. The process and statements can be adapted to other databases and storage engines, but details depend on supported index and table structures.

PARTICIPATION ACTIVITY 5.1.1: Logical and physical design.





Physical design also specifies indexes and index types.

Captions

1. Logical design specifies tables and columns.
2. Logical design also specifies primary and foreign keys.
3. Physical design specifies table structure. Flight table is sorted on DepartureAirportCode.
4. Physical design also specifies indexes and index types.

Feedback?

PARTICIPATION ACTIVITY 5.1.2: Logical and physical design.

Indicate whether each task is a logical or physical design activity.

1) Select a storage engine for MySQL.
   - ○ Logical design
   - ◉ Physical design

Correct

MySQL storage engines determine available table structures and index types, as well as query execution time.

2) Specify a column is UNIQUE.
   - ◉ Logical design
   - ○ Physical design

Correct

The UNIQUE constraint ensures values in all rows of one column are different. The UNIQUE constraint affects query results, such as UPDATE and INSERT, and therefore is logical.

3) Specify an index is CLUSTERED.

- ○ Logical design
- ◉ Physical design

Correct

A CLUSTERED index determines the order in which table rows are stored. Storage order affects query performance but not query results, and therefore is physical.

4) Determine foreign keys.

- ◉ Logical design
- ○ Physical design

Correct

Foreign keys are constrained to match the values of the corresponding primary keys. Specifying a foreign key affects results of UPDATE, INSERT, and DELETE queries and is logical.

Feedback?

CREATE INDEX, DROP INDEX, and SHOW INDEX statements

In MySQL with InnoDB:

- Indexes are always B+tree indexes.
- A primary index is automatically created on every primary key.
- A secondary index is automatically created on every foreign key.
- Additional secondary indexes are created manually with the CREATE INDEX statement.
- Tables with a primary key have sorted structure. Tables with no primary key have a heap structure.

The ***CREATE INDEX*** statement creates an index by specifying the index name and table columns that compose the index. Most indexes specify just one column, but a composite index specifies multiple columns.

The ***DROP INDEX*** statement deletes a table's index.

The ***SHOW INDEX*** statement displays a table's index. SHOW INDEX generates a result table with one row for each column of each index. A multi-column index has multiple rows in the result table.

The SQL standard includes logical design statements such as CREATE TABLE but not physical design statements such as CREATE INDEX. Nevertheless, CREATE INDEX and many other physical design statements are similar in most relational databases.

Table 5.1.1: INDEX statements.

Statement	Description	Syntax

CREATE INDEX	Create an index	`CREATE INDEX IndexName ON TableName (Column1, Column2, ..., ColumnN);`
DROP INDEX	Delete an index	`DROP INDEX IndexName ON TableName;`
SHOW INDEX	Show an index	`SHOW INDEX FROM TableName;`

Feedback?

Table 5.1.2: SHOW INDEX result table (selected columns).

Column Name	Column Meaning
Table	Name of indexed table
Non_unique	0 if index is on unique column 1 if index is on non-unique column
Key_name	Name of index as specified in CREATE INDEX statement or created by MySQL
Seq_in_index	1 for single-column indexes Numeric order of column in multi-column indexes
Column_name	Name of indexed column
Cardinality	Number of distinct values in indexed column
Null	YES if NULLs are allowed in column Blank if NULLs are not allowed in column
Index_type	Always BTREE for InnoDB storage engine

Feedback?

PARTICIPATION ACTIVITY 5.1.3: INDEX Statements.

