|  |  |  |  |
| --- | --- | --- | --- |
| **Begriffe** | | | |
| Assoziation / Beziehungstyp | Verknüpft mehrere Entitätstypen. Binäre Assoziationen verknüpfen zwei Entitätstypen; n-wertige Assoziationen verknüpfen n Entitätstypen. | | |
| Attribut | Beschreibt eine Eigenschaft einer Entität oder Assoziation. Das Attribut besteht aus einem Namen und einem Datentyp | | |
| Attributwert | Ist der Inhalt/Wert eines Attributes. («Fritz» für das Attribute name) | | |
| Data Dictionary, Datenkatalog, Metadatenbank | Inhaltsverzeichnis des DBS. Enthält Informationen über Datenbasis (Tabellen, Views,Attribute,Relationen/Beziehungen, Benutzerrechte) | | |
| Datenbasis | Die eigentlichen physisch, persistent gespeicherten Anwendungsdaten, kurz Daten, des DBS. | | |
| Datenbanksystem | Besteht aus Datenbankmanagementsystem und einer oder mehreren Datenbasen (DB’s) | | |
| Entität | Eindeutig identifizierbares Objekt der realen Welt (z.B Der Angestellte Fritz Meier) | | |
| Entitätsmenge | Gleichartige Menge von Entitäten (z.B Alle Angestellte aus Zürich) | | |
| Entitätstyp | Die Menge aller möglichen gleichartig strukturierten Entitäten. Jeder Entitätstyp hat einen eindeutigen Namen und eine Mengen von Attributen. (z.B Angestellter) | | |
| Kanonische Lösung | Keine Nullwerte | | |
| Schwacher Entitätstyp | Ein Entitätstyp dessen Primärschlüssel von der anderen Entitäten und deren Primärschlüssel abhängig ist | | |
| Relation/Beziehung | Verknüpfung mehrerer Entitäten miteinander. (z.B 'Fritz Müller' gehört zur Abteilung 'Entwicklung') | | |
| Surrogatschlüssel | Besteht ein Primärschlüssel aus mehreren Attributen, kann auf eine Entität nicht mehr Eindeutig Zugegriffen werden ohne alle Attribute anzugeben. Zudem wird der Index weniger effizient. Eine Lösung stellt hierbei ein künstlicher Primärschlüssel als fortlaufende Nummer dar. (Anwendungsintern) | | |
| **ANSI 3-Ebenen Modell** | | | |
| **1) Externe Ebene:** Benutzersicht auf DB. SQL oder über GUI **2) Logische, konzeptionelle Ebene:** Beschreibt Daten in Gesamtheit. Beziehungen, Integritätsbedinungen, Zugriffsbedingungen. **3) Interne, physiche Ebene:** Speicherstrukturen. Wie und wo Daten in der DB gespeichert werden. Effizienter Zugriff auf Daten (Index, B-Tree, Zugriffsrechte)" | | | |
| **Benutzer (analog zu ANSI 3-Ebenen Modell)** | | | |
| **DBA:** Installation, Wartung, Backup, Recovery, Speicherplatz, Benutzer, Laden von Daten  **DB Entwickler:** Design und Entwicklung, Datenmodell, Masken, Applikation, SQL  **DB Benutzer:** Benutzt Applikation und führt allenfalls einfache SQL aus | | | |
| **Vorteile DBS gegenüber Dateiablage** | | | |
| - kontrollierbare Datenintegrität/Constraints  - Transaktionen  - Mehrbenutzerbetrieb (Sessionmngmt.) | | | - Kapselung der Daten  - Abfrage Sprache (SQL)  - Zugriffskontrolle / Sicherheit / Authent. |
| **ACID** | | | |
| **Atomicity:** Transaktion wird ganz oder gar nicht ausgeführt **Consistency:** Nach Transaktion sind Daten konsistent **Isolation:** Gleichzeitige Ausführungen beeinflussen sich nicht **Durability:** Auswirkungen auf Daten bleiben dauerhaft | | | |
