HARP Raw Event Database

A Case Study in Oracle Development @ CERN

Andrea Valassi

(CERN IT-ADC)

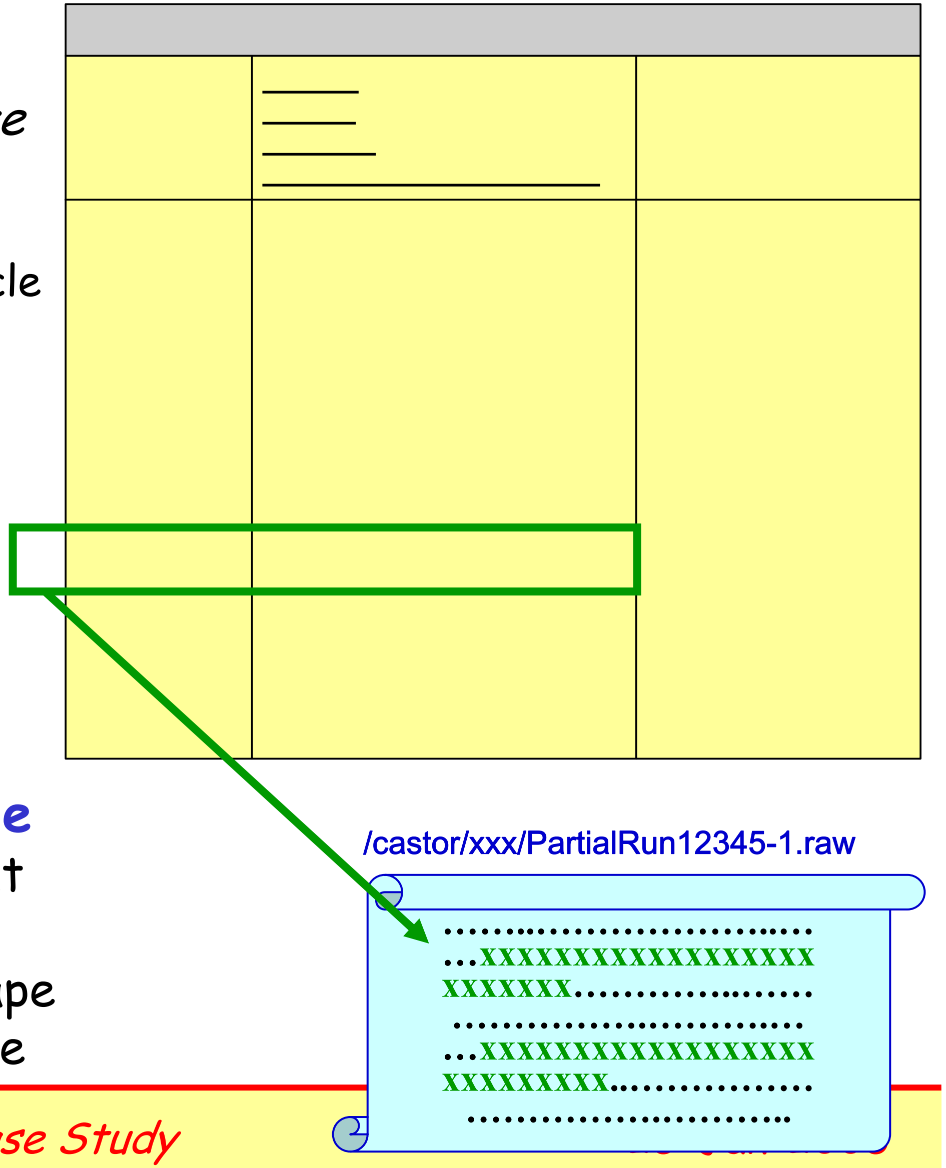
Introducing the context: Objectivity to Oracle migration

- **End of Objectivity support at CERN in 2003**
  - HARP migration project started in 2002, completed in January 2004
  - Significant reuse of COMPASS migration tools and software
- **Migration of both HARP data and software**
  - Raw events and conditions data: will only cover event data in this talk
  - *200 GB metadata in Oracle* and *25 TB raw BLOBs in Castor flat files*
  - Oracle read-only software for event analysis in HARP Gaudi framework
- **For more information: CHEP 2004 poster and paper**

  http://indico.cern.ch/contributionDisplay.py?contribId=448&sessionId=24&confId=0

Raw data (BLOBs on tape files) vs. metadata (Oracle)

- **Main Oracle table: the event table**
  - 800M events = 800M rows
  - Event table alone: *>95% Oracle disk space*
    - Total: 100 GB data and 110 GB indexes
  - Event table is main COMPASS table too
    - COMPASS factor 10 larger: >3 TB in Oracle
    - HARP also has more complex metadata
- **Each event row has the metadata for the raw event record (BLOB) on tape**
  - Record file offset and record size
  - Event records alone: *>95% data on tape*
    - Total: 25 TB data in 30k files
    - COMPASS: >300 TB data in Castor
- **BLOBs on flat files vs BLOBS in Oracle**
  - No need to query events by BLOB content
  - No need for 25 TB of Oracle space
  - No need for ~AMS-like retrieval from tape
  - Easier to manage, no obvious disadvantage



