2012

Datové sklady

Cvičení 1

Emil Řezanina
12.3.2012

Obsah

Zadání

1) *Fyzický návrh datového skladu a Vysvětlení pojmů:*
   - Partitioned Tables,
   - Materialized Views,
   - Dimensions.
2) *Na základě schématu SH popište jednu:*
   - Partitioned table,
   - Materialized View,
   - Dimenzi.

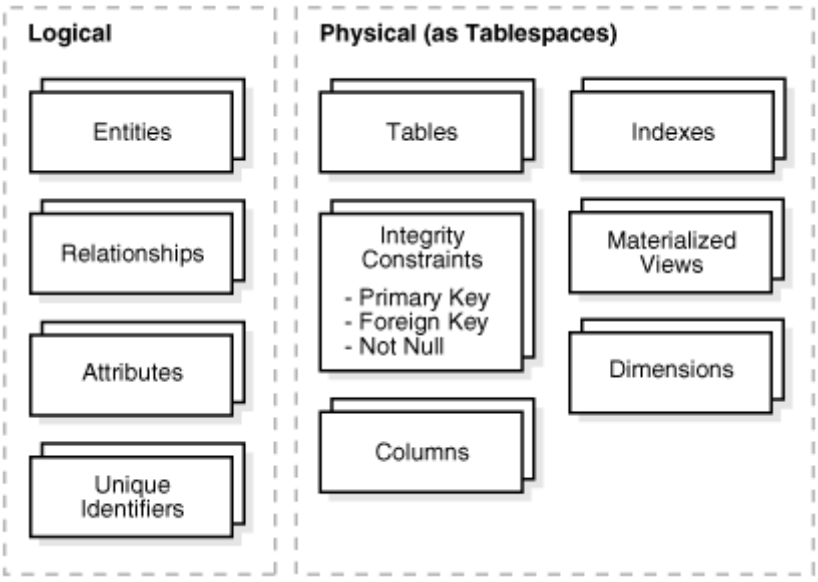
Fyzický návrh datového skladu a vysvětlení pojmů

Fyzický návrh datového skladu

U vytváření datového skladu je jednou z nejdůležitějších částí návrh, který se dělí na dvě části a to logický a fyzický.

Logický návrh je to, co nakreslíte tužkou na papír nebo návrh s Oracle Warehouse Boilderem nebo Oracle Designerem před budování vašeho datového skladu. Můžeme ho definovat jako pohled od uživatele. Cílem logického návrhu je model složený z entit, atributů a vztahů mezi nimi.

Fyzický návrh je vytváření databáze SQL příkazy. Který zase můžeme definovat jako pohled z databáze. Během procesu fyzického návrhu, konvertujete data získané během fáze logického návrhu do popisu fyzické databázové struktury. Je to tedy převod logického návrhu.



Obrázek 1: Porovnání logického a fyzického návrhu

Během procesu fyzického návrhu, převádíme očekávané schéma do aktuální struktury databáze. Fyzický návrh je rozhodující pro výkon databáze. V této chvíli je třeba namapovat:

- Entity na tabulky,
- Vztahy na omezení cizích klíčů,
- atributy na sloupce,
- Primární unikátní klíče na omezení primárních klíčů,
- Unikátní identifikátory na omezení unikátních klíčů.

Struktury fyzikálního návrhu

Jakmile jste převedli váš logický návrh na fyzický, musíte vytvořit některé nebo všechny následující struktury:

- tabulkový prostor,
- tabulky a rozdělené tabulky,
- pohledy,
- integritní omezení,

- dimenze.

Některé z těchto struktur požadují místo na disko. Jiné existují pouze v datovém slovníku. Navíc, následující struktury mohou být vytvořeny pro zlepšení výkonu:

- indexy a rozdělené indexy,
- materializované pohledy.

Popis struktur

Tabulkové prostory (tablespaces) jsou složeny z jednoho nebo více datových souborů (fyzický soubor na disku). Každý datový soubor patří jednomu tabulkovému prostoru. Z hlediska návrhu je tabulkový prostor kontejner pro fyzický návrh struktur. Tabulkové prostory je třeba rozdílně oddělit. Měly by být odděleny tabulky od svých indexů a malé tabulky od velkých. Pro velmi rozsáhlé databáze se dají použít datové soubory ultralarge.

Tabulky (tables) jsou základní jednotkou pro uložení dat, slouží jako kontejnery pro očekávané množství nezpracovaných dat v datovém skladu. **Rozdělené tabulky** (partitioned tables) se používají pro velké objemy dat, které se rozdělí na menší a lépe ovladatelné části. Důležitým kritériem pro rozdělení je tedy ovladatelnost a zlepšení výkonu. Rozdělování dat se typicky provádí pomocí transakčních dat. Pro vysoce redundantní data se můžou data v tabulkách zkomprimovat, neměly by se ale komprimovat hodně aktualizované tabulky nebo tabulky s mnoho DML operacemi z důvodů vysoké režie.