| **Indexe** | | | |
| **B-Tree, B+Tree (Balanced)** | | Grosse Datenmengen die häufig ändern (schnell) | |
| **ISAM (Index Sequential Access Method)** | | Einfügen und suchen schnell Aktualisieren langsam | |
| **HASH** (B-Tree unterlegen) | | Nicht im WAL-> Kaputt bei Crash | |
| **BRIN (Block Range Index)** | | Sehr kleiner Index | |
| **GIST / GIN** | | Volltextsuchen | |
| **Normalformen (Vorhergehende Normalformen werden immer vorausgesetzt)** | | | |
| **1. Normalform:** Daten sind Atomar (keine mengenwertige Werte), Komalisten in mehrere Datensätze **2. Normalform:** Festlegen, welche Attribute von welchem Primärschlüssel abhängig sind. Jedes Nichtschlüsselattribut voll funktional abhängig vom einem Primarschlüssel. Keine Abhängigkeit zu einem Teilschlüssel.  **3. Normalform:** Auslagern in mehrere Tabellen mit Relationen (Kein Nichtschlüsselattribut transitiv abhängig (=über zwei Schlüssel abhängig) von einem Schlüsselattribut) **Boyce-Codd Normalform:** Aufteilen in Tabellen, damit kein Attribut nur von einem Teil einer zusammengesetzten ID abhängig ist. **4. Normalform:** Keine Daten ohne Zusammenhang in einer Tabelle **5. Normalform:** Nur eine Abbildung pro Tabelle | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Relationale Algebra** | | | |
| **Durchschnitt ∩** |  | **Vereinigung ∪** |  |
| **Differenz \** |  | **Projektion ∏ (SELECT DISTINCT)** |  |
| **Kartesisches Produkt X** |  | **Selektion σ (WHERE)** |  |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SQL** | | | | | | | | | | |
| **DDL: Data Definition Language** | | | | | | | | | | |
| **CREATE DATABSE, TABLE, INDEX, VIEW, Temporary Table** | | | | | | | | | | |
| **CREATE** DATABASE **[**dbname**]** **WITH** OWNER **=** 'name' | | | | | | | | | | |
| **CREATE** **TYPE** sex\_type **AS** ENUM **(**'male'**,** 'female'**,** 'trans'**);**  **CREATE** **TABLE** **[**tablename**]** **(**  ID SERIAL **PRIMARY** **KEY,**  sex sex\_type**,**  employeeno **INTEGER,**  age **INTEGER** **CHECK** age **>** 18  name **VARCHAR(**255**)** **UNIQUE** **NOT** **NULL,**  startdate **DATE** **DEFAULT** **CURRENT\_DATE,**  insertdate **TIMESTAMP** **NOT** **NULL** **DEFAULT** **CURRENT\_TIMESTAMP,**  paid **BOOLEAN** **default** **FALSE,**  price **REAL** **NOT** **NULL,** -- that's a float value  **FOREIGN** **KEY** **(**employeeno**)** **REFERENCES** person**(**personid**)**  **[ON** **DELETE** **{CASCADE|RESTRICT|NO** ACTION**|SET** **NULL|SET** **DEFAULT}]);** | | | | | | | | | | |
| **CREATE** **INDEX** **[**indexname**]** **ON** **[**tablename**]** **{(**col**)** **|** **(**col1**,**col2**)** **|** **(**immutable\_function**(**col**))** **|** **[USING** **{**btree**|**gist**|...}(**col**)]}**  **[WHERE** condition**];** | | | | | | | | | | |
| **CREATE** **VIEW** **[**viewname**]** **AS** **SELECT** name**,** **date,** age **FROM** **[**t/v**]** **WHERE** age **>** 18 | | | | | | | | | | |
| **CREATE** RULE kurs\_update **AS** **ON** **UPDATE** **TO** kursuebersicht  **DO** INSTEAD **UPDATE** kurs **SET** beschreibung **=NEW.**beschreibung  **WHERE** kursnummer **=** **NEW.**kursnummer**;** | | | | | | | | | | |