Oracle development in practice: a few points from the HARP case

- **Logical schema design for the HARP event data model**
  - Run/…/Event table hierarchy with composite primary keys (indexes)
  - Collection/Setting/Run table hierarchy with redundant integrity constraints
- **Users and roles**
  - Separate owner/writer accounts (one per data set) from the reader account
- **Physical schema design**
  - Separate data tablespaces and index tablespaces
  - Run range partitioning for the Run/../Event tables with local indexes
  - Use one tablespace for each partition (of table or index)
- **Simplify and optimize read access to summary information**
  - Encapsulate complex joins, subqueries, group by rollup into views
  - Speed up read access via materialized views with global query rewrite
- **C++ access (using the OCCI library)**
  - Write access: bulk insertion with bind variables, one statement per table
  - Read access: bulk retrieval with prefetching/caching
  - Execute queries (basic selection) on the database server, not in the C++ client
- **Data management practices and useful tools**
  - Encapsulate schema creation in SQL scripts executed through sqlplus
  - Interactive tests of DDL/DML/select using Benthic and TOra

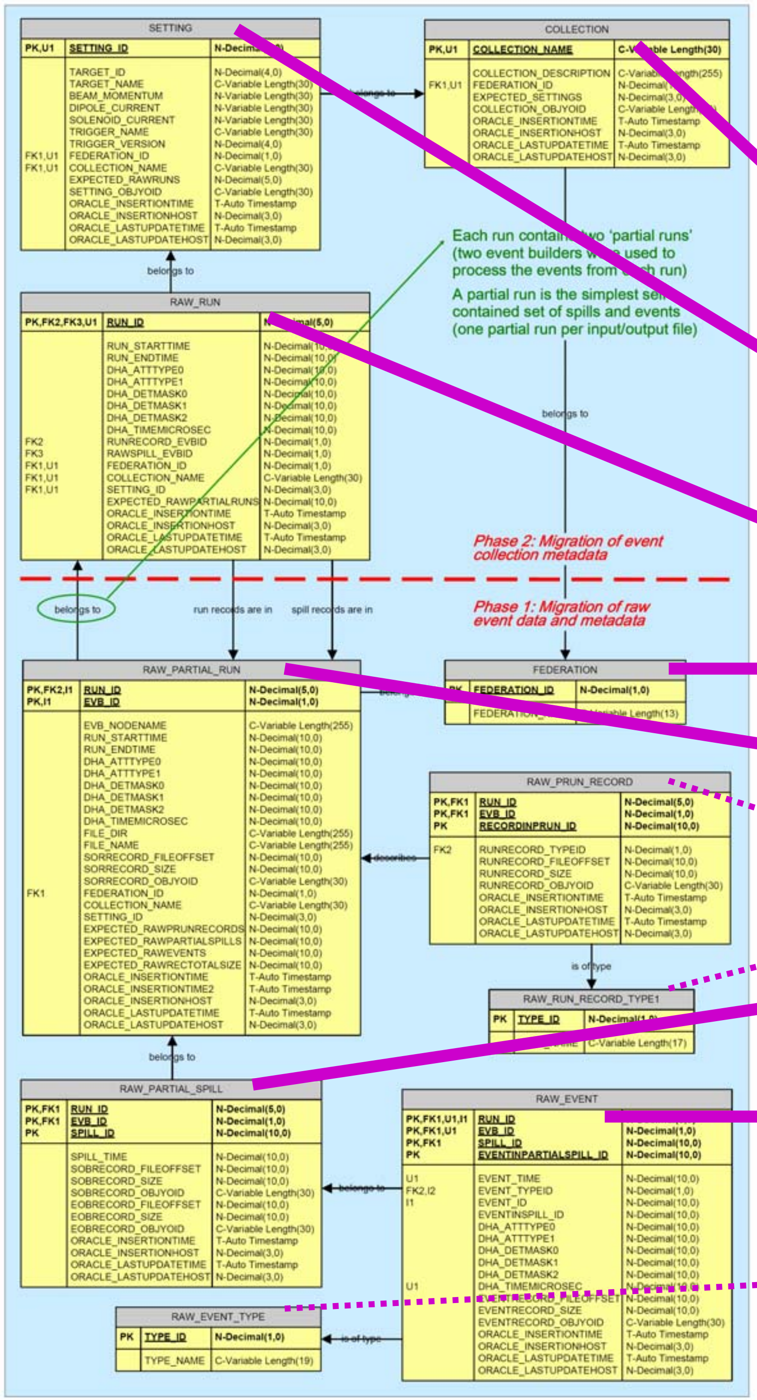
Logical schema design: the HARP hierarchical event data model (1)