1 2 3 4 5

2x speed

Address

●AddressID	Street	District	○CityID	PostalCode
1	1913 Hanoi Way	Nagasaki	463	35200
2	1121 Loja Avenue	California	449	17886
3	692 Joliet Street	Attika	38	83579
4	1566 Inegl Manor	Mandalay	349	53561
5	53 Idfu Parkway	Nantou	361	42399

```
CREATE INDEX PostalCodeIndex
ON Address (PostalCode);

SHOW INDEX FROM Address;
```

Results

Table	Non_unique	Key_name	Seq_in_index	Column_name	Cardinality	Null	Index_
Address	0	PRIMARY	1	AddressID	603		BTREE
Address	1	idx_fk_city_id	1	CityID	99		BTREE
Address	1	PostalCodeIndex	1	PostalCode	597	YES	BTREE

(selected columns)

PostalCodeIndex was created with the CREATE INDEX statement.
PostalCodeIndex is an index on a non-unique column with 597 distinct values.

Captions

1. The Address table has primary key AddressID and foreign key CityID.
2. The CREATE INDEX statement creates a secondary index on PostalCode.
3. The SHOW INDEX statement displays all indexes on the Address table. Some columns of the result table have been omitted for clarity.
4. MySQL created indexes on the primary and foreign keys automatically when Address was created. AddressID has 603 distinct index values, and CityID has 99 distinct index values.
5. PostalCodeIndex was created with the CREATE INDEX statement. PostalCodeIndex is an index on a non-unique column with 597 distinct values.

Feedback?

PARTICIPATION ACTIVITY 5.1.4: INDEX statements.

Indexes are created on the Flight table as follows:

```
CREATE INDEX FirstIndex
ON Flight (FlightNumber, AirlineName);
```

```
CREATE INDEX SecondIndex
ON Flight (DepartureAirportCode);
```

The questions below refer to the result table of the following statement:

```
SHOW INDEX FROM Flight;
```

1) How many result table rows have 'FirstIndex' in the Key_name column?

2

Check Show answer

Correct

two or 2

SHOW INDEX returns one row for each indexed column. FirstIndex is an index on two columns and therefore has two rows in the result table.

2) FlightNumber is the primary key of Flight. Flight has 90 rows. In the row with Column_name = "FlightNumber", what is the value of Cardinality?

90

Check Show answer

Correct

90 or ninety

Cardinality is the number of distinct values in the indexed column. Since FlightNumber is the primary key, FlightNumber is unique and has one distinct value for each table row.

3) In the row with Column_name = "AirlineName", what is the value of Seq_in_index?

2

Check Show answer

Correct

2 or two

AirlineName is the second column of FirstIndex, so Seq_in_index is 2.

Feedback?

EXPLAIN statement

The ***EXPLAIN*** statement generates a result table that describes how a statement is executed by the storage engine. EXPLAIN syntax is simple and uniform in most databases: `EXPLAIN statement;` The `statement` can be any SELECT, INSERT, UPDATE, or DELETE statement.

Although the EXPLAIN statement is supported by most relational databases, the result table varies significantly. In MySQL with InnoDB, the result table has one row for each table in the statement. If the statement contains multiple queries, such as a main query and a subquery, the result table has one row for each table in each query.

The type column of the EXPLAIN result table indicates how MySQL processes a join query. Processing join queries efficiently is a complex problem, so the type column has many alternative values. Example values appear in the table below. For more information, see MySQL EXPLAIN result table.

Table 5.1.3: EXPLAIN result table (selected columns).

Column Name	Column Meaning
select_type	The query type. Example query types: **SIMPLE** indicates query is neither nested nor union **PRIMARY** indicates query is the outer SELECT of nested query **SUBQUERY** indicates query is an inner SELECT of nested query
table	Name of table described in row of EXPLAIN result table
type	The join type. Example join types: **const** indicates the table has at most one matching row **range** indicates a join column is compared to a constant using operators such as BETWEEN, LIKE, or IN() **eq_ref** indicates one table row is read for each combination of rows from other tables (typically, an equijoin) **ALL** indicates a table scan is executed for each combination of rows from other tables
possible_keys	All available indexes that might be used to process the query
key	The index selected to process the query **NULL** indicates a table scan is performed
ref	The constant, column, or expression to which the selected index is compared
rows	Estimated number of rows read from table
filtered	Estimated number of rows selected by WHERE clause / estimated number of rows read from table

Feedback?