Pohledy (views) přizpůsobují prezentaci dat z jedné nebo více tabulek nebo z jiných pohledů. S pohledy se zachází stejně jako s tabulkami ale neukládají žádná data.

Integritní omezení (integrity constraints) jsou pravidla, která zabraňují ukládání neplatných dat do tabulek. Integritní omezení v datových skladech se liší od integritních omezení aplikovaných v prostředí OLTP.

V tomto prostředí se integritní omezení starají o to, aby se do databáze neukládala neplatná data, jenže tento problém není v datových skladech potřeba řešit, protože platná data jsou zaručena. V datových skladech se omezení používá pouze pro dotazy na přepsání dat. Běžné omezení v datových skladech je NOT NULL. Za jistých okolností potřebují integritní omezení i místo v databázi. Tyto omezení jsou ve formě unikátních indexů.

Indexy (indexes) jsou volitelné struktury, které souvisí s tabulkami nebo clustery. Kromě klasických indexů s B-stromem se v datových skladech často používají i bitmapové indexy. Bitmapové indexy jsou optimalizované indexové struktury orientované na vkládací operace. Dále jsou nutné pro optimalizované metody, které slouží pro přístup k datům. Indexy lze stejně jako tabulky vytvořit jako rozdělené. **Rozdělené indexy** (partitioned indexes) usnadňují správu datového skladu během obnovování a zlepšují výkon dotazů.

Materializované pohledy (materialized views) jsou výsledky dotazů, které byly uloženy v předstihu pro dlouho běžící výpočty. Z fyzického hlediska se materializované pohledy podobají tabulkám nebo rozděleným tabulkám, chovají se stejně jako indexy v tom, že jsou transparentní a zlepšují výkonnost.

Dimenze (dimension) je objekt schématu, který definuje hierarchické vztahy mezi sloupci a sadou sloupců. Hierarchický vztah je funkční závislost z jedné úrovně hierarchie k další úrovni. Dimenze je kontejner pro logické vztahy a nevyžaduje žádné místo v databázi.

Partitioned Tables

Datové sklady často obsahují velké tabulky a požadují techniky jak pro správu těchto velkých tabulek ale i pro zajištění vysokého výkonu dotazu přes tyto velké tabulky. Důležitým nástrojem pro dosažení tohoto cíle, stejně jako zlepšení přístupu k datům a zlepšení celkového výkonu aplikace je rozdělení disku.

Rozdělení disku nabízí podporu velkým tabulkám a indexům, pomocí něhož můžete je rozložit do menších a lépe zvládnutelných částí nazývaných oddíly. Tato podpora je zvláště důležitá pro aplikace, které přistupují k tabulkám a indexům o milionech řádků a několik gigabytů dat.

Každý oddíl tabulky nebo indexu musí mít stejné logické atributy, jako například jména sloupců, datové typy a omezení, ale každý oddíl může mít rozdělené fyzické atributy, jako například komprese povolena nebo zakázána, nastavení fyzického uložiště a tabulkový prostor.

Rozdělení disku nabízí tyto výhody:

- Umožňuje operace pro správu dat, jako jsou načtení dat, vytvoření indexů a jejich přestavba, a zálohování a obnovení na úrovni oddílů, spíše než na celou tabulku. To má za výsledek výrazné snížení doby pro tyto operace.
- Zlepšuje výkon dotazů. Často výsledky dotazů mohou být dosaženy přistupováním k podmnožině oddílů, spíše než nad celou tabulkou.
- Významně snižuje dopad plánované odstávky pro provádění údržby.
- Zvyšuje dostupnost milion-critical databází, když kritické tabulky a indexy jsou rozděleny do oddílů ke snížení údržby oken, časy obnovení a dopad poruch.
- Paralelní vykonávání poskytuje speciální výhody k optimalizaci zdrojů využití a minimalizuje vykonávaný čas. Paralelní vykonávání proti rozděleným objektům je klíč pro škálovatelnost v prostředí clusterů. Paralelní vykonávání je podporováno pro dotazy a pro DML a DDL.

Výhody rozdělení disku není pouze pro velmi velké databáze. Každá databáze, i malé databáze, mohou prosperovat z rozdělení disku. Zatímco rozdělování je nutné pro velké databáze, rozdělení disku je samozřejmě výhodné i pro menší databáze.