| **CREATE** **TEMPORARY** **TABLE** **[**tablename**]** **AS** **SELECT** **\*** **FROM** B | | | | | | | | | | |
| **ALTER TABLE/COLUMN** | | | | | | | | | | |
| **ALTER** **TABLE** **[**tablename**]** **ADD** **COLUMN** **[**colname**]** **varchar(**30**);** | | | | | | | | | | |
| **ALTER** **TABLE** **[**tablename**]** **DROP** **COLUMN|CONSTRAINT** **[**name**];** | | | | | | | | | | |
| **ALTER** **TABLE** **[**tablename**]** **RENAME** **[**colname**]** **TO** **[**new\_name**];** | | | | | | | | | | |
| **ALTER** **TABLE** **[**tablename**]** **ALTER** **COLUMN** **[**colname**]** **TYPE** **integer;** | | | | | | | | | | |
| **ALTER** **TABLE** **[**tablename**]** **ALTER** **COLUMN** **[**colname**]** **TYPE** sex\_type **USING** **[**colname**]::**sex\_type**;** | | | | | | | | | | |
| **ALTER** **TABLE** **[**tablename**]** **ADD** **CONSTRAINT** fk\_one **FOREIGN** **KEY** **(**col**)** **REFERENCES** B**(column)** **ON** **DELETE** **CASCADE;** | | | | | | | | | | |
| **DQL: Data Query Language** | | | | | | | | | | |
| **SELECT** **DISTINCT** sex**,** age **FROM** **[**tablename**]** **WHERE** age **>** 18 **AND** name **LIKE** '%name%' **AND** **MAX(**price**)** **GROUP** **BY** sex**,** age **ORDER** **BY** age **DESC** **LIMIT** 1**;** | | | | | | | | | | |
| **WINDOW FUNCTIONS** | | | | | | | | | | |
| **SELECT** persnr**,** abtnr**,** **avg(**salaer**)** OVER **(PARTITION** **BY** abtnr**)**  **FROM** angestellter**;** --compare empl. salary with average of department | | | | | | | | | | |
| row\_number() | | | Number of the current row within its partion (start 1) | | | | | | | |
| rank() | | | Rank of the current row with gaps | | | | | | | |
| ntile(num) | | | Integer ranging from 1 to the argument value | | | | | | | |
| lag(offset) | | | Value at the row that is offset rows before the current row | | | | | | | |
| lead(offset) | | | Value at the row that is offset row after the current row | | | | | | | |
| first\_value(val) | | | Value at the row that is the first row of the window frame | | | | | | | |
| **SUBQUERIES (Korreliert, Unkorreliert)** | | | | | | | | | | |
| **SELECT** Name**,** Description **FROM** Products p **WHERE** Quantity **<** 2 **\*** **(**  **SELECT** **AVG(** Quantity **)** **FROM** SalesOrderItems s **WHERE** p**.**ID**=**s**.**ProductID **);** | | | | | | | | | | |
| **SELECT** Name**,** Description **FROM** Products **WHERE** Quantity **<** 2 **\*** **(**  **SELECT** **AVG(** Quantity **)** **FROM** SalesOrderItems **);** --unkorreliert | | | | | | | | | | |
| **EXISTS / IN** | | | | | | | | | | |
| **SELECT** **\*** **FROM** Kunden **WHERE** **EXISTS** **(SELECT** **\*** **FROM** Aufträge **WHERE** Kunden**.**Kun\_Nr **=** Auftraege**.**Kun\_Nr**);** | | | | | | | | | | |
| **SELECT** **\*** **FROM** Kunden **WHERE** Kun\_Nr **IN** **(SELECT** Kun\_Nr **FROM** Auftraege**);** | | | | | | | | | | |
| **ALL / ANY, SOME** | | | | | | | | | | |
| **SELECT** **\*** **FROM** angestellter **WHERE** gehalt **<** **{ALL|ANY,SOME}** **(SELECT** gehalt **FROM** angestellter**);** -- any=some -> at least one row must match | | | | | | | | | | |
