Bioinformatics Resources
- Structural Resources / SQL -

Lecture & Exercises
Prof. B. Rost, Dr. L. Richter
Institut für Informatik I12
# Preliminary Schedule

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
<th>Date</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>April 26&lt;sup&gt;th&lt;/sup&gt;</td>
<td>Intro, General Overview (1. sh.)</td>
<td>June 14&lt;sup&gt;th&lt;/sup&gt;</td>
<td>NoSql 2 (7.sh.)</td>
</tr>
<tr>
<td>May 3&lt;sup&gt;rd&lt;/sup&gt;</td>
<td>Sequence Databases (2. sh.)</td>
<td>June 21&lt;sup&gt;th&lt;/sup&gt;</td>
<td>No lecture</td>
</tr>
<tr>
<td>May 10&lt;sup&gt;th&lt;/sup&gt;</td>
<td>Sequence Databases (3. sh.)</td>
<td>June 28&lt;sup&gt;th&lt;/sup&gt;</td>
<td>PredictProtein MongoDB (8.sh.)</td>
</tr>
<tr>
<td>May 17&lt;sup&gt;th&lt;/sup&gt;</td>
<td>Structure Databases (4. sh.)</td>
<td>July 5&lt;sup&gt;th&lt;/sup&gt;</td>
<td>JavaScript (9.sh.)*</td>
</tr>
<tr>
<td>May 24&lt;sup&gt;th&lt;/sup&gt;</td>
<td>SQL (5. sh.)*</td>
<td>July 12&lt;sup&gt;th&lt;/sup&gt;</td>
<td>JavaScript / D3.js</td>
</tr>
<tr>
<td>May 31&lt;sup&gt;st&lt;/sup&gt;</td>
<td>No lecture</td>
<td>July 19&lt;sup&gt;th&lt;/sup&gt;</td>
<td>Wrap Up, Q&amp;A</td>
</tr>
<tr>
<td>June 7&lt;sup&gt;th&lt;/sup&gt;</td>
<td>SQL, NoSql (6. sh)</td>
<td>July 31&lt;sup&gt;st&lt;/sup&gt;</td>
<td>Exam, if the number of students is low, it can be an oral exam</td>
</tr>
</tbody>
</table>

* These exercises can earn you a bonus
Orga - Exam Date

- Exam scheduled for Wednesday, Jul 31\textsuperscript{st}
- Time: 10:30-12:00
- Room: MW 1450 Willy-Messerschmidt Zeichensaal (Mechanical Engineering Building)
- The room may change due to a low number!
- Registration is MANDATORY
Databases - SQL

- Overlap with database lecture
- “SQL crash course”
- no design theory
- no normalization
- standard books like:
More Books

Selected SQL Topics

- Table modifications
  - insert, update, create, alter

- Data retrieval and reporting/aggregation
  - select, average, sum

- Combination and Performance
  - join

- Access control and permissions
  - grant

- Backup and Restore / Input-output
Reasons for DBMS

- redundancy, consistency
- limited access
- difficult multi-user access
- loss of information
- loss of integrity
- security issues
- expensive application development
Abstraction layers

View 1

Logical Layer

View 2

Physical Layer
Various Data Models

- Network model
- Hierarchical model
- **Relational Model**
- XML schema
- Object-oriented model
- Deductive model
### Relational Model

<table>
<thead>
<tr>
<th>Students</th>
<th>Attends</th>
<th>Lectures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matric</td>
<td>Matric</td>
<td>LectureNo</td>
</tr>
<tr>
<td>123455</td>
<td>123455</td>
<td>2</td>
</tr>
<tr>
<td>233457</td>
<td>233457</td>
<td>5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Select** Name

**From** Students, Attends, Lectures

**Where** Students.Matric = Attends.Matric and Attends.LectureNo = Lectures.LectureNo and Lectures.Title = ‘Genomics’;

**Update** Lectures

**Set** Title = ‘Genomics of Mammalian’

**Where** LectureNo = 5;

BioinfRes SoSe 19 / 13
Entity Relationship Model

- Graphical Notation
- Models real world “entities” and “relation”
- allows for “attributes”
- allows for functionalities (1:1, 1:n, n:m)
- allows to define keys
- key: a set for attributes which values combination allow unambiguous instance identification
Notation

**Student**

*(strong)* Entity

**Name**

Attribute, key: underlined

**Attends**

Relation

**weak Entity (depend on others)**
Functionality

- Student
- Lecture
- Grade

Relationship:
- Attends:
  - N (many) to Lecture
  - M (many) to Student
  - N (many) to Grade
taken from Prof. Kempers database lecture WS 13/14
BioinfRes SoSe 19 / 18
Prüfungen als schwacher Entitytyp

Studenten

ablegen

Prüfungen

Note

PrüfTeil

MatrNr

N

umfassen

VorlNr

M

Vorlesungen

Professoren

PersNr

N

abhalten

N

taken from Prof. Kempers database lecture WS 13/14
SQL

- implemented by most available database management system manufacturer
- but: not always all specified features implemented
- not everything is specified!
- especially admin/server maintenance is often vendor specific

BioinfRes SoSe 19 / 20
SQL Data Types

- char
- varchar
- binary and varbinary
- blob and text
- numeric, decimal, integer (exact)
- approximate: float, double
SQL Data Types

- various formats for time and date
- enum: one out of a defined set
- set: zero or more items out of a predefined list

For more information see the live tour through
ACID-Principle for Transactions

- A: Atomicity: All-or-nothing, i.e. a sequence of operations is executed like a single atomic operation which cannot be interrupted
- C: Consistency: After every operation the database is consistent, i.e. all conditions and constraints about context and relationships are fulfilled
ACID-Principle II