- **Runs and below – data and metadata from the DAQ**
  - Two 'partial runs' in each run – two separate event builders in the DAQ
  - Many (partial) 'spills' in each (partial) run – bursts from the PS beam
  - Many events in each spill

- **Runs and above – higher level (bookkeeping) metadata**
  - Runs are grouped in "settings" (beam momentum, target type, etc)
  - Settings are grouped in "collections" (physics, cosmics, etc.)
  - Two Objectivity "federations": 2001 and 2002 data sets

- **Migration proceeded in two phases**
  - 1. Bulk data migration, one 'partial run' (file) at a time
    - One partial run = one Objectivity file
    - *Start with a self-contained Oracle subschema for partial runs, partial spills, events*
  - 2. Higher level metadata migration
    - Lower data volumes, more complexity, extensive checks of internal consistency
    - *Complete full Oracle schema with setting and collection metadata*
    - *Add all final summary tables for end-user analysis*

Logical schema design: the HARP hierarchical event data model (2)



Collections contain settings

Settings contain runs

Runs contain partial runs

Partial runs contain partial spills

Run record

Run record type (SOR, EOR, …)

Partial spills contain events

Events

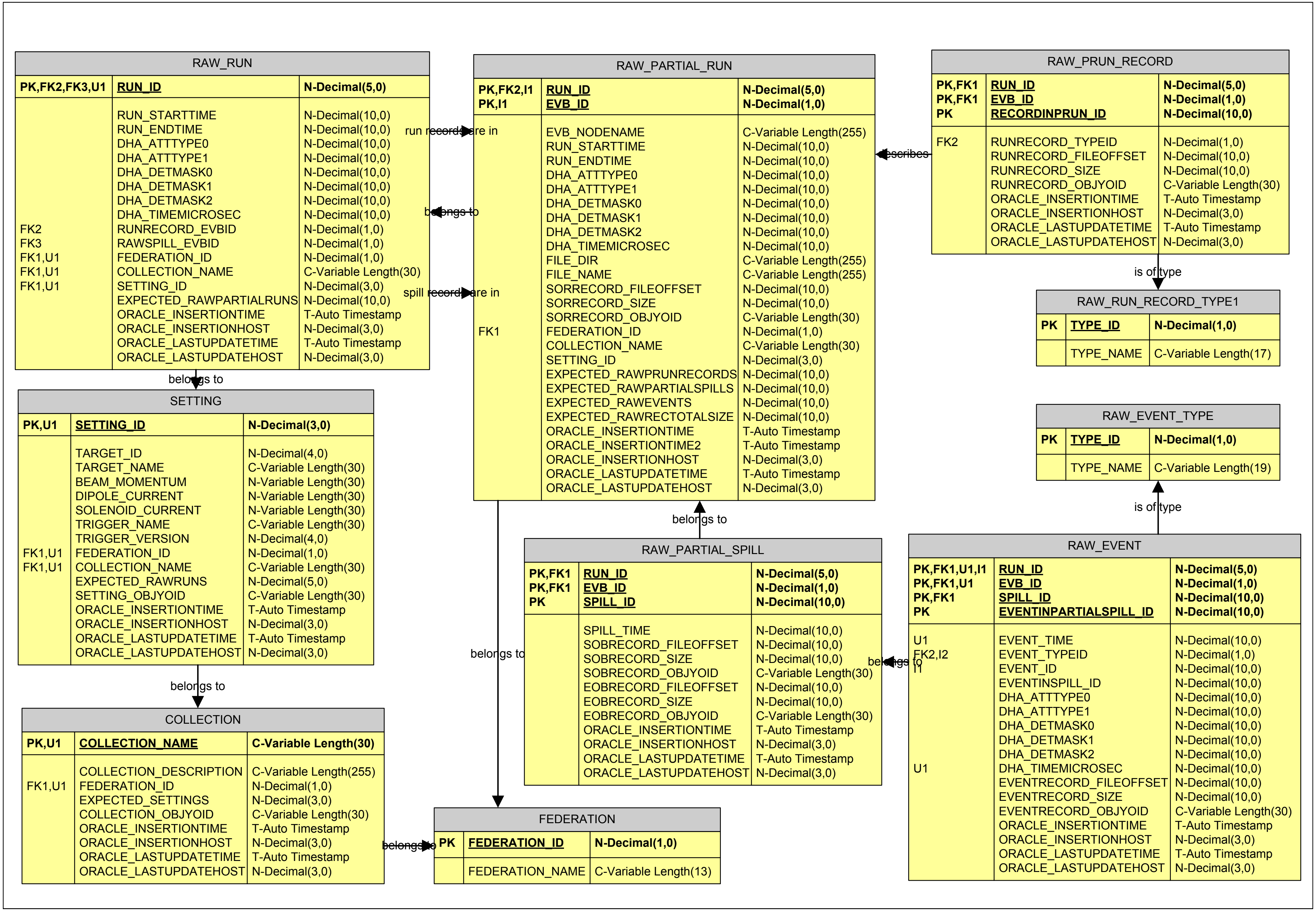
Event type (calibration, physics)