| **UNION (OR), INTERSECT (AND), EXCEPT (OHNE)** | | | | | | | | | | |
| SELECT ID, Kennzeichen, Farbe FROM Dienstwagen  {UNION [ALL]|INTERSECT [ALL]|EXCEPT [ALL]} -- ALL= Incl. duplicate  SELECT ID, Kennzeichen, Farbe FROM Fahrzeug; | | | | | | | | | | |
| **HAVING** | | | | | | | | | | |
| **SELECT** ID**,** name**,** age**,** address**,** salary **FROM** customers  **GROUP** **BY** age **HAVING** **COUNT(**age**)** **>=** 2**;** | | | | | | | | | | |
| **CTE : COMMON TABLE EXPRESSION, WITH (Vereinfachung komplexer Queries)** | | | | | | | | | | |
| Wiederverwendbar innerhalb einer Abfrage, rekursiv, Ersatz für Views | | | | | | | | | | |
| **WITH** regional\_sales **AS** **(**  **SELECT** region**,** **SUM(**amount**)** **AS** total\_sales **FROM** orders **GROUP** **BY** region  **),** top\_regions **AS** **(**  **SELECT** region **FROM** regional\_sales **WHERE** total\_sales **>**  **(SELECT** **SUM(**total\_sales**)/**10 **FROM** regional\_sales**)**  **)** **SELECT** region**,** product**,** **SUM(**quantity**)** **AS** product\_units**,** **SUM(**amount**)** **AS** product\_sales  **FROM** orders  **WHERE** region **IN** **(SELECT** region **FROM** top\_regions**)**  **GROUP** **BY** region**,** product**;** | | | | | | | | | | |
| **WITH RECURSIVE** | | | | | | | | | | |
| **WITH** **RECURSIVE** unter **(** persnr**,** name**,** chef **)** **AS**  **(SELECT** A**.**persnr**,** A**.**name**,** A**.**chef **FROM** angestellter A  **WHERE** A**.**chef **=** 1010 **UNION** **ALL**  **SELECT** A**.**persnr**,** A**.**name**,** A**.**chef **FROM** angestellter A  **INNER** **JOIN** unter B **ON** B**.**persnr **=** A**.**chef**)**  **SELECT** **\*** **FROM** unter **ORDER** **BY** chef**,**persnr**;** | | | | | | | | | | |
| **NULL, CAST/CASE, COALESCE (Nullwerte ersetzen)** | | | | | | | | | | |
| **SELECT** **\*** **FROM** A **WHERE** A**.**col **IS** **NOT** **NULL** | | | | | **SELECT** **CAST(CASE** **WHEN** paid **IS** **NULL** **THEN** 0 **ELSE** 1 **END** **AS** **BOOLEAN)** **FROM** A | | | | | |
| **select** **avg(**r**.**date**::timestamp** **-** r**.**date**::timestamp)** **from** rueckgabe | | | | | | | | | | |
| **SELECT** **COALESCE(**firstname**,**'no first name'**)**-- Ersetzt null Werte mit dem jeweilig folgenden Argument: COALESCE(firstname, company, title); | | | | | | | | | | |
| **SELECT(null** **=** **null);** --eine leere Zeile ‘unknown’, da Dreiwertigkeitslogik | | | | | | | | | | |
| **DML: Data Manipulation Language** | | | | | | | | | | |
| **INSERT** **INTO** **[**tablename**](**col1**,** col2**)** **VALUES** **(**1**,** 'Test'**);** | | | | | | | | | | |
| **UPDATE** **[**tablename**]** **SET** col1**=**5000 **WHERE** col1 **=** 1**;** | | | | | | | | | | |
| **DELETE** **FROM** **[**tablename**]** **WHERE** col1 **<** 8000 RETURNING **\*;** | | | | | | | | | | |
| **TRUNCATE** **[**tablename**];** -- performanter wie DELETE | | | | | | | | | | |
| **DCL: Data Control Language** | | | | | | | | | | |
| **CREATE** ROLE **[user]** **WITH** LOGIN PASSWORD **=** 'secure' | | | | | | | | | **DROP** ROLE **[**rolename**]** | |