Partitioning Key

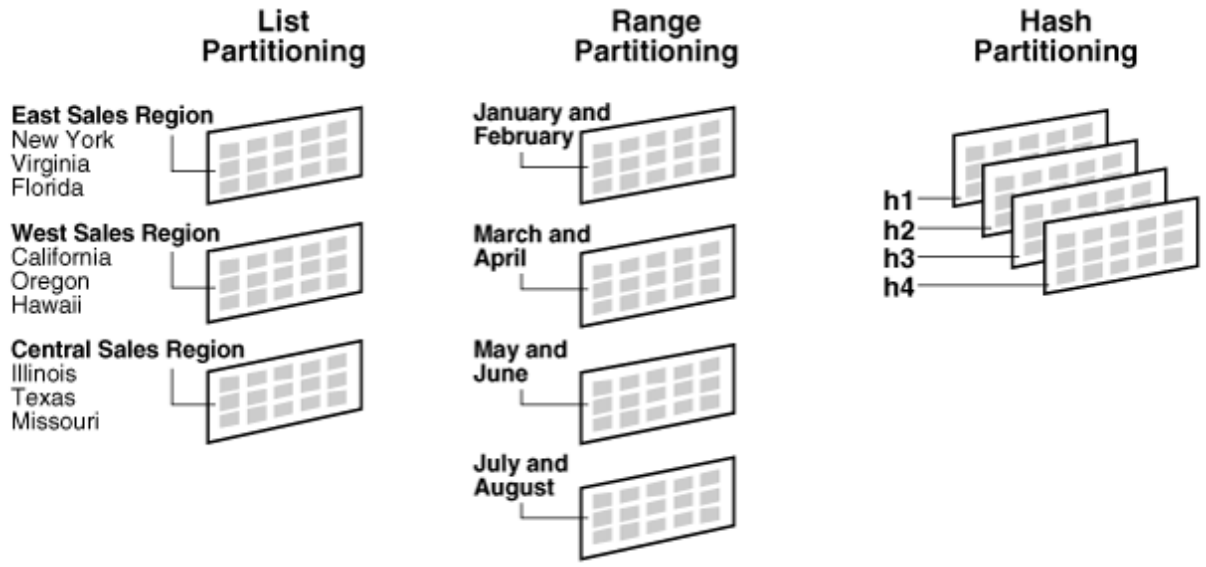
Každý řádek v partitioned tabulce je jednoznačně přiřazen k jednomu oddílu. Partitioning Key se skládá z jednoho nebo více sloupců, které určují oddíl, kde je každý řádek uložen. Oracle automaticky směřuje insert, update a delete operace k příslušnému oddílu s partitioning key.

Rozdělení disku strategie

Oracle rozdělení disku poskytuje tři základní metody datové distribuce jako základní rozdělení disku strategie, které kontrolují, jak data jsou umístěna v jednotlivých oddílech:

Range Partitioning	Rozdělení podle rozsahu ukládaných hodnot
List Partitioning	Rozdělení podle konkrétních hodnot některého atributu
Hash Partitioning	Rozdělení podle hashovacích tabulek
Composite Partitioning	Složené

Každá metoda má jiné výhody a hlediska návrhu a je vhodnější pro konkrétní situaci.



Obrázek 2: Strategie rozdělení disku na oddíly

Range Partitioning

Range Partitioning zmapuje data na oddíly rozdělené na základě rozdělovacího klíče. Je to nejběžnější metoda rozdělení, která se často používá s údaji o datu. Například můžeme rozdělit data o prodeji na měsíční oddíly (viz obrázek výše).

Range Partitioning zmapuje řádky na oddíly na základě hodnot ve sloupcích. Range Partitioning je definováno pro tabulku nebo index v PARTITION BY RANGE(column_list) a pro jednotlivé oddíly v VALUES LESS THAN(value_list), kde column_list je seznam sloupců, který určuje, do kterého oddílu patří řádky. Tyto sloupce se nazývají rozdělené sloupce. Seřazený seznam hodnot pro sloupce v seznamu sloupců se nazývá value_list. Každá hodnota musí být buď literál nebo funkce TO_DATE nebo RPAD s konstantními argumenty. Povolena je pouze klauzule VALUES LESS THAN, která specifikuje horní mez oddílu. Vytvoření tabulky s Range Partititoning:

```
CREATE TABLE sales_range ( salesman_id NUMBER(5), salesman_name VARCHAR2(30),
sales_amount NUMBER(10), sales_date DATE )
COMPRESS PARTITION BY RANGE(sales_date)
( PARTITION sales_jan2000 VALUES LESS THAN(TO_DATE('02/01/2000','DD/MM/YYYY')),
PARTITION sales_feb2000 VALUES LESS THAN(TO_DATE('03/01/2000','DD/MM/YYYY')),
PARTITION sales_mar2000 VALUES LESS THAN(TO_DATE('04/01/2000','DD/MM/YYYY')) );
```

List Partitioning

List Partitioning umožňuje explicitní kontrolu, jak se mají řádky mapovat do oddílů. Zadají se diskrétní body do seznamu pro rozdělovaný sloupec pro každý oddíl. Výhodou tohoto dělení je, že se mohou

organizovat skupiny neuspořádaných a nezávislých dat normálním způsobem. Vytvoření tabulky s List Partitioning:

```
CREATE TABLE sales_list ( salesman_id NUMBER(5), salesman_name VARCHAR2(30),
sales_state VARCHAR2(20), sales_amount NUMBER(10), sales_date DATE )
PARTITION BY LIST(sales_state)
( PARTITION sales_west VALUES('California', 'Hawaii') COMPRESS,
PARTITION sales_east VALUES('New York', 'Virginia', 'Florida'),
PARTITION sales_central VALUES('Texas', 'Illinois') );
```