PARTICIPATION ACTIVITY 5.1.5: EXPLAIN statement.

1 2 3 4 5 2x speed

Address

AddressID	Street	District	CityID	PostalCode
1	1913 Hanoi Way	Nagasaki	463	35200
2	1121 Loja Avenue	California	449	17886
3	692 Joliet Street	Attika	38	83579
4	1566 Inegl Manor	Mandalay	349	53561

(603 rows total)

City

CityID	CityName
1	Abu Dhabi
2	Acua
3	Adana
4	Addis Abeb

(57 rows total)

```
EXPLAIN SELECT Street, CityName, PostalCode
        FROM Address, City
        WHERE Address.CityID = City.CityID AND Address.PostalCode > 40000;
```

Join results

Street	CityName	PostalCode
53 Idfu Parkway	Nantou	42399
613 Korolev Drive	Masqat	45844
419 Iligan Lane	Bhopal	72878
320 Brest Avenue	Kaduna	43331
96 Tafuna Way	Crdoba	99865

Explain results

select_type	table	type	possible_keys	key	ref	rows
SIMPLE	Address	ALL	idx_fk_city_id, PostalCodeIndex	NULL	NULL	603
SIMPLE	City	eq_ref	PRIMARY	PRIMARY	sakila.address.city_id	1

(selected columns)

The City table is accessed via primary index. One City row is read and selected for each Address

Captions

1. Address table has 603 rows. City table has 57 rows.
2. The SELECT statement joins Address and City tables.
3. EXPLAIN generates a results table with one row for each table in the SELECT statement.
4. The Address table is accessed via table scan. Of 603 rows read, 33% are selected.
5. The City table is accessed via primary index. One City row is read and selected for each Address row.

Feedback?

PARTICIPATION ACTIVITY 5.1.6: EXPLAIN statement.

The Flight table has 90 rows and primary key FlightNumber. No CREATE INDEX statements are executed on Flight. The questions below refer to the result table of the following statement:

```
EXPLAIN
  SELECT FlightNumber
  FROM Flight
  WHERE AirportCode IN
      (SELECT AirportCode
      FROM Airport
      WHERE Country = "Cuba");
```

1) In the row with table = "Airport", what is the value of select_type?

SUBQUERY

Check Show answer

Correct

SUBQUERY

The Airport table appears in the inner query, or subquery, of a nested query.

2) In the row with table = "Flight", what is the value of possible_keys?

NULL

Check Show answer

Correct

NULL

The Flight table appears in the outer query. The outer query WHERE clause specifies AirportCode values. Since the Flight table has no index on AirportCode, the outer query cannot use an index. So NULL appears in the possible_keys column of the Flight row.

3) Assume the Airport table contains an AirportCode for Cuba. In the row with table = "Flight", what is the value of rows?

90

Check Show answer

Correct

90 or ninety

Since the AirportCode column has no index, all 90 rows of Flight must be compared to the AirportCodes returned by the subquery.

Feedback?

Physical design process

A database administrator may take a simple approach to physical design for MySQL with InnoDB:

1. **Create initial physical design**. Create a primary index on primary keys and a secondary index on foreign keys. In MySQL with InnoDB, these indexes are created automatically for all tables. In other databases, this step is necessary for tables larger than roughly 100 kilobytes, but can be omitted for smaller tables.
2. **Identify slow queries**. The MySQL ***slow query log*** is a file that records all long-running queries submitted to the database. Identify slow queries by inspecting the log. Most other relational databases have similar query logs.
3. **EXPLAIN slow queries**. Run EXPLAIN on each slow query to assess the effectiveness of indexes. A high value for **rows** and a low value for **filtered** indicates either a table scan or an ineffective index.
4. **Create and drop indexes** based on the EXPLAIN result table. Consider creating an index when the **rows** value is high and the **filtered** value is low. Consider dropping indexes that are never used.
5. **Partition large tables**. If some queries are still slow after indexes are created, consider partitions. Partition when slow queries access a small subset of rows of a large table. The partition column should appear in the WHERE clause of slow queries. Often, a range partition is best.