Federations contain collections

Federations contain partial runs

Second phase
- Add collections, settings, runs

First phase
- Self-contained subschema
- Only federations, partial runs, partial spills, events and the associated records and types


RAW_RUN
PK,FK2,FK3,U1 RUN_ID N-Decimal(5,0)
RUN_STARTTIME N-Decimal(10,0)
RUN_ENDTIME N-Decimal(10,0)
DHA_ATTTYPE0 N-Decimal(10,0)
DHA_ATTTYPE1 N-Decimal(10,0)
DHA_DETMASK0 N-Decimal(10,0)
DHA_DETMASK1 N-Decimal(10,0)
DHA_DETMASK2 N-Decimal(10,0)
DHA_TIMEMICROSEC N-Decimal(10,0)
FK2 RUNRECORD_EVBID N-Decimal(1,0)
FK3 RAWSPILL_EVBID N-Decimal(1,0)
FK1,U1 FEDERATION_ID N-Decimal(1,0)
FK1,U1 COLLECTION_NAME C-Variable Length(30)
FK1,U1 SETTING_ID N-Decimal(3,0)
EXPECTED_RAWPARTIALRUNS N-Decimal(10,0)
ORACLE_INSERTIONTIME T-Auto Timestamp
ORACLE_INSERTIONHOST N-Decimal(3,0)
ORACLE_LASTUPDATETIME T-Auto Timestamp
ORACLE_LASTUPDATEHOST N-Decimal(3,0)
run records are in
belongs to
spill records are in
belongs to
SETTING
PK,U1 SETTING_ID N-Decimal(3,0)
TARGET_ID N-Decimal(4,0)
TARGET_NAME C-Variable Length(30)
BEAM_MOMENTUM N-Variable Length(30)
DIPOLE_CURRENT N-Variable Length(30)
SOLENOID_CURRENT N-Variable Length(30)
TRIGGER_NAME C-Variable Length(30)
TRIGGER_VERSION N-Decimal(4,0)
FK1,U1 FEDERATION_ID N-Decimal(1,0)
FK1,U1 COLLECTION_NAME C-Variable Length(30)
EXPECTED_RAWRUNS N-Decimal(5,0)
SETTING_OBJYOID C-Variable Length(30)
ORACLE_INSERTIONTIME T-Auto Timestamp
ORACLE_INSERTIONHOST N-Decimal(3,0)
ORACLE_LASTUPDATETIME T-Auto Timestamp
ORACLE_LASTUPDATEHOST N-Decimal(3,0)
belongs to
COLLECTION
PK,U1 COLLECTION_NAME C-Variable Length(30)
COLLECTION_DESCRIPTION C-Variable Length(255)
FK1,U1 FEDERATION_ID N-Decimal(1,0)
EXPECTED_SETTINGS N-Decimal(3,0)
COLLECTION_OBJYOID C-Variable Length(30)
ORACLE_INSERTIONTIME T-Auto Timestamp
ORACLE_INSERTIONHOST N-Decimal(3,0)
ORACLE_LASTUPDATETIME T-Auto Timestamp
ORACLE_LASTUPDATEHOST N-Decimal(3,0)
belongs to
RAW_PARTIAL_RUN
PK,FK2,I1 RUN_ID N-Decimal(5,0)
PK,I1 EVB_ID N-Decimal(1,0)
EVB_NODENAME C-Variable Length(255)
RUN_STARTTIME N-Decimal(10,0)
RUN_ENDTIME N-Decimal(10,0)
DHA_ATTTYPE0 N-Decimal(10,0)
DHA_ATTTYPE1 N-Decimal(10,0)
DHA_DETMASK0 N-Decimal(10,0)
DHA_DETMASK1 N-Decimal(10,0)
DHA_DETMASK2 N-Decimal(10,0)
DHA_TIMEMICROSEC N-Decimal(10,0)
FILE_DIR C-Variable Length(255)
FILE_NAME C-Variable Length(255)
SORRECORD_FILEOFFSET N-Decimal(10,0)
SORRECORD_SIZE N-Decimal(10,0)