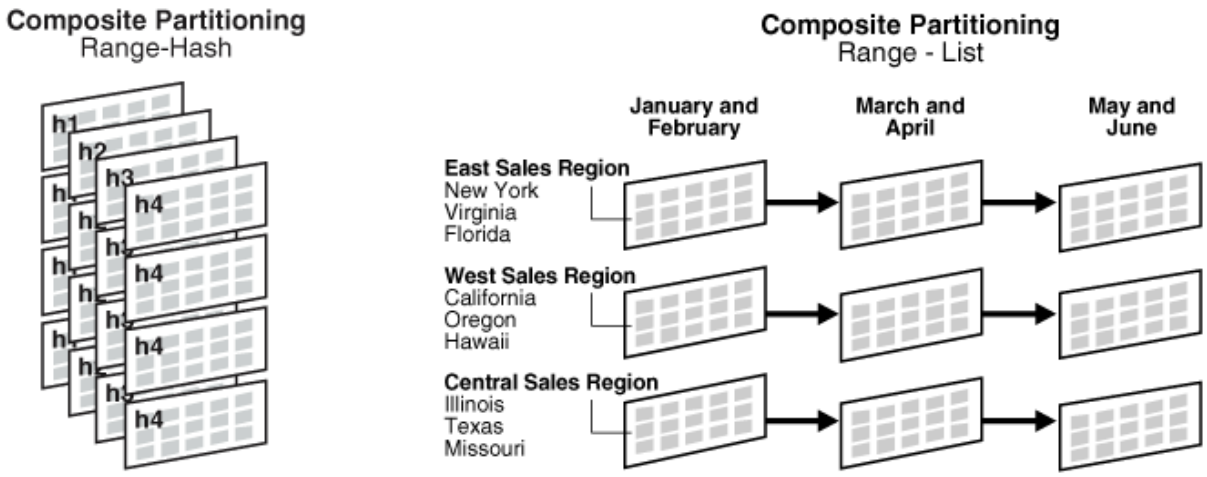
Hash Partitioning

Hash Partitioning zmapuje data na oddíly pomocí hashovacího algoritmu. Algoritmus Oracle aplikuje na vybrané rozdělovací klíče. Hashovací algoritmus rovnoměrně rozděluje řádky mezi oddíly, což vytváří oddíly přibližně stejné velikosti. Hash Partitioning je snadno použitelná alternativní metoda k Range Partitioning, zvláště když dělená data nejsou historického rázu. Oracle používá lineární hashovaní algoritmus. Vytvoření tabulky s Hash Partitioning:

```
CREATE TABLE sales_hash ( salesman_id NUMBER(5), salesman_name VARCHAR2(30),
sales_amount NUMBER(10), week_no NUMBER(2) )
PARTITION BY HASH(salesman_id)
PARTITIONS 4;
```

Composite Partitioning

Composite Partitioning kombinuje Range a Hash nebo List Partitioning. Databáze Oracle nejdříve rozdělí data do oddílů intervalu pomocí Range Partitioning. Pak pro Range-Hash Partitioning, Oracle rozdělí data pomocí hashovacího algoritmu na subpartitions pro každý oddíl získaný z Range Partitioning. Pro Range-List Partitioning, Oracle rozdělí data na supartitions pomocí Range Partitioning pro každý vybran seznam.



Obrázek 3: Composite Partitioning

Materialized Views

Jedna technika používaná v datových skladech pro zvýšení výkonu je vytvoření přehledů. Přehledy jsou speciálními typy agregovaných pohledů, které zlepšují časy spouštěných dotazů podle před výpočtových drahých spojení a agregační operace před výkonem a ukládaní výsledků do tabulky v databázi. Například můžete vytvořit souhrnnou tabulku obsahující součty prodejů podle krajů a podle produktů.

Přehledy nebo agregace, které jsou na datových skladech vytvářeny v Oracle databázi pomocí schématu objektu, se nazývají materializovaný pohled. Materializované pohledy mohou provádět celou řadu rolí, jako je například zlepšení výkonu dotazu nebo poskytování replikovaných dat.