Steps 2 through 5 are ongoing activities. As the database grows and usage increases, the database administrator periodically reviews the query log, runs EXPLAIN, and adjusts the physical design for optimal performance. Additional tuning techniques can be found in the "Exploring further" section.

The five steps above can be adapted to other databases and storage engines, but the details depend on supported table structures, index types, and partition types.

PARTICIPATION ACTIVITY 5.1.7: Physical design for MySQL with InnoDB.

1 2 3 4 5 2x speed

Address

AddressID	Street	District	CityID	PostalCode
1	1913 Hanoi Way	Nagasaki	463	35200
2	1121 Loja Avenue	California	449	17886
3	692 Joliet Street	Attika	38	83579
4	1566 Inegl Manor	Mandalay	349	53561

(603 rows total)

City

CityID	CityName
1	Abu Dhabi
2	Acua
3	Adana
4	Addis Abeba

(600 rows total)

slow query log

```
EXPLAIN SELECT District
        FROM Address, City
        WHERE Address.CityID = City.CityID AND CityName = "Aurora";
```

select_type	table	type	possible_keys	key	ref	rows	filtered

SIMPLE SIMPLE	City Address	ALL ref	PRIMARY idx_fx_city_id	NULL idx_fk_city_id	NULL sakila.city.city_id	600 1	10.00 100.00

(selected columns)

```
CREATE INDEX CityNameIndex
ON City (CityName);
```

select_type	table	type	possible_keys	key	ref	rows	fil
SIMPLE SIMPLE	City Address	ref ref	PRIMARY, CityNameIndex idx_fx_city_id	CityNameIndex idx_fk_city_id	const sakila.city.city_id	1 1	1 1

(selected columns)

Running EXPLAIN again shows the query uses CityNameIndex, reads only one row, and executes

Captions

1. The SELECT statement appears in the MySQL slow query log.
2. The database administrator runs EXPLAIN on the SELECT query.
3. The EXPLAIN result shows no index was used, many rows were read, and few rows were selected.
4. To speed up the SELECT query, the database administrator creates an index on the CityName column.
5. Running EXPLAIN again shows the query uses CityNameIndex, reads only one row, and executes faster.

Feedback?

PARTICIPATION ACTIVITY 5.1.8: Physical design process for MySQL.

1) An index never appears in the key column of the EXPLAIN result table for any slow query. Should this index always be dropped?

- ○ True
- ◉ False

Correct

An index that does not appear in the EXPLAIN result table for slow queries is often, but not always, dropped. The index might be used in queries that do not appear in the slow query log. Dropping the index might turn fast queries into slow queries.

2) In the EXPLAIN result table for one slow query, the key column is NULL, the filtered column is 1%, and the rows column is

Correct

A NULL value in the key column of the EXPLAIN result table indicates a table scan. Since only 1% of 10 million

10 million. Should an index be created?

- True
- False

rows are selected, the query usually benefits from an index.

3) Additional indexes never slow performance of SELECT queries.

- True
- False

Correct

Assuming the query processor always selects the best index, additional indexes never degrade performance of SELECT statements. However, additional indexes usually degrade performance of INSERT, UPDATE, and DELETE statements.

4) In step 1 of physical design for MySQL with InnoDB, the database administrator determines table structures.

- True
- False

Correct

In MySQL with InnoDB, all tables are automatically sorted on the primary key. Tables with no primary key are structured as a heap. Thus, the database administrator effectively determines table structures by creating primary keys.

Feedback?

Exploring further:

- MySQL InnoDB storage engine
- MySQL CREATE INDEX syntax
- MySQL EXPLAIN result table
- MySQL slow query log
- Optimizing MySQL queries with EXPLAIN

How was this section? **Provide feedback**