SORRECORD_OBJYOID C-Variable Length(30)
FK1 FEDERATION_ID N-Decimal(1,0)
COLLECTION_NAME C-Variable Length(30)
SETTING_ID N-Decimal(3,0)
EXPECTED_RAWPRUNRECORDS N-Decimal(10,0)
EXPECTED_RAWPARTIALSPILLS N-Decimal(10,0)
EXPECTED_RAWEVENTS N-Decimal(10,0)
EXPECTED_RAWRECTOTALSIZE N-Decimal(10,0)
ORACLE_INSERTIONTIME T-Auto Timestamp
ORACLE_INSERTIONTIME2 T-Auto Timestamp
ORACLE_INSERTIONHOST N-Decimal(3,0)
ORACLE_LASTUPDATETIME T-Auto Timestamp
ORACLE_LASTUPDATEHOST N-Decimal(3,0)
belongs to
belongs to
RAW_PARTIAL_SPILL
PK,FK1 RUN_ID N-Decimal(5,0)
PK,FK1 EVB_ID N-Decimal(1,0)
PK SPILL_ID N-Decimal(10,0)
SPILL_TIME N-Decimal(10,0)
SOBRECORD_FILEOFFSET N-Decimal(10,0)
SOBRECORD_SIZE N-Decimal(10,0)
SOBRECORD_OBJYOID C-Variable Length(30)
EOBRECORD_FILEOFFSET N-Decimal(10,0)
EOBRECORD_SIZE N-Decimal(10,0)
EOBRECORD_OBJYOID C-Variable Length(30)
ORACLE_INSERTIONTIME T-Auto Timestamp
ORACLE_INSERTIONHOST N-Decimal(3,0)
ORACLE_LASTUPDATETIME T-Auto Timestamp
ORACLE_LASTUPDATEHOST N-Decimal(3,0)
belongs to
FEDERATION
PK FEDERATION_ID N-Decimal(1,0)
FEDERATION_NAME C-Variable Length(13)
RAW_PRUN_RECORD
PK,FK1 RUN_ID N-Decimal(5,0)
PK,FK1 EVB_ID N-Decimal(1,0)
PK RECORDINPRUN_ID N-Decimal(10,0)
FK2 RUNRECORD_TYPEID N-Decimal(1,0)
RUNRECORD_FILEOFFSET N-Decimal(10,0)
RUNRECORD_SIZE N-Decimal(10,0)
RUNRECORD_OBJYOID C-Variable Length(30)
ORACLE_INSERTIONTIME T-Auto Timestamp
ORACLE_INSERTIONHOST N-Decimal(3,0)
ORACLE_LASTUPDATETIME T-Auto Timestamp
ORACLE_LASTUPDATEHOST N-Decimal(3,0)
describes
is of type
RAW_RUN_RECORD_TYPE1
PK TYPE_ID N-Decimal(1,0)
TYPE_NAME C-Variable Length(17)
RAW_EVENT_TYPE
PK TYPE_ID N-Decimal(1,0)
TYPE_NAME C-Variable Length(19)
is of type
RAW_EVENT
PK,FK1,U1,I1 RUN_ID N-Decimal(5,0)
PK,FK1,U1 EVB_ID N-Decimal(1,0)
PK,FK1 SPILL_ID N-Decimal(10,0)
PK EVENTINPARTIALSPILL_ID N-Decimal(10,0)
U1 EVENT_TIME N-Decimal(10,0)
FK2,I2 EVENT_TYPEID N-Decimal(1,0)
I1 EVENT_ID N-Decimal(10,0)
EVENTINSPILL_ID N-Decimal(10,0)
DHA_ATTTYPE0 N-Decimal(10,0)
DHA_ATTTYPE1 N-Decimal(10,0)
DHA_DETMASK0 N-Decimal(10,0)
DHA_DETMASK1 N-Decimal(10,0)
DHA_DETMASK2 N-Decimal(10,0)
U1 DHA_TIMEMICROSEC N-Decimal(10,0)
EVENTRECORD_FILEOFFSET N-Decimal(10,0)
EVENTRECORD_SIZE N-Decimal(10,0)
EVENTRECORD_OBJYOID C-Variable Length(30)
ORACLE_INSERTIONTIME T-Auto Timestamp
ORACLE_INSERTIONHOST N-Decimal(3,0)
ORACLE_LASTUPDATETIME T-Auto Timestamp
ORACLE_LASTUPDATEHOST N-Decimal(3,0)