- I: Isolation: Concurrent operations to not affect each other
- D: Durability: Upon successful completion of a transaction it is guaranteed that all modifications are persistent, i.e. they are stored in the database, even in case of an unexpected power loss.
Relational Algebra

- $\sigma$ Selection
- $\pi$ Projection
- $\rho$ Rename
- $\times$ Cross Product
- $\Join$ Join
- $-$ Difference
- $\div$ Division
Relational Algebra

- $\cup$ Union
- $\cap$ Intersection
- $\bowtie$ Semi Join (left)
- $\bowtie$ Left Outer Join
- $\bowtie$ (Full) Outer Join
## Demonstration Table

<table>
<thead>
<tr>
<th>gene</th>
<th>indiv</th>
<th>organism</th>
<th>function</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>cytox</td>
<td>1</td>
<td>mouse</td>
<td></td>
<td>prep</td>
</tr>
<tr>
<td>gapdh</td>
<td>1</td>
<td>human</td>
<td>glycolysis</td>
<td>completed</td>
</tr>
<tr>
<td>gapdh</td>
<td>2</td>
<td>human</td>
<td>glycolysis</td>
<td>completed</td>
</tr>
<tr>
<td>ttn</td>
<td>2</td>
<td>human</td>
<td>muscle</td>
<td>ongoing</td>
</tr>
<tr>
<td>unkno</td>
<td>3</td>
<td>human</td>
<td>NULL</td>
<td>prep</td>
</tr>
</tbody>
</table>
Selection

- The **SELECT** operation (denoted by $\sigma$ (sigma)) is used to select a subset of the tuples from a relation based on a **selection condition**
- It acts as a (row) filter
- Specified in the **WHERE**-clause
  - $\sigma$ status = “ongoing” (STATUS)
Selection

- General: the *select* operation is denoted by
  \[ \sigma_{\text{<selection condition>}}(R) \] where:
    - the \( \sigma \) (sigma) is used to denote the select operator
    - the selection condition is a Boolean (conditional) expression specified on the attributes of relation R
    - tuples that make the condition true are selected (appear in the result of the operation)
    - tuples that make the condition false are filtered out (discarded from the result of the operation)
Selection

- The Boolean expression specified in <selection condition> is made up of a number of clauses of the form:
  <attribute name> <comparison op> <constant value>
  or
  <attribute name> <comparison op> <attribute name>

- <attribute name> is the name of an attribute of R, <comparison op> id normally one of the operations {=,>,>=,<,<=,!=}

- Clauses can be arbitrarily connected by the Boolean operators and, or and not
Selection

- **NULL** is tested for with special operators
- Select $\sigma$ is commutative
- can be cascade of *select* operations of a conjunction of conditions:

  $$\sigma_{<\text{condition}_1>}(\sigma_{<\text{condition}_2>}(R)) = \sigma_{<\text{condition}_2>}(\sigma_{<\text{condition}_1>}(R))$$

  $$\sigma_{<\text{cond}_1>}(\sigma_{<\text{cond}_2>}(\sigma_{<\text{cond}_3>}(R))) = \sigma_{<\text{cond}_1> \ AND \ <\text{cond}_2> \ AND \ <\text{cond}_3>}(R)$$
Projection

- **PROJECT** Operation is denoted by $\pi$ (pi)
- use PROJECT to retrieve specific attributes of relation R
- It acts as a (column) filter of the tuples
- Example:
  $\pi_{\text{Gene, status}} (\text{STATUS})$
- Project removes duplicates which might occur (in SQL: SELECT DISTINCT instead of simple SELECT)
Single Expression vs. Sequence of Relational Operations

- To retrieve completed genes from our example:
  - Single expression:
    \[ \pi_{\text{gene, status}}(\sigma_{\text{status=completed}}(\text{STATUS})) \]
  - Sequence of operation:
    \[ \text{ALL_COMP} \leftarrow \sigma_{\text{status=completed}}(\text{STATUS}) \]
    \[ \text{RESULT} \leftarrow \pi_{\text{gene, status}}(\text{ALL_COMP}) \]
Rename

- $RENAME$ is denoted by $\rho$ (rho)
- In some cases, we may want to rename the attributes of a relation or the relation name or both
  - Useful when a query requires multiple operations
  - Necessary in some cases (see JOIN operation later)
RENAMEN

- RENAME operations $\rho$ can be expressed by any of the following forms:
  - $\rho_{S}(R)$ changes: the *relation name* only to $S$
  - $\rho_{(B_1, B_2, \ldots, B_n)}(R)$ changes: the *column (attribute)* names only to $B_1, B_1, \ldots, B_n$
  - $\rho_{S(B_1, B_2, \ldots, B_n)}(R)$ changes both: the relation name to $S$, *and* the column (attribute) names to $B_1, B_1, \ldots, B_n$
Relational Operators from Set Theory

- Union
- Intersection
- Minus
- Cartesian Products
Union

- It is a Binary operation, denoted by $\cup$
- The result of $R \cup S$, is a relation that includes all tuples that are either in $R$ or in $S$ or in both $R$ and $S$
- Duplicate tuples are eliminated
- $R$ and $S$ have to type compatible:
  - they must have the same number of attributes
  - corresponding attributes are type compatible
Intersection

- INTERSECTION is denoted by $\cap$
- The result of the operation $R \cap S$, is a relation that includes all tuples that are in both $R$ and $S$
- The attribute names in the result will be the same as the attribute names in $R$
- The two operand relations $R$ and $S$ must be “type compatible”
Set Difference