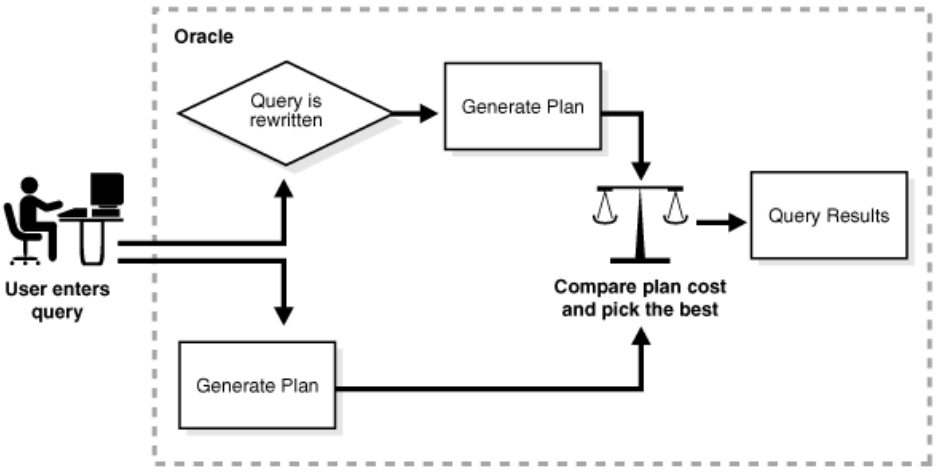
Materializované pohledy jsou pohledy, které poskytují pohled na určitá souhrnná data. Takových pohledů je vytvořeno více a jsou využívány uživateli, kteří dostávají požadovaná data. Mechanismus Oracle serveru upravuje SQL dotaz pro použití více souhrnů pro tvorbu ekvivalentních materializovaných pohledů. Materializované pohledy jsou dostupné pomocí dotazů od uživatele a v rámci datového skladu jsou transparentní pro koncového uživatele nebo databázové aplikace.

V datových skladech můžete používat materializované pohledy k předvýpočtům a ukládáním agregovaných dat jako například součet tržeb. Materializované pohledy v těchto prostředích jsou často označovány jako souhrny, protože ukládají souhrnné údaje. Mohou být také použity k předvypočítání spojení s nebo bez agregace. Materializované pohled eliminuje režii spojenou s nákladnými spojeními a agregací pro velké nebo důležité třídy dotazů. Aby data byla aktuální, je potřeba také nastavit určitý proces, kterým se zajistí jejich obnova.

Materializované pohledy našli také své využití v distribuovaných výpočtech a při zisku dat mobilními klienty.

Potřeby pro materializované pohledy

V datových skladech se materializované pohledy využívají ke zrychlení dotazů nad velkou databází, kde dochází často ke spouštění dotazů obsahujících agregační funkce. Operace agregace jsou náročné z hlediska času a operačního výkonu. Oracle optimalizátor automaticky rozpozná, kdy je vhodné využít některý z materializovaných pohledů pro odpověď na dotaz zadaný uživatelem. Tuto skutečnost popisuje následující schéma.



Obrázek 4: Transparent Query Rewrite

Motivací k používání materializovaných pohledů je zvýšení výkonu. Přesto se může stát, že materializované pohledy výkon serveru nezvyšují, protože mohou být špatně nastaveny nebo nejsou využívány v pravém okamžiku.

Při použití dotazu přepsání, vytvoří se materializovaný pohledy, které splňují největší počet dotazů. Například pokud identifikujete 20 dotazů, které jsou běžně aplikovány na detaily nebo skutečnosti tabulek, pak byste měli být schopni uspokojit je s pěti nebo šesti dobře napsanými materializovanými pohledy. Definice materializovaného pohledu může obsahovat libovolný počet agregací (SUM, COUNT(x), COUNT(*), COUNT(DISTINCT x), AVG, VARIANCE, STDDEV, MIN a MAX). To může také obsahovat libovolný počet spojení.

Přehled úkolů řízení materializovaného pohledu

Motivace pro použití materializovaných pohledů je zlepšení výkonu, ale režie spojené s řízením materializovaného pohledu se může stát významným problémem systémového řízení. Při kontrole nebo vyhodnocování některých nezbytných činností řízení materializovaného pohledu, zvažte některé z následujících akcí:

- Identifikování toho, co materializovaný pohledy potřebují k vytvoření zpočátku.
- Indexování materializovaných pohledů.
- Zajistit, aby všechny materializované pohledy a indexy materializovaných pohledů byli aktualizovány správně pokaždé, když je databáze aktualizovaná.
- Kontrola, které materializované pohledy byly použity.
- Určování toho, jak efektivní každý materializovaný pohled byl na pracovní zátěži výkonu.
- Určení, které nové materializované pohledy by měly být vytvořeny.
- Určení, které existující materializované pohledy by měli být zrušeny.
- Archivace staršího detailu a materializovaného pohledu dat, která již nejsou užitečná.
- Hromadné načtení dat do skladu.