Logical schema design: primary keys

- **Define primary keys whenever appropriate (almost always!)**
  - All Harp tables have a primary key
  - They make the schema more understandable (even to the developer!)
  - *They automatically imply the creation of an index for faster access*

- **Primary keys can span more than one column**
  - For Harp runs to events: follow the hierarchical data model
    - Runs: (run#)
    - Partial runs: (run#, evb#)
    - Partial spills: (run#, evb#, spill#)
    - Events: (run#, evb#, spill#, event-in-partial-spill#)
  - *The most common HARP read access pattern (via run#) uses these indexes*

- **Choose meaningful IDs (but you may use system assigned IDs too)**
  - For Harp events
    - From DAQ: run# (e.g. 123), event-builder# (e.g. partial run 123-0 or 123-1), spill#
    - System assigned (well, assigned by the C++ developer): event-in-partial-spill #

Logical schema design: integrity constraints (foreign/unique keys)

- **Define integrity constraints whenever appropriate**
  - Almost always, if you have more than one relational table you want to JOIN!
  - All Harp tables are related via integrity constraints (foreign keys)
  - They make the schema more understandable (even to the developer!)
  - *They force the database to perform data consistency checks*
- **Foreign keys can span more than one column**
  - For Harp runs to events: follow the hierarchical data model
    - E.g. partial spill (run#, evb#, spill#) belongs to partial run (run#, evb#)
- **Foreign keys may reference either primary or unique keys**
  - For Harp, some *redundant constraints* are used because some redundant columns are present (schema is not fully normalized), e.g. collection information
- **May define foreign key constraints now and enable/validate them later**
  - E.g.: partial run (run#, evb#) belongs to (references) run (run#), but the run table was filled only in the second phase of the migration

Users and roles

- **Two (sets of) accounts**
  - One with full privileges for writing: data owner (DDL) and writer (DML)
    - Actually two: HARPTESTFDRAW (2001 data) and HARPPHYSFDRAW (2002)
    - *Here I was both the schema owner and the writer: in other situations, it may even be better to use different accounts if the owner and the writer are distinct persons*
  - One with read-only privileges for analysis: SELECT only
    - A single account HARPUSER with read privileges on both of the above

- **Granting (e.g. read-only) privileges**
  - Either directly, or you may grant privileges to a role, and the role to a user
    - Role HARPREADER is granted SELECT privileges on all tables of HARP*RAW
    - User HARPUSER is granted the HARPREADER role (by the DBA)
  - Privileges must be granted by the table owner <u>for each relevant table</u>
    - HARP*RAW grant SELECT privileges to HARPREADER for each relevant table
    - At CERN *we do not like to grant "SELECT ANY" privileges*

Physical schema design: table and index partitioning

- **Use range partitioning by run# for the Run/PRun/PSpill/Event tables**
  - *The run# is a variable used in ALL access patterns to the Harp data*
  - Range partitioning is much more natural than hash or list partitioning here
- **Apply the same partitioning to all PKs and indexes: use local indexes**
  - *Performance: when selecting events for a run#, only one index partition is used*
  - Partitioning on first column of composite PK ensures that PK is *globally unique*
- **Store each partition in a separate tablespace**
  - Approximately 30 * 2 (data+index) partitions with 3 to 4 GB in each
  - Manageability: original idea was to handle them as transportable tablespaces
    - Minor technical issue: materialized views are treated as simple tables in this case
    - More importantly: the need to transport data has never occurred (yet?)
- **Was it really necessary? Maybe not, but *here* it does not harm!**
  - Performance? Largest help comes from indexing, but there may only be some extra (though not very large) benefit from use of partitioned (local) indexes
  - Manageability? Data rather static (no need to bring partitions online/offline from tape backup) and not distributed outside CERN, but this was a useful test
  - In summary: use partitioning wisely, it is not a synonym for faster and better!

Summary information: views and materialized views

- **Use views to hide the complexity of JOIN and GROUP BY queries**
  - Simplify developers' lives: views (virtual tables) can be queried like real tables
  - Ex. 1: 'spill' view merges information for two 'partial spills'
    - JOIN on the 'run' and 'partial spill' tables
    - No need to write a separate 'spill' table: *all the information is already there!*
  - Ex. 2: 'extended (partial) run' views hold summary info for each (partial) run
    - GROUP BY queries on the 'event' table determines #events in each partial spill/run
    - Additional GROUP BY and JOIN queries merge together all relevant information

- **Speed up read access to (most of) these views via materialized views**
  - For HARP these were built (refreshed) only once after the Objy migration
    - The HARP data are now read-only, no updates are foreseen
  - Technical issue: you must enable global query rewrite to profit from m. views
    - You query views or base tables, the optimizer rewrites them as queries on m. views
  - Perform complex JOIN and GROUP queries only once and store the results
    - Essential for summary information that is frequently accessed and heavy to compute
  - Materialized views can be partitioned and indexed too (for HARP, they all are)

RAW_PARTIAL_RUN_EXTENDED
RUN_ID
EVB_ID
EVB_NODENAME
RUN_STARTTIME
RUN_ENDTIME
DHA_ATTTYPE0
DHA_ATTTYPE1
DHA_DETMASK0
DHA_DETMASK1
DHA_DETMASK2
DHA_TIMEMICROSEC
FILE_DIR
FILE_NAME
SORRECORD_FILEOFFSET
SORRECORD_SIZE
SORRECORD_OBJYOID
FEDERATION_ID
COLLECTION_NAME
SETTING_ID
EXPECTED_RAWPRUNRECORDS
EXPECTED_RAWPARTIALSPILLS
EXPECTED_RAWEVENTS
EXPECTED_RAWRECTOTALSIZE
ORACLE_INSERTIONTIME
ORACLE_INSERTIONTIME2
ORACLE_INSERTIONHOST
ORACLE_LASTUPDATETIME
ORACLE_LASTUPDATEHOST
RAWPRUNRECORDS_NUMBER
RAWPRUNRECORDS_RECSIZE
RAWPARTIALSPILLS_NUMBER
RAWPARTIALSPILLS_RECSIZE
RAWEVENTS_NUMBER
RAWEVENTS_RECSIZE
RAWPHYEVENTS_NUMBER
RAWPHYEVENTS_RECSIZE
RAWCALEVENTS_NUMBER
RAWCALEVENTS_RECSIZE

```
create view raw_partial_run_extended
as
select
rpr.run_id,
… ,
rpr.oracle_lastupdatehost,
rprrpr0.rawprunrecords_number rawprunrecords_number,
rprrpr0.rawprunrecords_recsize rawprunrecords_recsize,
rpspr0.rawpartialspills_number rawpartialspills_number,
…,
rpspr0.rawcalevents_recsize
from
raw_partial_run rpr,
raw_prun_record_byprun0 rprrpr0,
raw_partial_spill_byprun0 rpspr0
where rprrpr0.run_id=rpr.run_id
and rprrpr0.evb_id=rpr.evb_id
and rpspr0.run_id=rpr.run_id
and rpspr0.evb_id=rpr.evb_id
order by run_id, evb_id
```