- SET DIFFERENCE (also called MINUS or EXCEPT) is denoted by –

- The result of R – S, is a relation that includes all tuples that are in R but not in S

- The attribute names in the result will be the same as the attribute names in R

- The two operand relations R and S must be “type compatible”
Properties of Union, Intersection and Difference

- Both union and intersection are commutative; that is:
  \[ R \cup S = S \cup R, \text{ and } R \cap S = S \cap R \]

- Union and intersection are associative operations; that is:
  \[ R \cup (S \cup T) = (R \cup S) \cup T \]
  \[ (R \cap S) \cap T = R \cap (S \cap T) \]

- The minus operation is not commutative; that is:
  \[ R - S \neq S - R \]
Cross Product (Cartesian Product)

- CROSS PRODUCT Operation
- Used to combine tuples from two relations in a combinatorial fashion
- Denoted by $R(A_1, A_2, \ldots, A_n) \times S(B_1, B_2, \ldots, B_m)$
- Result is a relation $Q$ with degree $n + m$ attributes:
  $Q(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$
Cartesian Product (Cross Product)

- The resulting relation contains every possible combination of the tuples from R and S -- one from R and one from S.

- Hence, if R has \( n_R \) tuples (denoted as \( |R| = n_R \)), and S has \( n_S \) tuples, then \( R \times S \) will have \( n_R \times n_S \) tuples.

- The two operands do NOT have to be "type compatible”

- Generally, CARTESIAN PRODUCT is not a meaningful operation, but can become meaningful when followed by other operations.
Join

- JOIN Operation (denoted by \( \bowtie \))
- Sequence of CARTESIAN PRODUCT followed by SELECT is used to identify and select related tuples from two relations
- very important for any relational database with more than a single relation, because it allows to combine related tuples from various relations
Join

- The general form of a join operation on two relations \( R(A_1, A_2, \ldots, A_n) \) and \( S(B_1, B_2, \ldots, B_m) \) is:
  \[ R \bowtie_{\text{<join condition>}} S \]

- where \( R \) and \( S \) can be any relations that result from general relational algebra expressions
Join

Consider the following JOIN operation:

- If $R(A_1, A_2, \ldots, A_n)$ and $S(B_1, B_2, \ldots, B_m)$
  Think about $R.A_i = S.B_j$

- Result is a relation $Q$ with degree $n + m$ attributes:
  $Q(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$

- The resulting relation state has one tuple for each combination of tuples – $r$ from $R$ and $s$ from $S$, but only if they satisfy the join condition $r[A_i] = s[B_j]$

- if $R$ has $n_R$ tuples, and $S$ has $n_S$ tuples, then the join result will generally have less than $n_R \times n_S$ tuples
Join (more precise)

- The general case of JOIN operation is called a Theta-join: \( R \bowtie_{\theta} S \)
- The join condition is called theta
- Theta can be any general boolean expression on the attributes of R and S; for example: 
  \( R.A_i < S.B_j \) AND (\( R.A_k = S.B_l \) OR \( R.A_p < S.B_q \))
Equijoin

- The most common use of join involves join conditions with equality comparisons only.
- Such a join, where only the comparison operator used is =, is called an EQUIJOIN.
- The JOIN seen in the previous example was an EQUIJOIN.
Natural Join

- Another variation of JOIN called NATURAL JOIN — denoted by * or $\bowtie$ without condition

- It was created to get rid of the second (superfluous) attribute in an EQUIJOIN condition.

- $Q \leftarrow R(A,B,C,D) \ast S(C,D,E)$

- implicit join condition includes each pair of attributes with the same name, “AND”ed together:
  - $R.C = S.C$ AND $R.D = S.D$

- keeps only one attribute of each such pair:
  - $Q(A,B,C,D,E)$
Semi Join

- acts like a filter based on a specified attribute
- $R \bowtie S$ means: if $R$ and $S$ have a common attribute $C$ the result are all tuples from $R$ which $C$ value occurs also in $S$, $n_Q \leq n_R$ tuples
- $Q <- R(A,B,C) \bowtie S (C,D,E)$
- $Q(A,B,C)$ with $n_R$ attributes
- $\pi_{A,B,C} (\sigma_{R.C=S.C}(R \times S))$
Left Outer Join

- Right version is analogous
- add information to corresponding left side tuples
- $R \bowtie S$ means: if $R$ and $S$ have a common attribute $C$ the result are all combined tuples from $R$ and $S$ where $R.C = S.C$ and in addition all remaining tuples from $R$, $n_Q = n_R$ tuples
- $Q \leftarrow R(A,B,C) \bowtie S (C,D,E)$
- $Q(A,B,C,D,E)$ with $n_{RUS}$ attributes
- if no matching tuples found in $S$ attributes $D$ and $E$ contain no values
(Full) Outer Join

- combines corresponding tuples from R and S where possible, else attributes left blank
- R S means: if R and S have a common attribute C the result are all combined tuples from R and S where R.C = S.C and in addition all remaining tuples from R and S, n_Q ≤ n_R+S tuples
- Q <- R(A,B,C) S (C,D,E)
- Q(A,B,C,D,E) with n_R∪S attributes
- if no matching tuples found in R or S attributes A, B or D and E contain no values
Division

- Gives all attribute tuple for R-S where a values for R-S co-occurs with all tuples in S
- R(A,B) and S(B)
- R÷S: Q(A) where each result tuple in Q can be found in R in combination with every tuple from S
Complete Set of Relational Operations

- The set of operations including SELECT $\sigma$, PROJECT $\pi$, UNION $\cup$, DIFFERENCE $-$, RENAME $\rho$, and CARTESIAN PRODUCT $X$ is called a complete set because any other relational algebra expression can be expressed by a combination of these five operations.