| **CREATE** ROLE **[group]** | | | | | | | | |
| **ALTER** ROLE **[**role**]** **WITH** PASSWORD 'new\_secure' –**-** **set** **new** password | | | | | | | | | | |
| **ALTER** ROLE **[**role**]** **WITH** **{**SUPERUSER**|**CREATEDB**|**CREATEROLE**|**CREATEUSER**}** | | | | | | | | | | |
| **GRANT** **[group]** **TO** **[user]** -- add user to group | | | | | | | | | | |
| **GRANT** **{SELECT|INSERT|UPDATE|DELETE|ALL}** **ON** **TABLE** **[**tablename**]** **TO** **{user|group|PUBLIC}** **[WITH** **GRANT** **OPTION]** | | | | | | | | | | |
| **REVOKE** **{SELECT|INSERT|UPDATE|DELETE|ALL}** **ON** **TABLE** **[**tablename**]** **FROM** **{user|group|PUBLIC}** **[CASCADE|RESTRICT]** | | | | | | | | | | |
| REVOKE CASCADE entzieht Rechte auch bei 3th Party. Wurden Rechte weitervergeben und es wird kein CASCADE angegeben, schlägt REVOKE fehl! | | | | | | | | | | |
| Berechtigungen können auf Tabellen, Columns, Views, Sequenzen, Datenbanken, Schemas oder Tablespaces vergeben werden. | | | | | | | | | | |
| **Joins** | | | | | | | | | | |
| Inner Join **SELECT** **\*** **FROM** A **INNER** **JOIN** B **ON** A**.Key** **=** B**.Key** | | | Gibt alle Zeilen zurück, die eine Übereinstimmung in beiden Tabellen haben | | | | | | |  |
| Left Join **⋉** **SELECT** **\*** **FROM** A **LEFT** **JOIN** B **ON** A**.Key** **=** B**.Key** | | | Gibt alle Zeilen der linken Tabelle und alle Übereinstimmungen in der Rechte Tabelle zurück | | | | | | |  |
| Right Join **⋊** **SELECT** **\*** **FROM** A **RIGHT** **JOIN** B **ON** A**.Key** **=** B**.Key** | | | Gibt alle Zeile der rechten Tabelle und alle Übereinstimmungen in der Linken Tabelle zurück | | | | | | |  |
| Outer Join / Full Join **⋈** **SELECT** **\*** **FROM** A **OUTER** **JOIN** B **ON** A**.Key** **=** B**.Key** | | | Gibt alle Zeilen zurück wenn es eine Übereinstimmung in einer der Tabellen gibt | | | | | | |  |
| Excluding Join | | | **SELECT \* FROM** A **INNER** **JOIN** B **ON** A**.Key** **=** B**.Key** **WHERE** B**.KEY** **IS** **NULL** | | | | | | | |
| **Aggregatsfunktionen** | | | | **Skalare Funktionen** | | | | | | |
| AVG(col) | Durchschnitt | | | UPPER(col) | | | | | Uppercase | |
| COUNT(col) | Anzahl Zeilen | | | LOWER(col) | | | | | Lowercase | |
| FIRST(col) | Erster Wert | | | SUBSTR(col, start, length) | | | | | Substring | |
| LAST(col) | Letzter Wert | | | LENGTH(col) | | | | | Lenge eines Textfeldes | |
| MAX(col) | Grösster Wert | | | ROUND(col, 2) | | | | | Rundet Dezimalstellen | |
| MIN(col) | Kleinster Wert | | | NOW() | | | | | Aktuelle Systemzeit | |
| SUM(col) | Summe | | | **FORMAT(**'%s'**,** NOW**())** | | | | | | |
| **Konvertieren** | | | | | | | | | | |
| Number 🡪 String | | **to\_char(**125**,** '999'**);** -- 9=Wildcard | | | | | | | | |
| String 🡪 Date | | **to\_date(**'01.01.2016'**,** 'DD.MM.YYYY'**);** | | | | | | | | |
| String 🡪 Timestamp | | **to\_timestamp(**'01.01.2016 16:00:00'**,** 'DD.MM.YYYY HH24:MI:SS'**)** | | | | | | | | |
| String 🡪 Number | | **to\_number(**'12,454.8-'**,** '99G999D9S'**);** -- = -12454.8 | | | | | | | | |