CREATE VIEW AS JOIN OF 1 TABLE AND 2 VIEWS

CREATE MATERIALIZED VIEW AS SELECT FROM 1 VIEW

```
create materialized view raw_partial_run_extended_mv
partition by range (run_id) (…)
build immediate refresh on demand enable query rewrite
as
select * from raw_partial_run_extended;

create index rawpartialrunext_mv_indx
on raw_partial_run_extended_mv (run_id, evb_id) local
tablespace …;
```

CREATE VIEW AS JOIN OF 2 VIEWS

```
select
rr.run_id,  …, rr.rawspills_recsize,
rpr_sum.rawprunrecords_number,  …, rpr_sum.rawcalevents_recsize
from raw_run_withrecspi rr, raw_partial_run_byrun rpr_sum
where rr.run_id=rpr_sum.run_id
order by rr.run_id
```

RAW_RUN_EXTENDED

RUN_ID
RUN_STARTTIME
RUN_ENDTIME
DHA_ATTTYPE0
DHA_ATTTYPE1
DHA_DETMASK0
DHA_DETMASK1
DHA_DETMASK2
DHA_TIMEMICROSEC
RUNRECORD_EVBID
RAWSPILL_EVBID
FEDERATION_ID
COLLECTION_NAME

QUERY REWRITE:

QUERY ON RUN_EXTENDED VIEW IS EXECUTED VIA PARTIAL_RUN_EXTENDED_MV MATERIALIZED VIEW!

```
-- Query Plan: 01/20/2005 04:50:49 pm

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------

--------------------------------------------------------------------------------------------------------
| Id  | Operation                | Name                          | Rows  | Bytes |TempSpc| Cost  | Pstart| Pstop |
--------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                               |  5172 |  2363K|       |   758 |       |       |
|   1 |  VIEW                    | RAW_RUN_EXTENDED              |  5172 |  2363K|       |   758 |       |       |
|   2 |   MERGE JOIN             |                               |  5172 |  2338K|       |   758 |       |       |
|   3 |    VIEW                  | RAW_RUN_WITHRECSPI            |  5172 |  1661K|       |   504 |       |       |
|   4 |     SORT ORDER BY        |                               |  5172 |  1545K|  3320K|   504 |       |       |
|*  5 |      HASH JOIN           |                               |  5172 |  1545K|       |   265 |       |       |
|   6 |       PARTITION RANGE ALL|                               |       |       |       |       |     1 |     8 |
|   7 |        TABLE ACCESS FULL | RAW_PARTIAL_RUN_EXTENDED_MV   | 10959 |   149K|       |    58 |     1 |     8 |
|   8 |       VIEW               | RAW_RUN_WITHREC               |  5479 |  1562K|       |   197 |       |       |
|   9 |        SORT ORDER BY     |                               |  5479 |   567K|  1368K|   197 |       |       |
|  10 |         PARTITION RANGE ALL|                             |       |       |       |       |     1 |     8 |
|* 11 |          HASH JOIN       |                               |  5479 |   567K|       |   103 |       |       |
|  12 |           TABLE ACCESS FULL| RAW_PARTIAL_RUN_EXTENDED_MV | 10959 |   310K|       |    58 |     1 |     8 |
|  13 |           TABLE ACCESS FULL| RAW_RUN                     |  5805 |   436K|       |    37 |     1 |     8 |
|* 14 |    SORT JOIN             |                               |  5805 |   759K|  1832K|   254 |       |       |
|  15 |     VIEW                 | RAW_PARTIAL_RUN_BYRUN         |  5805 |   759K|       |   131 |       |       |
|  16 |      SORT GROUP BY       |                               |  5805 |   272K|  1272K|   131 |       |       |
|  17 |       PARTITION RANGE ALL|                               |       |       |       |       |     1 |     8 |
|  18 |        TABLE ACCESS FULL | RAW_PARTIAL_RUN_EXTENDED_MV   | 10959 |   513K|       |    58 |     1 |     8 |
--------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   5 - access("RR"."RUN_ID"="RAW_PARTIAL_RUN_EXTENDED_MV"."RUN_ID" AND
              "RR"."RAWSPILL_EVBID"="RAW_PARTIAL_RUN_EXTENDED_MV"."EVB_ID")
  11 - access("RR"."RUN_ID"="RAW_PARTIAL_RUN_EXTENDED_MV"."RUN_ID" AND
              "RR"."RUNRECORD_EVBID"="RAW_PARTIAL_RUN_EXTENDED_MV"."EVB_ID")
  14 - access("RR"."RUN_ID"="RPR_SUM"."RUN_ID")
       filter("RR"."RUN_ID"="RPR_SUM"."RUN_ID")

Note: cpu costing is off
```