- Examples:
  - $R \cap S = (R \cup S) - ((R - S) \cup (S - R))$
  - $R \bowtie_{\text{join condition}} S = \sigma_{\text{join condition}} (R \times S)$
Beyond Classical Algebra

- Grouping: group by, having
- Sorting: order by
- Aggregation: count, sum, average, min, max
Keys and Indexes

- Each relation represents a subset of the cartesian product of its domains (attributes)
- Some values might be unique for a row others are not
- To address and access a specific tuple in a relation we need to define a primary key
- A primary key is set of attributes which combination allows us to unambiguously identify a certain row in the relation
Keys and Indexes

- **Consequences:**
  - Each primary key (combination) can occur only once in a table
  - Entries which miss even one of these attribute values are not allowed (NOT NULL)
  - Default values for these attributes make no sense
  - These system has to keep track of with the help of an index

- The key depends on the modeling and the domain
Indexes/Attribute Constraints

- **PRIMARY KEY**: UNIQUE, NOT NULL
- **UNIQUE**: If there is a value it must be unique, if there is no value but NULL it can occur multiple times
- **INDEX**: A search structure which allows to find tuples (rows) which a specific attribute value efficiently
  - must explicitly requested in the table structure
  - for character types you can the prefix length
Performance Considerations

- A simple example shows why indexes make sense:
- There are three relations to join A*B*C:
  - A(1,000,000 rows)
  - B(100 rows)
  - C (10,000 rows)
Performance Considerations

- (Worst) Case w/o indexes and bad execution sequence:
  A*C: 10,000,000,000 comparisons \(O(n \times m)\) -> D(10,000,000,000 rows)
  D*B(1,000,000,000,000 comparisons) \(O(n \times m)\)
  - of course tuples might be dropped in reality because of missing join partners and join conditions

- Case with indexes and clever sequence:
  B*A: 100 * \(\log(10,000,000)\) comparisons -> D (10,000,000 rows)
  C*D: 10,000 * \(\log(10,000,000)\) comparisons
Performance Considerations

- Sequence of evaluation can be optimized by the database engine
  - clever order with exploitation of associativity and commutativity
  - example: $100 \times \log(10,000,000)$ vs $10,000,000 \times \log(100)$
  - maybe not effective in worst case but definitely everytime else
SQL in use

- Syntax of MySQL server 5.6/7
- useful command line utilities: mysql, mysqladmin
- GUI: SQL Workbench
- frequent tasks / operations
- command line
- Python integration
Prerequisites

- Know your server version: Even if SQL is a standard, different vendors implement different versions of add vendor/version specific features
- Have client programs installed (mysql, mysqladmin)
- Have a language driver/connector installed
Prerequisites

- command line clients come typically with the installation
- GUI clients are also available
- Connectors come in different flavors for many languages and may have to be installed
- Connectors may offer different APIs
(User) Administration

- Administration information is stored in the database ‘mysql’
- Users have to connect to the database server
- Users are managed via accounts:
  - username
  - hostname / IP address
  - optional password
(User) Administration

- Example: ‘dowj’@’myhost’
- special hostnames: localhost, 127.0.0.1, ::1
- ‘empty’ user name: anonymous
- hostnames may contain wildcards: % or _
- omission of user or host name allowed
- password can be and is set individually for each name/host combination
Example

| user    | host              | password  |
|---------+-------------------+-----------|
| root    | localhost         | *CABC7    |
| root    | phoenix.fritz.box | *CABC7    |
| root    | 127.0.0.1         | *CABC7    |
| root    | ::1               | *CABC7    |
| richter | localhost         | *75B62    |
|         | phoenix.fritz.box |           |

- Set / change password:
  SET PASSWORD for ‘dowj’@’localhost’ = PASSWORD(‘cleartext_password’)

BioinfRes SoSe 19 / 66
More Administration

- default: only root account, might be unsecured
- databases: mysql, information_schema, performance schema, test
- further creation of user account and databases needed
More Administration

- use the mysqladmin tool to create a new, empty database:
  `mysqladmin -u root -p create resource_db`

- create a user (in a mysql session):
  `CREATE USER dowj [identified by 'clear_pw']`

- now you have to grant privileges on a certain database to the user:
  `GRANT ALL ON resource_db.* TO 'dowj'@'localhost'`
Structure of a Grant Clause

- which privilege (action) should be granted
- on which database
- on which table
- to whom: specification via user/host combination
- wild cards allowed
Full Grant Syntax 1

GRANT

    priv_type [(column_list)]
    [, priv_type [(column_list)]] ...

ON [object_type] priv_level

TO user_specification [, user_specification] ...

[REQUIRE {NONE | tsl_option [[AND] tsl_option] ...}]

[WITH {GRANT OPTION | resource_option} ...]

GRANT PROXY ON user_specification

    TO user_specification [, user_specification] ...