| **Transaktionen** | | | | | | | | | | |
| **BEGIN** **TRANSACTION** **ISOLATION** **LEVEL** **[level]** **;**  **INSERT** **INTO** A **VALUES** **(**123**,** 'test'**)** **;**  **SAVEPOINT** my\_safepoint **;**  **INSERT** **INTO** B **VALUES** **(**454**,** 'qwerz'**);**  **ROLLBACK** **TO** my\_safepoint **;**  **INSERT** **INTO** B **VALUES(**545**,** 'bob' **;**  **COMMIT;** | | | | | | | | | | |
| **Isolation Levels** | | | | | | | | | | |
| **READ UNCOMMITTED:** Liest auch Daten, welche UNcommited sind **READ COMMITTED:** Liest nur Daten, welche COMMITed sind **REPEATABLE READ:** Lesevorgänge gleich wiederholbar **SERIALIZABLE:** Nacheinander | | | | | | | | | | |
| **Serialisierbarkeit** | | | | | | | | | | |
|  | | | | | | | | | | |
| 1) Konfliktpaare bestimmen  - Mindestens eine Write Operation  - Beide Operation auf gleiches Objekt (auch innerhalb derselben Transaktion)  2) Serialisierbarkeitsgraph erstellen  3) Commit Reihenfolge erstellen | | | | | |  | | 2) Serialisierungsgraph | | |
|  | | | | | | | | | | |
| Serialisierbar wenn keine gegenseitigen Abhängigkeiten der Transaktionen (T1 ⬄ T2) | | | | | | | | | | |
| **Locking (X-Lock=Exclusive, S-Lock=Shared)** | | | | | | | | | | |
| **Two Phase Locking :** 1 Growing Phase, 2. Shrinking Phase (UNLOCK()) | | | | | | | | | | |
| **SLOCK:** Wird bei einem lesenden Zugriff abgesetzt. Ein Objekt kann von mehreren Transaktionen gleichzeitig gesperrt werden (allerdings kein XLOCK) | | | | | | | | | | |
| **XLOCK:** Wird bei einem schreibenden Zugriff abgesetzt. Ein Objekt kann nicht doppelt gesperrt werden. | | | | | | | | | | |
| **Deadlock:** Mehrere Transaktionen behindern sich gegenseitig beim Locking der jeweilig anderen Ressource. | | | | | | | | | | |
| **Optimistisches Locking:** Geht von wenigen schreibenden Zugriffen aus; Auf SLOCK wird verzichtet, aber es wird geprüft ob sich ein Zeitstempel geändert hat | | | | | | | | | | |
| **Pessimistisches Locking:** Geht von vielen schreibenden Zugriffen aus; SLOCK und XLOCKS | | | | | | | | | | |
| **Multiuser Probleme** | | | | | | | | | | |
|  | | | | | | | **Dirty Read:** Uncommitete Daten werden gelesen | | | |
| **Non Repeatable Read/Fuzzy Read:** Daten werden zwischendurch geändert, gelöscht | | | |
| **Phantom Read:** NeueDaten werden zwischendurch eingefügt (tritt in Postgres nicht auf) | | | |
|  | | | | | | | | | | |
| **Backup** | | | | | | | | | | |
| **Full Backup:** Daten und Log-Files / **Incremental Backup:** Log-File  **Export:** pg\_dump dbname > outfile, **Import:** psql dbname < infile  **Mirroring :** Server Replikation | | | | | | | | | | |
| **Write Ahead Log (WAL)** | | | | | | | | | | |
| 1) Änderungen der Transaktion in Log schreiben 2) Commit ins Log schreiben 3) Commit in die physiche Datenbank | | | | | | | | | | |

|  |  |  |
| --- | --- | --- |
| **JDBC** | | |
|  | | |
| C:\Users\Michi\Dropbox\share (1)\Db\screenshot_insertcoffees.png | | |
| **C:\Users\Michi\Dropbox\share (1)\Db\db_metadata.png** |  | |
|  | | |
|  | | 1)JDBC-ODBC Brücke  2)Native plattformeigene JDBC Treiber  3)Universeller JDBC Treiber (JAVA)  4)Direkte Netzwerktreiber (JAVA) |