Hlavní typy materializovaných pohledů

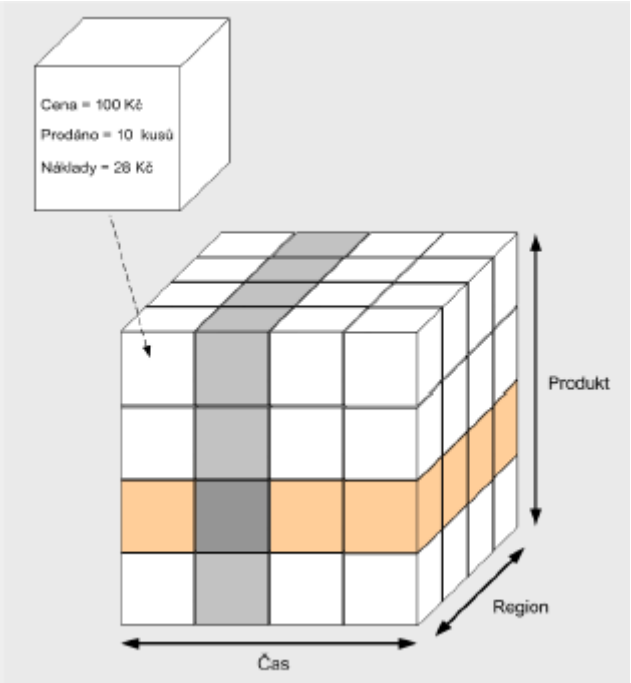
- materializované pohledy s agregací,
- materializované pohledy obsahující pouze spojení,
- vnořené materializované pohledy.

Příklad vytvoření materializovaného pohledu

```
CREATE MATERIALIZED VIEW product_sales_mv
PCTFREE 0 TABLESPACE demo
STORAGE (INITIAL 8k NEXT 8k PCTINCREASE 0)
BUILD IMMEDIATE
REFRESH FAST
ENABLE QUERY REWRITE
AS SELECT p.prod_name, SUM(s.amount_sold) AS dollar_sales, COUNT(*) AS cnt, COUNT(s.amount_sold) AS cnt_amt FROM
sales s, products p WHERE s.prod_id = p.prod_id GROUP BY p.prod_name;
```

Dimensions

Dimenze je struktura, která kategorizuje data s cílem umožnit uživatelům odpovědět na byznys otázky. Běžně používané dimenze jsou zákazníci, produkty a čas.



Obrázek 5: Multidimenzionální datová kostka

Data v obchodníkově datovém skladu má dvě důležité složky: dimenze a fakta. Dimenze jsou produkty, zákazníci, propagace, kanály a čas. Jeden přístup pro identifikaci vašich dimenzí je zkontrolovat vaši referenční tabulky, jako je například produkt tabulka, která obsahuje všechno kolem produktu, nebo propagační tabulka obsahující veškeré informace o promo akcích. Fakta jsou tržby (prodaných kusů) a zisky. Datový sklad obsahuje údaje o prodeji jednotlivých produktů na denní bázi.

Dimenze pomáhá organizovat a grupovat informace do hierarchie. Reprezentuje přirozený vztah 1:N mezi sloupci nebo skupinou sloupců (úrovních hierarchie), které nemohou být reprezentovány omezujícími podmínkami. V hierarchii se jít o úroveň výš nazývá rolling up a jít o úroveň dolů drilling down. Příklad:

- V časové dimenzi, měsíce „rolují“ na čtvrtletí, čtvrtletí rolují na roky. Analýza dat obvykle začíná na vyšší úrovni v dimenzionální hierarchii a postupně se „vrtá“ dolů, pokud situace vyžaduje.
- V rámci produktu dimenzi, výrobky „rolují“ do podkategorií, podkategorie „rolují“ do kategorií a kategorie „rolují“ do skupiny všech výrobků.
- V rámci dimenze zákazníků, zákazníci „rolují“ do města. Potom města „rolují“ do krajů. Pak kraje „rolují“ do země atd. (viz obrázek vpravo)

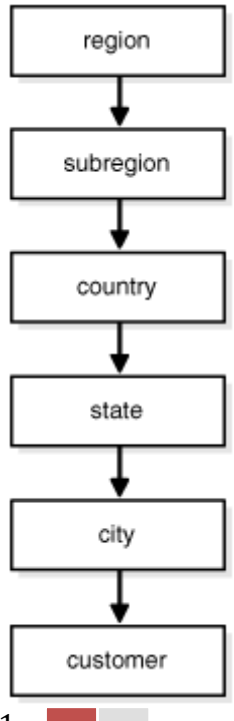
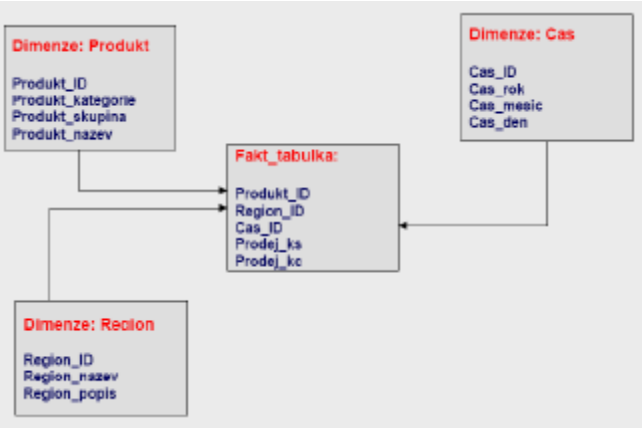