    [WITH GRANT OPTION]

object_type: {

    TABLE
    |
    | FUNCTION
    |
    | PROCEDURE

}
Full Grant Syntax 2

```
priv_level: {
    *
    | *
    | db_name.*
    | db_name.tbl_name
    | tbl_name
    | db_name.routine_name
}

user_specification:
    user [ auth_option ]

auth_option: {
    IDENTIFIED BY 'auth_string'
    | IDENTIFIED BY PASSWORD 'hash_string'
    | IDENTIFIED WITH auth_plugin
    | IDENTIFIED WITH auth_plugin AS 'hash_string'
```
Full Grant Syntax 3

tsl_option: {
    SSL
    | X509
    | CIPHER 'cipher'
    | ISSUER 'issuer'
    | SUBJECT 'subject'
}

resource_option: {
    | MAX_QUERIES_PER_HOUR count
    | MAX_UPDATES_PER_HOUR count
    | MAX_CONNECTIONS_PER_HOUR count
    | MAX_USER_CONNECTIONS count
}
# Privileges

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Meaning and Grantable Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL (PRIVILEGES)</td>
<td>Grant all privileges at specified access level except <code>GRANT OPTION</code></td>
</tr>
<tr>
<td>ALTER</td>
<td>Enable use of <code>ALTER TABLE</code>. Levels: Global, database, table.</td>
</tr>
<tr>
<td>ALTER ROUTINE</td>
<td>Enable stored routines to be altered or dropped. Levels: Global, database, procedure.</td>
</tr>
<tr>
<td>CREATE</td>
<td>Enable database and table creation. Levels: Global, database, table.</td>
</tr>
<tr>
<td>CREATE ROUTINE</td>
<td>Enable stored routine creation. Levels: Global, database.</td>
</tr>
<tr>
<td>CREATE TABLESPACE</td>
<td>Enable tablespaces and log file groups to be created, altered, or dropped. Level: Global.</td>
</tr>
<tr>
<td>CREATE TEMPORARY TABLES</td>
<td>Enable use of <code>CREATE TEMPORARY TABLE</code>. Levels: Global, database.</td>
</tr>
<tr>
<td>CREATE USER</td>
<td>Enable use of <code>CREATE USER</code>, <code>DROP USER</code>, <code>RENAME USER</code>, and <code>REVOKE ALL PRIVILEGES</code>. Level: Global.</td>
</tr>
<tr>
<td>CREATE VIEW</td>
<td>Enable views to be created or altered. Levels: Global, database, table.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Enable use of <code>DELETE</code>. Level: Global, database, table.</td>
</tr>
<tr>
<td>DROP</td>
<td>Enable databases, tables, and views to be dropped. Levels: Global, database, table.</td>
</tr>
<tr>
<td>EVENT</td>
<td>Enable use of events for the Event Scheduler. Levels: Global, database.</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>Enable the user to execute stored routines. Levels: Global, database, table.</td>
</tr>
<tr>
<td>FILE</td>
<td>Enable the user to cause the server to read or write files. Level: Global.</td>
</tr>
<tr>
<td>GRANT OPTION</td>
<td>Enable privileges to be granted to or removed from other accounts. Levels: Global, database, table, procedure, proxy.</td>
</tr>
<tr>
<td>INDEX</td>
<td>Enable indexes to be created or dropped. Levels: Global, database, table.</td>
</tr>
<tr>
<td>INSERT</td>
<td>Enable use of <code>INSERT</code>. Levels: Global, database, table, column.</td>
</tr>
<tr>
<td>LOCK TABLES</td>
<td>Enable use of <code>LOCK TABLES</code> on tables for which you have the <code>SELECT</code> privilege. Levels: Global, database.</td>
</tr>
<tr>
<td>PROCESS</td>
<td>Enable the user to see all processes with <code>SHOW PROCESSES</code>. Level: Global.</td>
</tr>
<tr>
<td>PROXY</td>
<td>Enable user proxying. Level: From user to user.</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>Enable foreign key creation. Levels: Global, database, table, column.</td>
</tr>
<tr>
<td>RELOAD</td>
<td>Enable use of <code>FLUSH</code> operations. Level: Global.</td>
</tr>
<tr>
<td>REPLICATION CLIENT</td>
<td>Enable the user to ask where master or slave servers are. Level: Global.</td>
</tr>
<tr>
<td>REPLICATION SLAVE</td>
<td>Enable replication slaves to read binary log events from the master. Level: Global.</td>
</tr>
<tr>
<td>SELECT</td>
<td>Enable use of <code>SELECT</code>. Levels: Global, database, table, column.</td>
</tr>
<tr>
<td>SHOW DATABASES</td>
<td>Enable use of <code>SHOW DATABASES</code> to show all databases. Level: Global.</td>
</tr>
<tr>
<td>SHOW VIEW</td>
<td>Enable use of <code>SHOW CREATE VIEW</code>. Levels: Global, database, table.</td>
</tr>
<tr>
<td>SHUTDOWN</td>
<td>Enable use of <code>mysqldadmin shutdown</code>. Level: Global.</td>
</tr>
<tr>
<td>SUPER</td>
<td>Enable use of other administrative operations such as <code>CHANGE MASTER TO</code>, <code>KILL</code>, <code>PURGE BINARY LOGS</code>, <code>SET GLOBAL</code>, and <code>mysqldadmin debug</code> command. Level: Global.</td>
</tr>
<tr>
<td>TRIGGER</td>
<td>Enable trigger operations. Levels: Global, database, table.</td>
</tr>
<tr>
<td>UPDATE</td>
<td>Enable use of <code>UPDATE</code>. Levels: Global, database, table, column.</td>
</tr>
<tr>
<td>USAGE</td>
<td>Synonym for &quot;no privileges&quot;</td>
</tr>
</tbody>
</table>

Useful / Important MySQL programs

- **mysqld**: database server (demon)
  - typically started via `mysqld_safe` or `mysql.server`
- **mysql_install_db**: initializes the MySQL data directory and the grant tables and executes once upon installation
- **client programs**: `mysql`, `mysqladmin`, `mysqldump`, `mysqlimport`, `mysqlshow`
- more program for logging and self-checks
mysqladmin

