Bioinformatics Resources
- SQL -

Lecture & Exercises
Prof. B. Rost, Dr. L. Richter, J. Reeb
Institut für Informatik I12

BioinfRes SS 15
Orga - Exam Date/Change?

- Change confirmed: Wednesday, Jul 22\textsuperscript{nd}, in the range of 15-17 o’clock
- Interimshörsaal 2
Databases - SQL

- Overlap with database lecture
- “SQL crash course”
- no design theory
- no normalization
- standard books like:
  - Dr. Bernhard Chen Ph.D., University of Central Arkansas
More Books

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></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Matric</td>
<td>Name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>123455</td>
<td>Mayer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>233457</td>
<td>Huber</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Lectures</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LectureNo</td>
<td>Title</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Bioinformatics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Genomics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<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 SS 15
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

BioinfRes SS 15
**Notation**

- **Student** (strong) Entity
- **Name** Attribute, key: underlined
- **Attends** Relation
- **weak Entity (depend on others)**
Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality

Funtionality
taken from Prof. Kempers database lecture WS 13/14
BioinfRes SS 15
Prüfungen als schwacher Entitytyp

Studenten

ablegen

1

Prüfungen

N

Note

PrüfTeil

MatrNr

VorlNr

umfassen

N

N

N

M

M

M

Vorlesungen

Professoren

PersNr

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

- implemented by most available dbms manufacturer
- but: not always all specified features implemented
- not everything is specified!
- especially admin/server maintenance is often vendor specific
SQL Data Types

- char
- varchar
- binary and varbinary
- blob and text
- numeric, decimal, integer (exact)
- approximate: float, double
- come in different sizes
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 fullfilled.
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

BioinfRes SS 15
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
- $\sigma$ \text{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 σ is commutative
- can be cascade of *select* operations of a conjunction of conditions:
  \[
  \sigma_{<\text{condition}_1>}(\sigma_{<\text{condition}_2>}(R)) = \sigma_{<\text{condition}_2>}(\sigma_{<\text{condition}_1>}(R))
  \]
  \[
  \sigma_{<\text{cond}_1>}(\sigma_{<\text{cond}_2>}(\sigma_{<\text{cond}_3>}(R))) = \sigma_{<\text{cond}_1> \text{ AND } <\text{cond}_2> \text{ AND } <\text{cond}_3>}(R)
  \]
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:
    \[ \pi_{\text{gene, status}}(\sigma_{\text{status}=\text{completed}}(\text{STATUS})) \]
  - Sequence of operation:
    \[ \text{ALL\_COMP} \leftarrow \sigma_{\text{status}=\text{completed}}(\text{STATUS}) \]
    \[ \text{RESULT} \leftarrow \pi_{\text{gene, status}}(\text{ALL\_COMP}) \]
Rename

- \textit{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)
RENAME

- **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:
  $$R \cup S = S \cup R,$$
  $$R \cap S = S \cap R$$

- 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 ⋈)
- 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:
  \[ R \Join_{\text{join condition}} S \]
- 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 $\bowtie_{R.A_i=S.B_j}$

- 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: $R.A_i < S.B_j \text{ AND } (R.A_k = S.B_l \text{ OR } R.A_p < S.B_q)$
Equijoin

- The most common use of join involves join conditions with equality comparisons only.
- Such a join, where the only 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:
  \( R.C=S.C \text{ AND } R.D = S.D \)

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

BioinfRes SS 15
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
- $Q \leftarrow R(A,B,C) \bowtie S (C,D,E)$
- $Q(A,B,C)$ 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

- \( R \bowtie_S S \) 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, \( n_Q \leq n_{R+S} \) tuples

- \( Q \leftarrow R(A,B,C) \bowtie_S (C,D,E) \)

- \( Q(A,B,C,D,E) \) with \( n_{R \cup S} \) 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 value 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 $\setminus$, RENAME $\rho$, and CARTESIAN PRODUCT $\times$ 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
- 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 at one of these attribute values are not allows (NOT NULL)
  - Default values for these attributes make no sense
  - These system has to keep track which the help of an index

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

- 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 sequence:
  A*C: 10,000,000,000 comparisons $O(n \times m) \rightarrow\n  D(10,000,000,000$ 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

- 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