Schéma typu hvězda

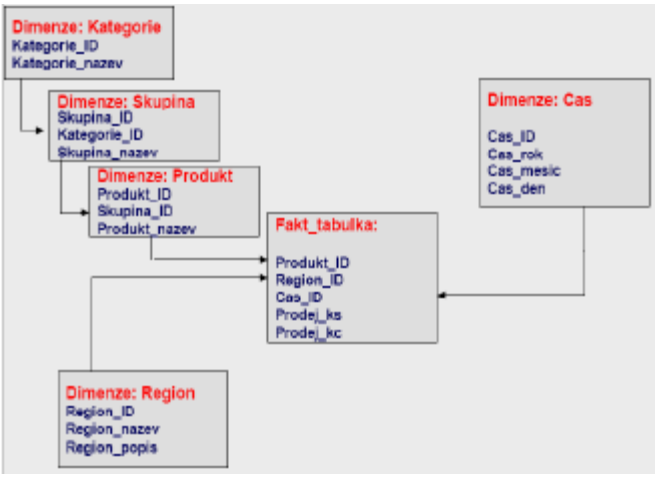
Data nejsou v 3NF. Zpracování dotazů je v tomto případě rychlejší a proto se toto schéma používá více (viz obrázek 6).



Obrázek 6: Schéma typu hvězda

Schéma typu sněhová vločka

Tabulky jsou v 3NF, ale schéma je méně používané jak schéma typu hvězda (viz Obrázek 7).



Obrázek 7: Schéma typu sněhová vločka

Příklad vytvoření dimenze

```
CREATE DIMENSION products_dim
LEVEL product IS (products.prod_id)
LEVEL subcategory IS (products.prod_subcategory)
LEVEL category IS (products.prod_category)
HIERARCHY prod_rollup (
product CHILD OF
subcategory CHILD OF category)
ATTRIBUTE product DETERMINES
(products.prod_name, products.prod_desc, prod_weight_class, prod_unit_of_measure,
prod_pack_size,prod_status, prod_list_price, prod_min_price)
ATTRIBUTE subcategory_att DETERMINES
(prod_subcategory, prod_subcategory_desc)
ATTRIBUTE category DETERMINES
(prod_category, prod_category_desc);
```

Popsané objekty schématu SH

Partitioned table – tabulka SALES

Struktura

Atribut	Datový typ	Koment
PROD_ID	NUMBER	FK to the products dimension table
CUST_ID	NUMBER	FK to the customers dimension table
TIME_ID	DATE	FK to the times dimension table
CHANNEL_ID	NUMBER	FK to the channels dimension table
PROMO_ID	NUMBER	Promotion identifier, without FK constraint (intentionally) to show outer join optimalization
QUANTITY_SOLD	NUMBER(10,2)	Product quantity sold with the transaction
AMOUNT_SOLD	NUMBER(10,2)	Invoiced amount to the customer

Omezení

Název	Typ	Sloupec
SALES_CHANNEL_FK	FK	CHANNEL_ID
SALES_CUSTOMER_FK	FK	CUST_ID
SALES_PRODUCT_FK	FK	PROD_ID
SALES_PROMO_FK	FK	PROMO_ID
SALES_TIME_FK	FK	TIME_ID
SYS_C005209	Check NOT NULL	PROD_ID
SYS_C005210	Check NOT NULL	CUST_ID
SYS_C005211	Check NOT NULL	TIME_ID
SYS_C005212	Check NOT NULL	CHANNEL_ID
SYS_C005213	Check NOT NULL	PROMO_ID
SYS_C005214	Check NOT NULL	QUANTITY_SOLD
SYS_C05215	Check NOT NULL	AMOUNT_SOLD

Indexy

Název	Typ	PARTITIONED	Sloupec
SALES_CUST_BIX	BITMAP	Ano	CUST_ID
SALES_PROD_BIX	BITMAP	Ano	PROD_ID
SALES_TIME_BIX	BITMAP	Ano	TIME_ID
SALES_PROMO_BIX	BITMAP	Ano	PROMO_ID
SALES_CHANNEL_BIX	BITMAP	Ano	CHANNEL_ID

SQL create příkaz

```
CREATE TABLE "SH"."SALES"
  (
    "PROD_ID" NUMBER NOT NULL ENABLE,
    "CUST_ID" NUMBER NOT NULL ENABLE,
    "TIME_ID" DATE NOT NULL ENABLE,
    "CHANNEL_ID"    NUMBER NOT NULL ENABLE,
    "PROMO_ID"      NUMBER NOT NULL ENABLE,
    "QUANTITY_SOLD" NUMBER(10,2) NOT NULL ENABLE,
    "AMOUNT_SOLD"   NUMBER(10,2) NOT NULL ENABLE,
    CONSTRAINT "SALES_CHANNEL_FK" FOREIGN KEY ("CHANNEL_ID") REFERENCES "SH"."CHANNELS"
("CHANNEL_ID") ENABLE NOVALIDATE,
```

```
    CONSTRAINT "SALES_TIME_FK" FOREIGN KEY ("TIME_ID") REFERENCES "SH"."TIMES" ("TIME_ID")
ENABLE NOVALIDATE,
    CONSTRAINT "SALES PRODUCT FK" FOREIGN KEY ("PROD ID") REFERENCES "SH"."PRODUCTS"
("PROD ID") ENABLE NOVALIDATE,
    CONSTRAINT "SALES_CUSTOMER_FK" FOREIGN KEY ("CUST_ID") REFERENCES "SH"."CUSTOMERS"
("CUST_ID") ENABLE NOVALIDATE,
    CONSTRAINT "SALES_PROMO_FK" FOREIGN KEY ("PROMO_ID") REFERENCES "SH"."PROMOTIONS"
("PROMO ID") ENABLE NOVALIDATE
  )
  PCTFREE 5 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS NOLOGGING STORAGE
  (
    BUFFER_POOL DEFAULT
  )
  TABLESPACE "EXAMPLE" PARTITION BY RANGE
  ("TIME_ID")
```