- tool to perform administrative tasks
- create a new database
- drop a database
- flush-commands (forced write to disc)
- retrieve status information
Hacking your MySQL database

- in case you forgot your database root account
- assumes you have admin privileges on your computer:
  - stop the server
  - restart the server with `--skip-grant-tables` and `--skip-working`
  - connect with the mysql client
  - enable privileges with `FLUSH PRIVILEGES`
Hacking your MySQL database

- depending on your version:
  - ALTER USER 'root'@'localhost'
    IDENTIFIED BY 'MyNewPass'; (version 5.7.6 and later)
  - SET PASSWORD FOR 'root'@'localhost'
    = PASSWORD('MyNewPass'); (version 5.7.5 and earlier)
  - stop and restart the server, now without the skip-clauses
mysqlshow


- access to various show commands like:
  - available databases
  - accessible tables
  - ...

- mysqlshow [options] [db_name [tbl_name [col_name]]]
mysqldump

- used to create a logical backup of a database
- output in sql, csv or xml format
- copy from one server to the other:
  `mysqldump --opt db_name | mysql --host=remote_host -C db_name`
mysqldump

● simple dump & restore:
  mysqldump db_name > backup-file.sql
  mysql db_name < backup-file.sql

● very easy, but:

● since you apply SQL commands to your database
  make sure they are save! Especially when you work
  as root!

● if you have a dump of the whole server, this includes
  the internal management tables which may override
  your own ones
mysqlimport


- interface to the `LOAD DATA INFILE` command

- `mysqlimport [options] db_name textfile1 [textfile2 ...]`

- data from `textfileX` is imported into table `textfileX`

- option `--local` allows to import local client data

- non-local load operations needs `FILE` privileges (on both the client and the server!)
LOAD DATA INFILE
mysql

- main client to interact with the database
- graphical alternatives are also available e.g. MySQLWorkbench
- used for most of the interactive work with the database
Important SQL Commands

- from https://dev.mysql.com/doc/refman/5.7/en/
- 2 sets of commands: DDL, DML
- Data Definition Language (DDL, affecting the tables structures):
  - to create and to modify the table features
  - create
  - drop
  - alter ...
Important SQL Command

- Data Manipulation Language (DML, affecting and accessing the table contents):
  - access the data
  - insert, update, delete
  - select
CREATE TABLE

CREATE TABLE

create_definition:
  col_name column_definition
  | [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,...)
  | [index_option] ...
  | {INDEX|KEY} [index_name] [index_type] (index_col_name,...)
  | [index_option] ...
  | [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY]
  | [index_name] [index_type] (index_col_name,...)
  | [index_option] ...
  | {FULLTEXT|SPATIAL} [INDEX|KEY] [index_name] (index_col_name,...)
  | [index_option] ...
  | [CONSTRAINT [symbol]] FOREIGN KEY
  | [index_name] (index_col_name,...) reference_definition
  | CHECK (expr)
CREATE TABLE

column_definition:

data_type [NOT NULL | NULL] [DEFAULT default_value]
[AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
[COMMENT 'string']
[COLUMN_FORMAT {FIXED|DYNAMIC|DEFAULT}]
[STORAGE {DISK|MEMORY|DEFAULT}]
[reference_definition]
| data_type [GENERATED ALWAYS] AS (expression)
| [VIRTUAL | STORED] [UNIQUE [KEY]] [COMMENT comment]
| [NOT NULL | NULL] [[PRIMARY] KEY]
CREATE TABLE

data_type:

<table>
<thead>
<tr>
<th>BIT[</th>
<th>length]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TINYINT[</td>
<td>length]</td>
<td>UNSIGNED</td>
</tr>
<tr>
<td>SMALLINT[</td>
<td>length]</td>
<td>UNSIGNED</td>
</tr>
<tr>
<td>MEDIUMINT[</td>
<td>length]</td>
<td>UNSIGNED</td>
</tr>
<tr>
<td>INT[</td>
<td>length]</td>
<td>UNSIGNED</td>
</tr>
<tr>
<td>INTEGER[</td>
<td>length]</td>
<td>UNSIGNED</td>
</tr>
<tr>
<td>BIGINT[</td>
<td>length]</td>
<td>UNSIGNED</td>
</tr>
<tr>
<td>REAL[</td>
<td>length,</td>
<td>decimals]</td>
</tr>
<tr>
<td>DOUBLE[</td>
<td>length,</td>
<td>decimals]</td>
</tr>
<tr>
<td>FLOAT[</td>
<td>length,</td>
<td>decimals]</td>
</tr>
<tr>
<td>DECIMAL[</td>
<td>length,</td>
<td>decimals]</td>
</tr>
<tr>
<td>NUMERIC[</td>
<td>length,</td>
<td>decimals]</td>
</tr>
<tr>
<td>DATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME[</td>
<td>fsp]</td>
<td></td>
</tr>
<tr>
<td>TIMESTAMPM[</td>
<td>fsp]</td>
<td></td>
</tr>
<tr>
<td>DATETIME[</td>
<td>fsp]</td>
<td></td>
</tr>
<tr>
<td>YEAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHAR[</td>
<td>length]</td>
<td>BINARY</td>
</tr>
<tr>
<td>[CHARACTER SET charset_name] [COLLATE collation_name]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARCHAR[</td>
<td>length]</td>
<td>BINARY</td>
</tr>
<tr>
<td>[CHARACTER SET charset_name] [COLLATE collation_name]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BINARY[</td>
<td>length]</td>
<td></td>
</tr>
<tr>
<td>VARBINARY[</td>
<td>length]</td>
<td></td>
</tr>
<tr>
<td>TINYBLOB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLOB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEDIUMBLOB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LONGBLOB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TINYTEXT [BINARY]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[CHARACTER SET charset_name] [COLLATE collation_name]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEXT [BINARY]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[CHARACTER SET charset_name] [COLLATE collation_name]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEDIUMTEXT [BINARY]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[CHARACTER SET charset_name] [COLLATE collation_name]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LONGTEXT [BINARY]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[CHARACTER SET charset_name] [COLLATE collation_name]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENUM(value1,value2,value3,...)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[CHARACTER SET charset_name] [COLLATE collation_name]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SET(value1,value2,value3,...)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[CHARACTER SET charset_name] [COLLATE collation_name]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JSON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>spatial_type</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CREATE TABLE