Note: partition-wise join (N instead of NxN)

PARTITION RANGE ALL is before HASH JOIN

C++ migration client: bind variables and bulk operations

- **Always use bind variables** in the SQL your client sends to the server
  - For write access
    - *'INSERT INTO RAW_RUN (RUN_ID, …) VALUES (:1, …);'* is GOOD!
    - *'INSERT INTO RAW_RUN (RUN_ID, …) VALUES (12345, …);'* is BAD!
  - For read access
    - *'SELECT * FROM RAW_EVENT WHERE RUN_ID = :1;'* is GOOD!
    - *'SELECT * FROM RAW_EVENT WHERE RUN_ID = 12345;'* is BAD!
  - *Do not waste the CPU and memory of the server* in parsing and optimizing SQL statements: do it only once and reuse the same statement with different values
- **Always use bulk operations** in client/server communication
  - *Minimize the number of client/server network round trips*
    - For both write (bulk insert/update) and read access (pre-fetched result set)
  - Using bulk insertion and bind variables was essential in HARP migration client
    - 4 separate (OCCI) C++ Statement instances to insert data into 4 different tables
- **Bind variables and bulk operations exist for many languages and drivers**
  - OCCI was used for HARP: very intuitive and user friendly API, excellent doc
  - The same concepts exist for OCI, ODBC, JDBC, and also for the POOL RAL

C++ end-user read access: let the database do the basic selection

- **HARP hierarchical event loop**
  - Example: “run this job on the runs belonging to the PHYSICS collection, with the beam colliding against the XXX target”
  - Objectivity implementation: retrieve all and loop in the C++ client memory
    - Retrieve (from server into C++ client memory) all settings in PHYSICS collection
    - *Loop (in C++ client memory) over all such settings, select those with target=XXX*
    - For each selected setting, retrieve all runs in it, etc etc.
  - Oracle implementation: let the database server do the selection
    - *Query database server for settings in PHYSICS collection AND with target=XXX*
    - Retrieve (from server into C++ client memory) only the selected settings
    - For each selected setting, retrieve all runs in it, etc etc

- **Reengineering of jobOptions-driven C++ event selection**
  - Objy: EventSelection class can be applied to an Event object (returns bool)
  - Oracle: EventSelection class can also ‘describe itself’ as a string
    - Selection algorithmic becomes technology independent, the same code can be executed against both backends (could still be improved to use bind variables…)
  - More details in the CHEP2004 poster and paper linked to the Workshop agenda

Useful tools (IMO)

- **Interactive analysis and SQL optimization: Benthic, TOra**
  - IMO, much better than sqlplus for interactive optimization of SQL queries
    - *Display the execution plan*, add indexes and materialized views, iterate again...
  - Use Benthic for Oracle on Windows, use TOra on Linux (or for MySQL...)
- **Batch execution of SQL statements: sqlplus**
  - There are many ways to create the schema (DDL SQL): interactively using TOra/Benthic/sqlplus, executing scripts in batch, in the C++ client, ...
    - *For HARP, all DDL operations were kept in SQL scripts and executed with sqlplus*
    - If you need Oracle/MySQL compatibility, you may choose to let RAL handle it in C++
  - Only one recommandation: *make sure you can rebuild your schema in a reproducible way* (include in your scripts/C++ all changes, eg indexes, views, ...)
- **Reverse engineering: MS Visio, Oracle Designer**
  - My *personal* approach: design the schema on paper, prototype/finalize the DDL SQL using Benthic/sqlplus, then make nice pictures from reverse engineering
    - The pictures you saw in this talk come from MS Visio
  - You could also design the schema from the start using a tool (e.g. JDeveloper)

Conclusions

- BLOBs can be kept outside the database if not queried
- Always use primary keys and foreign keys!
  - Additional indexes and constraints may also help
  - Think of the most common access patterns for both write and read
- Partitioning may help, but it is not always a necessity
- Views and materialized views make very useful summaries
- Always use bind variables and bulk operations!
- Ensure you can rebuild your schema in a reproducible way
- Oracle databases with TB's of data already exist at CERN