Popis

Metoda partitioning:	Range
Partitioning sloupce:	TIME_ID
Počet Partitions:	28
Počet záznamů:	918843
Počet bloků:	1769
Poslední analýza:	13.12.2009

Materialized View – CAL_MONTH_SALES_MV

Struktura

Atribut	Datový typ
CALENDAR_MONTH_DESC	VARCHAR2(8 BYTE)
DOLLARS	NUMBER

SQL příkaz

```
SELECT   t.calendar_month_desc
  ,        sum(s.amount_sold) AS dollars
  FROM     sales s
  ,        times t
  WHERE    s.time_id = t.time_id
  GROUP BY t.calendar_month_desc
```

Nastavení

Rewritable:	True
Updatable:	True
Parallel:	No Parallel
Caching:	No

Obnovení dat

Refresh Type:	Force
Refresh Method:	Primary Key
Refresh Interval:	On Demand
Last Refresh Date:	13.12.09

Dimenze – CUSTOMERS_DIM

Úrovně

CITY, COUNTRY, CUSTOMER, CUST_TOTAL, GEOG_TOTAL, REGION, STATE, SUBREGION

Hierarchie

GEOG_ROLLUP ->GEOG_TOTAL, REGION, SUBREGION, COUNTRY, STATE, CITY, CUSTOMER

CUST_ROLLUP -> CUST_TOTAL, STATE, CITY, CUSTOMER

Řazení

Primárně:	Hierarchicky
Sekundárně:	Vzestupně

SQL příkaz pro vytvoření

```
CREATE DIMENSION SH.CUSTOMERS_DIM
LEVEL CITY IS CUSTOMERS.CUST_CITY_ID
LEVEL COUNTRY IS COUNTRIES.COUNTRY_ID
LEVEL CUSTOMER IS CUSTOMERS.CUST_ID
LEVEL CUST_TOTAL IS CUSTOMERS.CUST_TOTAL_ID
LEVEL GEOG_TOTAL IS COUNTRIES.COUNTRY_TOTAL_ID
LEVEL REGION IS COUNTRIES.COUNTRY_REGION_ID
LEVEL STATE IS CUSTOMERS.CUST_STATE_PROVINCE_ID
LEVEL SUBREGION IS COUNTRIES.COUNTRY_SUBREGION_ID
HIERARCHY CUST_ROLLUP (CUSTOMER CHILD OF CITY CHILD OF STATE CHILD
OF CUST_TOTAL)
HIERARCHY GEOG_ROLLUP (CUSTOMER CHILD OF CITY CHILD OF STATE CHILD
OF COUNTRY CHILD OF SUBREGION CHILD OF REGION CHILD OF GEOG_TOTAL
JOIN KEY CUSTOMERS.COUNTRY_ID REFERENCES COUNTRY)
ATTRIBUTE CITY DETERMINES CUSTOMERS.CUST_CITY
ATTRIBUTE COUNTRY DETERMINES COUNTRIES.COUNTRY_NAME
ATTRIBUTE CUSTOMER DETERMINES (CUSTOMERS.CUST_CREDIT_LIMIT,
CUSTOMERS.CUST_EMAIL, CUSTOMERS.CUST_FIRST_NAME,
CUSTOMERS.CUST_GENDER, CUSTOMERS.CUST_INCOME_LEVEL,
CUSTOMERS.CUST_LAST_NAME, CUSTOMERS.CUST_MARITAL_STATUS,
CUSTOMERS.CUST_MAIN_PHONE_NUMBER, CUSTOMERS.CUST_POSTAL_CODE,
CUSTOMERS.CUST_STREET_ADDRESS, CUSTOMERS.CUST_YEAR_OF_BIRTH)
ATTRIBUTE REGION DETERMINES COUNTRIES.COUNTRY_REGION
ATTRIBUTE STATE DETERMINES CUSTOMERS.CUST_STATE_PROVINCE
ATTRIBUTE SUBREGION DETERMINES COUNTRIES.COUNTRY_SUBREGION
```


Zdroje:

http://docs.oracle.com/cd/E11882_01/server.112/e25523/toc.htm
http://docs.oracle.com/cd/E11882_01/server.112/e25554/parpart.htm