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

index_type:
    USING {BTREE | HASH}

index_option:
    KEY_BLOCK_SIZE [=] value
    | index_type
    | WITH PARSER parser_name
    | COMMENT 'string'

reference_definition:
    REFERENCES tbl_name (index_col_name,...)
    [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
    [ON DELETE reference_option]
    [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT
CREATE TABLE

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

index_type:
   USING {BTREE | HASH}

index_option:
   KEY_BLOCK_SIZE [=] value
   | index_type
   | WITH PARSER parser_name
   | COMMENT 'string'

reference_definition:
   REFERENCES tbl_name (index_col_name,...)
   [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
   [ON DELETE reference_option]
   [ON UPDATE reference_option]

reference_option:
   RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT
CREATE TABLE

table_option:

- ENGINE [ = ] engine_name
- AUTO_INCREMENT [ = ] value
- AVG_ROW_LENGTH [ = ] value
- [ DEFAULT ] CHARACTER SET [ = ] charset_name
- CHECKSUM [ = ] { 0 | 1 }
- [ DEFAULT ] COLLATE [ = ] collation_name
- COMMENT [ = ] 'string'
- COMPRESSION [ = ] { 'ZLIB' | 'LZ4' | 'NONE' }
- CONNECTION [ = ] 'connect_string'
- DATA DIRECTORY [ = ] 'absolute path to directory'
- DELAY_KEY_WRITE [ = ] { 0 | 1 }
- ENCRYPTION [ = ] { 'Y' | 'N' }
- INDEX DIRECTORY [ = ] 'absolute path to directory'
- INSERT_METHOD [ = ] { NO | FIRST | LAST }
- KEY_BLOCK_SIZE [ = ] value
- MAX_ROWS [ = ] value
- MIN_ROWS [ = ] value
- PACK_KEYS [ = ] { 0 | 1 | DEFAULT }
- PASSWORD [ = ] 'string'
- ROW_FORMAT [ = ] { DEFAULT | DYNAMIC | FIXED | COMPRESSED | REDUNDANT | COMPACT }
- STATS_AUTO_RECALC [ = ] { DEFAULT | 0 | 1 }
- STATS_PERSISTENT [ = ] { DEFAULT | 0 | 1 }
- STATS_SAMPLE_PAGES [ = ] value
- TABLESPACE tablespace_name [ STORAGE { DISK | MEMORY | DEFAULT } ]
- UNION [ = ] ( tbl_name [, tbl_name ] ... )
CREATE TABLE Examples

- Create a table with the same layout as an existing table:
  
  ```sql
  CREATE TABLE new_tbl LIKE orig_tbl;
  ```

- Create as a copy of another table:
  
  ```sql
  CREATE TABLE new_tbl AS SELECT * FROM orig_tbl;
  ```

- CREATE TABLE `test` (blob_col BLOB, INDEX(blob_col(10)));

- CREATE TABLE `animals` (id MEDIUMINT NOT NULL AUTO_INCREMENT, name CHAR(30) NOT NULL, PRIMARY KEY (id));
DROP TABLE

- quite simple:
  DROP [TEMPORARY] TABLE [IF EXISTS]
  tbl_name [, tbl_name] ...
  [RESTRICT | CASCADE]

- other DROP statements are analogous
ALTER TABLE

- ALTER TABLE tbl_name
  [alter_specification [, alter_specification] ...]
  [partition_options]

- alter_specifications: mostly of following type:
  - ADD
  - DROP
  - RENAME
INSERT

INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name
[PARTITION (partition_name,...)]
[(col_name,...)]
{VALUES | VALUE} ({expr | DEFAULT},...),(...),...
[ ON DUPLICATE KEY UPDATE
  col_name=expr
  [, col_name=expr] ... ]

INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name
[PARTITION (partition_name,...)]
SET col_name={expr | DEFAULT}, ...
[ ON DUPLICATE KEY UPDATE
  col_name=expr
  [, col_name=expr] ... ]
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name
[PARTITION (partition_name,...)]
[(col_name,...)]
SELECT ... 
[ ON DUPLICATE KEY UPDATE
col_name=expr
 [, col_name=expr] ... ]
INSERT Examples

- INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
- INSERT INTO tbl_name (a,b,c) VALUES(1,2,3), (4,5,6), (7,8,9);
UPDATE

- UPDATE t1 SET col1 = col1 + 1;
- UPDATE items,month SET items.price=month.price WHERE items.id=month.id;
Delete

DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name

[PARTITION (partition_name,...)]

[WHERE where_condition]

[ORDER BY ...]