| **Constraints (Column, Table)** | | |
| NOT NULL, UNIQUE, CHECK, PRIMARY KEY, FOREIGN KEY, CONSTRAINT mit Namen | | |
| **CREATE** **TABLE** product **{**  product\_no **INTEGER,**  order\_no **INTEGER,**  description **VARCHAR(**20**),**  discounted\_price **INTEGER,**  price **INTEGER** **NOT** **NULL,**  **CHECK(**price **>** discounted\_price**),**  **UNIQUE** **(**product\_no**,** description**),**  **PRIMARY** **KEY(**product\_no**),**  **FOREIGN** **KEY(**order\_no**)** **REFERENCES** **order(**id**),**  **CONSTRAINT** constraint\_name **{CHECK(**a **<** b**)** **|** **UNIQUE()},**  **}** | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Datenbankentwurf Prozess** | | | | |
| 1) Informations-, Datenverarbeitungsanforderungen 2) konzeptionelles Domänen Modell 3) logisches, relationales Modell 4) physisches DB Modell 5) Internes Schema oder spez DB. | | | | |
| **Logischen Domainmodell (UML)** | | | | |
| Datentypen angeben, Keine Schlüsselattribute, n-m mit Assoziationstabellen | | | | |
|  | |  | | |
| Aggregation und Komposition: Ist-Teil-von-Ganzem Beziehung wobei Komposition stärker bindet. (Wird der Parent gelöscht, muss auch der Child gelöscht werden. CASCADE) | | | | |
| **Relationale Schreibweise** | | | | |
| Person (id INTEGER PK, name String NOT NULL, email String NOT NULL UNIQUE); Hund(id INTEGER PK, name String NOT NULL, PersID REFERENCES Person);  Bauer(id INTEGER PK, hofNr INTEGER NOT NULL CHECK > 0, PersID REFERENCES Person); PrivatPerson(id INTEGER PK, ausweisNr INTEGER, PersID REFERENCES Person); | | | | |
| REFERENCES verweisst implizit auf den Primärschlüssel. Ansonsten REFERENCES Person(name); | | | | |
| **Vererbung / Abbildungsregeln** | | | | |
| **Overlapping:** Attribute können zu mehreren Tabellen gehören. | | | |  |
| **Disjoint:** Attribute können zu genau einer Tabelle gehören. | | | |
| **Gemeinsame Attribute in Parent** | **Alles im Parent und mit Typ** | | | **Alles in Kindtabelle und Primary Keys über selbe Sequence** |
|  |  | | |  |
| **Vorteil:** Strikte Trennung. Keine Redundanzen  **Nachteil:** Mehrere Joins nötig | **Vorteil:** Keine Joins, alles in einer Tabelle  **Nachteil:** Constraint muss Zwitter unterbinden | | | **Vorteil:** Guter Mix zwischen den anderen beiden Lösungen  **Nachteil:** Reduntante Felder |
| **B-Tree** | | | | |
| m: max. Anzahl Element pro Knotenpunkt muss >3 sein  N: Anzahl Knotenpunkte  - Knotenpunkt:  - Ein Knotenpunkt beinhaltet mindestens m/2 Elemente (ausser root >=1)  - Ein Knotenpunkt hat maximal m+1 Unterknoten (sonst rebalancing)  - Der Baum ist überall gleich hoch (sonst rebalancing)  max. Höhe: h=1+[log (m/2) (N/2)] | | | | |
| Einfügen:  1. Suchen nach Ordnungsplatz, einfügen  2. Falls Überlauf, teilen  2.1. Neuer Knoten mit Zahlen rechts von Mittlerer  2.2. Mittlerer Eintrag in Vaterknoten  2.3. Neuer Knoten mit Vaterknoten verknüpfen  3. Falls Vaterknoten überfüllt:  3.1. Falls Root, neue Root anlegen  3.2. Ansonsten wie bei 2. | | | C:\Users\Michi\Dropbox\share (1)\screenshot_btree.png  4 und 7 ergeben neuen Unterknoten | |