[LIMIT row_count]

- DELETE FROM t1, t2 USING t1 INNER JOIN t2
  INNER JOIN t3 WHERE t1.id=t2.id AND t2.id=t3.id;

- DELETE t1 FROM t1 LEFT JOIN t2 ON t1.id=t2.id WHERE t2.id IS NULL;
Expression

```plaintext
expr:
  expr OR expr
| expr || expr
| expr XOR expr
| expr AND expr
| expr && expr
| NOT expr
| ! expr
| boolean_primary IS [NOT] {TRUE | FALSE | UNKNOWN}
| boolean_primary

boolean_primary:
  boolean_primary IS [NOT] NULL
| boolean_primary <=> predicate
| boolean_primary comparison_operator predicate
| boolean_primary comparison_operator {ALL | ANY} (subquery)
| predicate

comparison_operator: = | >= | > | <= | < | <> | !=
```

```plaintext
bit_expr:
  bit_expr | bit_expr
| bit_expr & bit_expr
| bit_expr << bit_expr
| bit_expr >> bit_expr
| bit_expr + bit_expr
| bit_expr - bit_expr
| bit_expr * bit_expr
| bit_expr / bit_expr
| bit_expr DIV bit_expr
| bit_expr MOD bit_expr
| bit_expr % bit_expr
| bit_expr ^ bit_expr
| bit_expr + interval_expr
| bit_expr - interval_expr
| simple_expr

simple_expr:
  literal
| identifier
| function_call
| simple_expr COLLATE collation_name
| param_marker
| variable
| simple_expr || simple_expr
| + simple_expr
| - simple_expr
| simple_expr
| simple_expr
| BINARY simple_expr
| (expr [, expr] ...)
| ROW (expr, expr [, expr] ...)
| (subquery)
| EXISTS (subquery)
| {identifier expr}
| match_expr
| case_expr
| interval_expr
```
SELECT Examples

- SELECT * FROM t1 INNER JOIN t2 ...
- SELECT t1.*, t2.* FROM t1 INNER JOIN t2 ...
- SELECT AVG(score), t1.* FROM t1 ...
- SELECT CONCAT(last_name,' ',first_name) AS full_name
  FROM mytable ORDER BY full_name;
- SELECT t1.name, t2.salary FROM employee AS t1, info AS t2
  WHERE t1.name = t2.name;
- SELECT t1.name, t2.salary FROM employee t1, info t2
  WHERE t1.name = t2.name;
SELECT Examples / GROUP/ORDER BY

- SELECT college, region, seed FROM tournament ORDER BY region, seed;

- SELECT college, region AS r, seed AS s FROM tournament ORDER BY r, s [ASC];

- SELECT a, COUNT(b) FROM test_table GROUP BY a DESC;

- SELECT COUNT(col1) AS col2 FROM t GROUP BY col2 HAVING col2 = 2;

- SELECT user, MAX(salary) FROM users GROUP BY user HAVING MAX(salary) > 10;
JOIN

```
table_references:
    escaped_table_reference [ , escaped_table_reference ] ... 

escaped_table_reference:
    table_reference
    | \{ OJ table_reference \}

table_reference:
    table_factor
    | join_table

table_factor:
    tbl_name [ PARTITION (partition_names) ]
    | [ AS ] alias [ index_hint_list ]
    | table_subquery [ AS ] alias
    | ( table_references )

join_table:
    table_reference [ INNER | CROSS ] JOIN table_factor [ join_condition ]
    | table_reference STRAIGHT_JOIN table_factor
    | table_reference STRAIGHT_JOIN table_factor ON conditional_expr
    | table_reference ( LEFT | RIGHT ) [ OUTER ] JOIN table_reference join_condition
    | table_reference NATURAL ( { LEFT | RIGHT | [ OUTER ] } ) JOIN table_factor

join_condition:
    ON conditional_expr
    | USING ( column_list )

index_hint_list:
    index_hint [ , index_hint ] ... 

index_hint:
    USE \{ INDEX | KEY \}
    | [ FOR { JOIN | ORDER BY | GROUP BY } ] ( [ index_list ] )
    | IGNORE \{ INDEX | KEY \}
    | [ FOR { JOIN | ORDER BY | GROUP BY } ] ( index_list )
    | FORCE \{ INDEX | KEY \}
    | [ FOR { JOIN | ORDER BY | GROUP BY } ] ( index_list )

index_list:
    index_name [ , index_name ] ... 
```
JOIN Examples

- SELECT * FROM t1 LEFT JOIN (t2, t3, t4)
  ON (t2.a = t1.a AND t3.b = t1.b AND t4.c = t1.c)

- SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3
  CROSS JOIN t4)
  ON (t2.a = t1.a AND t3.b = t1.b AND t4.c = t1.c)

- SELECT t1.name, t2.salary
  FROM employee AS t1 INNER JOIN info AS t2 ON
  t1.name = t2.name;

- SELECT t1.name, t2.salary FROM employee t1
  INNER JOIN info t2 ON t1.name = t2.name;
JOIN Examples

- a LEFT JOIN b USING (c1, c2, c3)

- Does not work:
  SELECT * FROM t1, t2 JOIN t3 ON (t1.i1 = t3.i3);

- Confusing:
  SELECT * FROM (t1, t2) JOIN t3 ON (t1.i1 = t3.i3);

- Better:
  SELECT * FROM t1 JOIN t2 JOIN t3 ON (t1.i1 = t3.i3);
DESCRIBE, SHOW, EXPLAIN, LIMIT

● DESCRIBE/EXPLAIN (synonymous):
  - DESCRIBE: information about a table definition
  - EXPLAIN: execution plan info

● SHOW: command used to create a table (https://dev.mysql.com/doc/refman/5.7/en/show-create-table.html)

● in combination with a SELECT statement you can limit the number of output rows (https://dev.mysql.com/doc/refman/5.7/en/limit-optimization.html)